Using Semantic Distance as a Heuristic for Service Planning

Abstract

With a growing number of available services, each having slightly different parameters, preconditions and effects, automated planning on general semantic services becomes highly relevant. However, most exiting planners only consider PDDL, or if they claim to use OWL-S, they usually translate it to PDDL, losing much of the semantics on the way. In this paper, we propose a new domain independent heuristic based on *semantic distance* that can be used by generic planning algorithms such as A* for automated planning of semantic services described with OWL-S. For the heuristic to include more relevant information we calculate the heuristic at runtime. Using this heuristic, we are able to produce better results (fewer expanded states) in less time than with established domain independent techniques.

1 Introduction

We motivate our work by the need of a heuristic for AI planning. Since the search space of domain-independent planners for large problems becomes computationally intractable (Kambhampati and Hendler 1992) we need heuristics to guide our search through the state space.

For domain-specific planners that have a special purpose (e.g., finding a route from one place to another for a GPS traffic guidance systems), a heuristic can easily be provided e.g. the Manhattan-Distance or the Euclidean distance. But for an agent which has the capability of creating general plans, these heuristics are not sufficient. This means it is impossible for our general purpose planner to create a problem specific heuristic at design time.

Even reusing old ones like it is done for meta-heuristics or learning parameters of hyper-heuristics have only been successfully applied to simple problems (Osman and Laporte 1996). Meta-heuristics or hyper-heuristics have an additional drawback: they need a learning phase to gather information about the problem to be solved.

The calculation of the heuristic during runtime is motivated by the additional information available like the grounding information which could consist of concrete individuals to abstract classes describing e.g. the parameters of a service.

The creation of heuristics during runtime can lead to the encounter of new concepts used in an interface definition like a service description, which then lead us back to a fundamental question in AI research: **How can AI make sense of new concepts?** For heuristics this means interpreting the new concepts and adding information to classical heuristic approaches. A function $H : state \rightarrow \mathbb{R}^+$ is called heuristic (Russel and Norvig 2002, p. 92) and estimates the distance from a state to a given goal. We extend this definition of heuristic to $H : service \times state \times goal \rightarrow \mathbb{R}^+$ making the heuristic more dynamic since now it is able to adapt with changing goals and services. With that, the heuristic determines the usefulness of the given service in the current state regarding a current goal. This is done because if an alone state would be the information source for the heuristic, information like the service description would be lost.

The interested reader is referred to (Pearl 1985) for a formal description of heuristics and their properties. During our analysis of this topic, we have found that understanding the described functionality of a service is an AI-hard task (Yampolskiy 2012). This is because interpretation of what a description creator might have understood the service to be, might not be entirely reflected in the description. Furthermore, the service can have multiple interpretations in different contexts. Here the context we defined is the additional information relevant to our problem. As an example strategy for problem-solving using a heuristic, we have selected planning. This means our context consists of the start and goal state which include a domain description.

Starting from this setup, we need to evaluate if a capability is useful in the endeavour of finding a plan solving our problem.

The approach presented in in Figure 1 is a goal-oriented heuristic at runtime utilizing the semantics of the goal and capability description. The heuristic is thought for a one-shop-planning problem, where it is expensive for the agent to try out services since we are looking at possible world-altering capabilities, which means a learning phase should be kept as short as possible. This is done at runtime so that we know the goal we want to fulfill and can create a heuris-

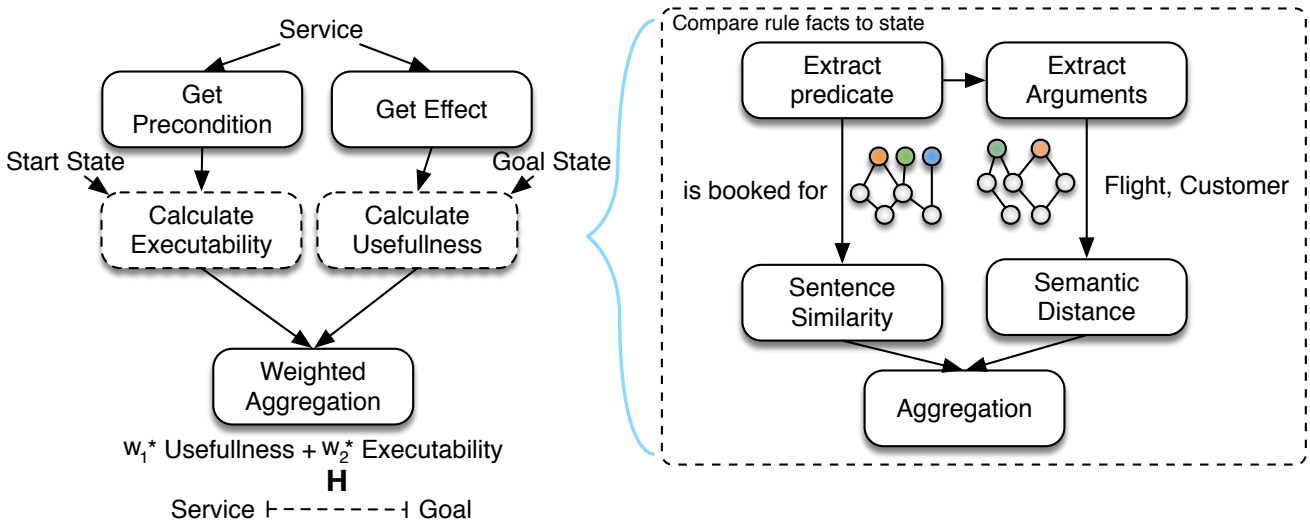

Figure 1: Abstract approach to a greedy heuristic

tic reflecting the given problem. We do so by looking at the goal and the capability description we encounter to calculate the usefulness of a capability. Here the idea of the heuristic is to find useful capabilities to try in our search to the goal state, and reprobate others. The heuristic additionally estimates how much of the precondition is fulfilled to see which capabilities are more likely to be executed. These two evaluations of the service are then combined to create our heuristic.

In this section we will first look at the state of the art of heuristics in Section 2. Then we create our goal oriented heuristic in Section 3, select a data set to test this heuristic on in Section 4 and discuss the results in Section 5. We conclude this experiment in Section 6.

2 State of the Art

The state of the art in general heuristics for the planning problem is limited. The main conference on AI Planning and heuristic search is the International Conference on Automated Planning and Scheduling (ICAPS) / Conference on Artificial Intelligence Planning Systems (AIPS) (ICA 2016). Here, starting from 1990, the last 28 years the community of AI planning has discussed the different approaches on problem solving.[1] During that research effort, multiple specializations of the general planning domain have been identified. Most of them use some kind of translation of the semantic domain to classical STRIPS planning in e.g. PDDL, like OWLS-Xplain (Klusch, Gerber, and Schmidt 2005; Klusch and Gerber 2006) or Simplanner (Kuzu and Cicekli 2012), or agent-based approaches as proposed in (Alves and Marchi 2017). A comprehensive overview can be found in the work of Markou and Refanidis (Markou and Refanidis 2016). Service composition approaches focus on Quality of Service, as in (Laleh et al. 2018), or use e.g. model checking as in (Du, Yang, and Hu 2018).

The different nature of a STRIPS-like planning problem without semantic and service planning address the problem of a search for a plan in different ways. Classical planners are highly optimized to solve problems like the 15-puzzle (Rat 1986) or the four-peg towers of Hanoi problem (Korf and Felner 2007). For semantic general-purpose planning, more general heuristics are needed.

There are heuristics like the minimal step count to the goal, called a **uniform action cost** (Pearl 1985). These heuristics is equal to the step count if an action cost is equal to 1 (Keyder and Geffner 2007). If the uniform cost function of 1, the heuristic is admissible since it predicts that the action to be executed is always only one step form the goal. This is sometimes also called an "optimistic" heuristic.

Greedy heuristics are those which count the overlap to the goal, thus measuring the usefulness of a service in how much of the goal it archives. This is a quite simple heuristic which performs well regarding its simplicity.

Haslum and Geffner (Haslum and Geffner 2000) describe a greedy heuristic which is derived from the STRIPS planning problems, executing the services with the highest overlap of their add-list and the goal first. This heuristic is admissible, it can be used for all planning problems, and is the basis of the most successful planners according to the International Planning Competition (Helmert, Röger, and Karpas 2011). However, it only works for the relaxed problem when the service effect does not have a delete-list, and only the add-list is added to the current state to create a new state.

The approach of (Haslum and Geffner 2000) has been extended by an optimization with an abstraction of the effects called **Patterns** (Haslum et al. 2007). Those patterns are then subtracted from each effect and the start and goal states, creating abstract states, which are then mapped to the state space through these patterns. The patterns represent subproblem of the original planning problem, which is

[1] See www.icaps-conference.org for the proceedings.

already solved, to get the patterns. A pattern is a set of variable assignments which reoccurs in different states, e.g. start and the goal state, creating homomorphism abstractions. The drawback of those **Pattern Database Heuristics (PDB)** is the that they do not scale up to real-world problems (Katz and Domshlak 2008).

The approach of Katz et al. (Katz and Domshlak 2008) again optimizes the result of (Haslum et al. 2007) by adding a **Causal Graph structure** and the **Domain transition graph** to the PDB heuristics, resulting in "causal graph structural patterns", which approximate the original problem but are more abstract. This abstraction in done with the SAS^+ formalization of the planning problems (Bäckström and Nebel 1995), which has an additional restriction for the domain descriptions. The simplification SAS^+ has the effect that abstractions like those done by Katz et al. are possible. Here e.g. "Post-uniqueness" means that an effect is only given by at most one action. Additionally, the "Binariness" restriction demands that all state variables have exactly two possible values. With an open world assumption and a distributed service development we cannot fulfill those restrictions, thus these results cannot be applied to our problem.

Learning the domain structure by observing plans is still subject to research. Gregory and Lindsay (Gregory and Lindsay 2016) propose a model of action cost, which is learned through the observation of plan traces. Even though the resulting cost function can be used as a heuristic, its creation, the observation of executed plans, puts this heuristic into the runtime. This interleaving of the plan- and runtime is out of scope for this work because we want to study the understanding of services and measure the degree of understanding by their use in a plan, not the other way around. Despite this being a valid approach, the idea is a trial and error mechanisms of learning the usefulness of services. For certain services, this might be appropriate, but we restrict our domain to intellectual problems where deterministic actions are analyzed.

The same argument can be applied with approaches leaning for other planning properties to plan optimality(Nedunuri, Cook, and Smith 2011). Other research on heuristic creation in uncertainty like (Marinescu and Coles 2016) is concentrated on numerical effects, which we neglect here.

Another approach of guiding the search through the state space is called Landmarking (Karpas and Domshlak 2009). **Landmarks** are facts which must hold true on the way towards reaching a goal. The two sub-problems of landmarks are: How to find landmarks (Porteous, Sebastia, and Hoffmann 2014), and how to use the information given by a landmark to create a heuristic (Zhang, Wang, and Xie 2015). These landmarks then can be used to decompose the search problem and use a local search to iteratively search from landmark to landmark (Karpas and Domshlak 2009).

Using landmarks for creating a heuristic is done in the LAMA planner from Richter and Westphal (Richter, Helmert, and Westphal 2008), which performed well in the IPC 2008 (Karpas and Domshlak 2009), counting fulfilled landmarks in contrast to unfulfilled ones.

In (Richter, Westphal, and Helmert 2011) they combine this greedy heuristic search with landmarks with preferred operators that take into account the usefulness of services by keeping them in a "preferred-operator queue". Those preferred operators are in consequence always tried first. Deciding which service is preferred is part of the heuristic. Again, the problem is formalized as a SAS^+ problem, which lets Richter et al. decide which service is a landmark (because its effect is unique). This is not given in our planning problem, thus this kind of heuristic needs adaption to be able to function with, e.g., the open world assumption.

As a conclusion, the state of the art in generating heuristics is mainly based on the relaxation of an original problem by abstraction. Some of them use the domain description to structure the search space, others analyze services to identify landmarks which help to break down the search problem. But no heuristic found so far has used the semantics of the planning problem. Thus, those heuristics are often not applicable to general planning in real-world problems.

3 Semantic Heuristics

In this work, we propose a new heuristic for service planning using the *Semantic Distance* between states. Since we are not considering services with different costs, the here looked at planning problem is a constraint satisfaction problem. As a planning algorithm, we used a variation of an A^* algorithm, due to its theoretical properties (see (Pearl 1985)). The variation from the basic A^* is that the function g from the cost function to be minimized $f = g + h$, where h is the used heuristic and g is supposed to be the cost of the path so far, is selected as $g = \sum_{i=0}^{n-1} h(s_i)$, where s_0 is the start state, s_n is the current state, and all states s_i are on the path to the current state.

In order to determine the semantic distance between two states, we first have to perform a *Semantic Decomposition* of those states. We then use a *Marker Passing* algorithm to determine the semantic distance.

Semantic Decomposition Algorithm

At first, we will have a look a the semantic decomposition algorithm. The decomposition takes a word and looks up definitions and relations of this work in data sources like Wikipedia, WordNet or other dictionaries. The words related to the original word are then decomposed recursively until a predefined decomposition depth is reached. With that the connectionist interpretation of meaning is represented as a resulting graph.

The functions $AddRelation$ and $AddConcept$ are convenience methods for adding the relation and concepts into the semantic graph. The functions *AddConcept(concept, decomposition)* and *AddRelation(relation, GetTargets(relation), decomposition)* add the concepts or relations to the graph which represents our decomposition. *AddConcept* adds the given concept to the graph nodes and *AddRelation* adds the relation between the concept its targets to the relations of the graph.

We now have a look at how such decomposition can be created and how automatisms might help. We identified the steps for a decomposition as described in the recursive Algorithm 1. The algorithm takes as input the concept that is

Algorithm 1 A decompositioning algorithm.

Name: Decompose **Input:** Concept word, Integer depth
Output: SemanticGraph

```
 1: decomposition ← ∅
 2: function DECOMPOSE(c, depth, decomp)
 3:     if depth ≥ 0 ∧ c ∉ decomp then
 4:         c ← Normalization(c)
 5:         relations ← GetRelations(c)
 6:         definitions ← LookUpDefinitions(c)
 7:         AddConcept(c, decomp)
 8:         if IsSynonymOfPrime(c) then
 9:             return
10:         end if
11:         for all r ∈ relations do
12:             AddConcept(r, decomp)
13:             AddRelation(r, GetTargets(r), decomp)
14:             DECOMPOSE(r, depth-1, decomp)
15:             for all target ∈ GetTargets(r) do
16:                 DECOMPOSE(target, depth-1, decomp)
17:             end for
18:         end for
19:         for all definition ∈ definitions do
20:             for all def ∈ definition do
21:                 AddConcept(def, decomp)
22:                 AddRelation("definition", {c, def}, de-
    comp)
23:                 DECOMPOSE(def, depth-1, decomp)
24:             end for
25:         end for
26:     else
27:     end if
28: end function
29: DECOMPOSE(word, depth, decomposition)
30: return decomposition
```

subject to the decomposition. As a successful decomposition will always build a graph, the semantic primes are the termination criterion for the recursion.

The Algorithm 1 reads as follows:

Line 1 initializes the semantic graph which we will build up during this algorithm and which represents the result at the end.

Line 2 to 28 represents the recursive function which is called on all decomposed concepts. This function adds the decomposition to the semantic graph initialized in Line 1. Which is called until the decomposition depth is reached or all concepts have been decomposed into semantic primes. We will build a hierarchical structure made up of concepts also referred to as lexical units. Those concepts include a lexical representation, the textual representation of a lexeme and a decomposition.

Line 3 checks if the concepts have been already decomposed or if the decomposition depth is reached. The decomposition depth is a parameter of the decomposition, which restricts the decomposition to an amount of relations to which the decomposition extends. The second part stops the decomposition of decomposing the same concepts over and over again. Additionally, the decomposition stops here, if a synonym of the concept has been decomposed previously. This is because if a synonym has been decomposed previously, its synonyms are added to the decomposition as well. Thus this synonym, which is supposed to get decomposed now, is already part of the decomposition and is not decomposed again.

Line 4 takes the concepts to decompose and normalizes them. Here the inflection is removed, revealing the stem of a concept. Furthermore a concept includes all its inflections (all concepts which can be created by applying grammatical forms to a concept like 'eating', 'ate', 'eaten'), all lexical paradigms for this concept (all concepts rooting from the same word stem like to 'dine', 'dinner') and all sub-categorization frames (like the valence which is the amount of parameters like 'ask', 'ask X', 'ask X for Y'). We remove this kind of inflection because we are interested in the concepts described by a word, not its relation to other words. We can integrate syntactic information into the graph by adding syntax relations and nodes. For this reduction, we use the linguistic process of **Lemmatization**.[2] The function Normalization in Algorithm 1 Line 4 hides this normalization of a concept.

Line 5 gets all the relations of the concept from the used dictionaries. This means we are looking through all our dictionaries and look up all the semantic relations we can find and remember them for later processing.

Line 6 likewise looks up the definitions of the concept in all available dictionaries.

Lines 8 to 10 check whether the concept itself is a semantic prime. If this is the case, the prime is added to the decomposition, the decomposition is finished for this concept and returned. This hides technical optimizations like that we check for synonyms of primes as well to make the search a bit broader. At the same time, we simplified the stop word removal here. Stop words represent words which can be ignored, taken from natural language processing theory (Wilbur and Sirotkin 1991). These are mostly words with little semantic meaning like, e.g., 'a', 'an' or 'the'. Those nodes are removed and are not further decomposed.

Lines 11 to 18 handle the relation of the concept we are decomposing. Here all relations are added to the decomposition as a relation between concepts. Then all concepts which are connected by those relations are recursively decomposed.

Lines 19 to 25 decompose the definitions. Each definition is a list of concepts which get decomposed again. The definition is connected to the definiendum via a "definition" relations.

Marker Passing Algorithm

This Marker Passing algorithm is a generalization of the algorithm described by *F. Crestani* (Crestani 1997, Figure 5,

[2]Sometimes Lemmatization is referred to a Stemming, where, e.g., suffixes of words are removed, like a plural s.

p. 461). Crestani describes the Marker Passing in four steps: Pre-adjustment, spreading, post-adjustment and termination condition evaluation. This is quite general and can result in inaccurate interpretations of the algorithm. Consequently, we introduce a more precise description of the algorithm by breaking the activation down into multiple steps without losing generality.

Crestani's algorithm is based on the following principle: Starting from a start activation, a concept has a threshold (seen as an upper limit of activation in a node to decide if the node is activated), with each incoming activation the activation level of the node builds up. If the threshold is reached, the node is selected as activated and is spreading in the next spreading step. This means that the node passes all its markers on to its neighbors. This step is repeated until a termination condition is reached (Crestani 1997).

Algorithm 2 Marker Passing Algorithm

Name: MarkerPassing
Input: NodeData M **Output**:NodeData

1: $pulse_{out} \leftarrow$ Map$\langle Concept, (Edge, Markers)^* \rangle$
2: **for all** srcC $\in$ getPassingConcepts(M) **do**
3: $\quad pulse_{out}[srcC] \overset{\cup}{\leftarrow}$ outFunction(M, srcC)
4: **end for**
5: $pulse_{in} \leftarrow$ Map$\langle Concept, (Edge, Markers)^* \rangle$
6: **for all** $e \in pulse_{out}$ **do**
7: $\quad pulse_{in}$[e] $\overset{\cup}{\leftarrow}$ edgeFunction(M, e, $pulse_{out}[e]$)
8: **end for**
9: **for all** tgtC $\in pulse_{in}$ **do**
10: $\quad M \leftarrow$ inFunction(M, tgtC, $pulse_{in}$[tgtC])
11: **end for**
12: **for all** srcC $\in$ getPassingConcepts(M) **do**
13: $\quad M \leftarrow$ afterSend(M, srcC)
14: **end for**
15: **return** M

Algorithm 2 describes our extension of the spreading activation algorithm of Crestani (Crestani 1997).

The algorithm defines two maps $pulse_{out}$ and $pulse_{in}$, which hold the markers passed during a pulse. The function "$\overset{\cup}{\leftarrow}$" describes the insertion of the remaining tuple into the appropriate set of, e.g., all markers of the current pulse (in contrast to replacing them). In line 3 of Algorithm 2 we add the result of the $outFunction$ in the form of $(Relation \times Edge \times Markers)^*$ to the map $pulse_{out}$, where for each edge the markers are sorted.

We separate the Algorithm 2 into four blocks each consisting of one loop:

Lines 2 – 4: All passing concepts activate their out-function and the result to the current pulse stored in the variable $pulse_{out}$. This is the input for the edge functions of the appropriate relations of the next step.

Lines 5 – 9: Each marker passed by the current pulse is given to the appropriate relation it is passed to, and this relation activates its edge-function. The result of the edge-function is added to the pulse which is used as input for the in-functions of the targets of this relations.

Lines 10 – 12: Concepts that are targets of the relations passing markers are given the markers passed to them and activate their in-function.

Lines 13 – 15: The after-send-function is activated to fix the markers on the source concepts if needed.

Semantic Distance Heuristic

With this marker passing, we then can set start markers e.g. onto the start state and the goal state and analyze how the pass to services. As depicted in Figure 1 our goal-oriented heuristic is composed of two parts: The closeness to the start state and the closeness to the goal state.

$$H(S, S_0, G) = 1 - \frac{w_1 \cdot UF(S, G) + w_2 \cdot E(S, S_0)}{2}$$

Here the set S denotes all services, S_0 denotes the start state and G describes the goal state. UF is the *usefulness* of the given state w.r.t. the goal, and E its *executability*, and $w_{1,2}$ are their weights. The marker information and the detailed parameters, like termination condition or weight configurations of the marker passing algorithms, can be found in (Fähndrich, Weber, and Ahrndt 2016).

We selected these two measurements for our heuristic, because if we leave out one of the aspects two effects happen:

Goal overcommitment If we only look at the usefulness the services fulfilling subgoals will be tried first. Even though the probability of them being at the end of the plan is higher. This means if we are not talking about a planning problem which is trivial because all service in the plan is independent, that one or more service needs to be executed to enable this useful service. By only looking at the usefulness the search will always try those service first.

Low hanging fruits If we only look the executability, services which are executable are always tried first. This is good at the beginning of the planning process because reaching the goal is less probable at this point. But the more service is executed, the more service preconditions become enabled and all of them are tried first. To reach the goal then becomes like a breadth-first-search, where all service possible are tried before we get closer to the goal.

This argumentation leads us to introduce two weights $w_{1,2}$ which can be adapted depending on how far the search has progressed towards the goal. In the beginning, the executability should, in consequence, be highly weighted and become less important the closer to the goal the search progresses. This is inverse for the usefulness.

Both parts use the same kind of mechanism to check whether a fact is fulfilled (in the precondition or effect of a service) which is given by the goal or start state. To check this fulfillment we extract the predicates from the service precondition (effects) and the start (goal) axioms and their arguments and compare them. The comparison is done in two ways: first, for the predicates, we separate the word included in the predicate e.g. "IsBookedFor(Flight x,

Customer c)" become the predicate "is booked for". Since this resembles a sentence, the *sentence similarity measure* d_{sen} is used to compare predicates with a sentence similarity measure based on the semantic similarity presented in (Fähndrich, Weber, and Ahrndt 2016). Second we compare the arguments with the same *semantic similarity measure* d_{sem}. The result of those both similarity is then aggregated. The aggregation is done in the following way:

The main difference of the sentence similarity measure d_{sen} to the semantic distance measure d_{sem} is that the marker carries the information from which sentence they started out from. This information is used in the interpretation of the markers in the way described by Equation 3.

$$d_{sen} = \Xi + \left(\frac{\text{AVGACTIVATION}(result)}{|s_1 \cap s_2| \cdot \textbf{ StartMarkers}} \right)$$

Where s_1 and s_2 are two lists of concepts (the sentences) and Ξ represents the activation of the set of concepts in both sentences. **StartMarkers** is the set of initially placed markers. The $result$ is the marked graph after the marker passing has finished passing markers.

$$\Xi = 2 \frac{|s_1| + |s_2| - |s_1 \cup s_2|}{|s_1| + |s_2|}$$

The $AvgActivation$ gets the average of the activation of all markers of all concepts that are activated by both sentences. In Equation 3 we calculate the concepts present in both sentences plus the average activation of the concepts activated by markers of both sentences normalized by the total activation present after the initial marking. The resulting similarity is then again normalized to the interval from 0 to 1.

With Equation 3 we calculate the similarity of two sentences by calculating the ratio of equivalent words in both sentences in Ξ. This means that if the two sentences are equivalent, then Ξ becomes 1. To this ratio, we add the normalized average activation of all concepts activated by markers of both sentences. This captures that if concepts are semantically closer together, then more markers of both sentences, carrying more activation exist. In extreme cases, this value can become larger than one, which makes a normalization to the interval of zero to one of the result necessary.

These measures, d_{sen} and d_{sem}, are used to calculate the *Distance* between two states, which in turn is used to calculate the usefulness *UF* and executability *E*.

$$Distance : [Predicate] \times [Predicate] \rightarrow \mathbb{R}$$

$$Distance(A, B) = \frac{\sum\limits_{a \in A} \max\limits_{b \in B} w_{prd} \cdot d_{sen}(a, b) + X(a, b)}{|A|}$$

with

$$X(a, b) = \frac{\sum\limits_{a' \in args(a)} \max\limits_{b' \in args(b)} w_{arg} \cdot d_{sem}(a', b')}{|args(a)|}$$

$$UF(S, Goal) = Distance(Goal, S.eff)$$

Here we sum up the maximal weighted name and arguments match over the set of subgoals. Thus our service gets a usefulness of 1 if it fulfills all subgoals. The argument matching follows the same structure: The arguments of the goal predicate are matched to the arguments of the effect predicate. The maximal match is then summed up over all predicates of the goal. This means we are collecting all predicates of the goal which are semantically close to the effect of the services. The semantic closeness is calculated in parts: with the predicate name and its arguments. These two parts are weighted to define their influence on the overall heuristic result.

This is done because we want to maximize the argument matches and maximize the number of effects the service can fulfill for the given goal. The d_{sen} and the argument matches are then weighted with weights w_{prd}, w_{arg} determining how much influence the different similarities have in the overall result.

For the comparison of predicates two kinds of similarity measures are used:

Predicate comparison is done with the sentence similarity measure d_{sen} based on the semantic similarity measure proposed in (Fähndrich, Weber, and Ahrndt 2016). This is because a predicate mostly describes verbs and their form, direction and if they are passive or active. Our example "is booked for" is a typical use case for a predicate in ontologies.

Argument comparison is done with the semantic similarity measure d_{sem} proposed in see (Fähndrich, Weber, and Ahrndt 2016). Here the arguments are compared and the maximum is summed up. This makes the argument comparison independent of argument order.

The result is then normalized w.r.t. the number of predicates in the goal. Here we can see that $H(S, Z)$ becomes 1.0 for a service fulfilling all predicates of the goal, and 0.0 when none of the effects fulfill anything from the goal.

The executability (E) then is calculated with a similar measure then the usefulness:

$$E(S, Start) = Distance(s.pre, Start)$$

The same measure as for the usefulness can be applied for the evaluation of services to the extent of their fulfilled preconditions, meaning that service with unsatisfied precondition will be avoided. The effect of this heuristic is that the search algorithm of our planner now will try all services fulfilling parts of the goal first if the weight on the usefulness is high, and try executable service first if the weight of executability is higher.

Now we need an example problem to test our heuristic upon. This will be discussed in the Section 4 next.

4 Data Set

There is one dataset which uses semantic service descriptions we could find, called the **Secure Agent-Based Perva-**

sive Computing (Scallop) domain[3]. It has a collection of 21 services in the domain of health, air travel, and medical transport, and includes a set of 24 ontologies building up the domain model. Here, the scenario in focus is the medical transport of victims to a medical facility mostly by airplane and some ground transport. For technical compatibility we have translated the services and the domain to OWL-S 1.2.

The problem to be solved is to transport a patient from one place to a hospital. This includes finding the nearest airport from which one can fly to an airport that is close to a hospital. In addition, transport from and to the airports has to be organized. To book a flight, a flight account has to be created, a fitting flight has to be found, and the flight needs to be booked. After the flight a ground transport to the nearest hospital needs to be organized. Having done this, the goal of our example problem is reached.

We created a start and end state of this domain in which a victim has to be transported to a medical destination. The goal state consist of 64 axiomatic facts which need to be fulfilled to reach the goal state. Since the start and goal states have a large overlap, Figure 2 shows both states combined. Here the red states are states from the start state, and blue nodes are from the goal state; the gray nodes are found in both states.[4] For readability, the top-most "owl:Thing" class is omitted, and subclass relations are shown as dashed lines and individuals with dotted lines.

The overall domain is modeled in an ontology, describing the individuals and their relations. This ontology is too big to be displayed as a whole here, but the interested reader can download it from the Scallop project website.

The initial state has declares multiple facts about the domain, e.g. the transports available in our domain, where we see that Vehicle Transports and Flights are the two possible transports.

We want to be at a certain location at a certain time. All the modeling around those facts is necessary because we have to make sure all individual are available so that we can evaluate a potential execution of a service and with that reason upon the effect of this service.

The goal state consist of the information we want to see fulfilled. We start out by declaring the individuals we need in Figure 2 with the blue nodes. This models our 'patient zero' who needs the "RequiredTreatmenat" that can be provided at a certain hospital. As part of the goal we state that we wish the flight to be booked for "Patient 0", who owns the credit card the flight can be booked with and that the flight arrives at our target destination at the desired time. Since we need a valid account to book a flight, we want that "Patient 0" has a personal account to transport our patient and to book the relevant flights and transports from and to the airports. Next we model the individuals that are needed to reach our destination, as well as the departure and arrival airport.

When starting to plan, the start and goal state already overlap with 57 out of 64 axioms in the goal state, thus the plan to be made has to fulfill seven more axioms to reach the goal state. The optimal plan for this problem includes 4 steps: two requests for flight information for departure and arrival time, creating a flight account, and booking the flights.

The planning problem, with its 21 services, which have continuous inputs, like dates, has an infinite search space. Since the goal state is specified as a set of OWL axioms, there is a multitude of states which subsume our goal state. All of them are considered a success. The next section will elaborate on the results of our planner with the different heuristics.

5 Evaluation Results

The evaluation is run using the data set described in Section 4. To measure the performance, we count the extended nodes during the search for the goal state.

The 'gold standard' (described in Section 2) for general purpose heuristics is still a *Greedy* heuristic (Pearl 1985), which checks the overlap of the effects of a service with the facts wanted in the goal state, always selecting the service with the highest overlap. Additionally we compare our results with the *Uniform Cost* distribution, where each execution of a service is calculated to cost 1 abstract cost measure. With that admissible heuristic, A* finds an optimal solution. The *Random* heuristic has been added to the comparison as a baseline.

Table 1 shows the average results of ten runs. The four columns of Table 1 describe the mean μ and standard derivation σ of the experiment results of the steps and time. A step here is an extended state during the search for the goal state. Thus less looked at states means a more directed search and less effort. Column μ_{steps} indicates how many states the search had to extend on average to find the goal state, and column μ_{time} describes how much time one search has taken on average in seconds. The average has been created over 10 runs.[5] All heuristics are tested on the same start and goal state, thus there is no difference in the planning problem state space the search had to traverse. The variation of the performance is rooted in the random selection of services if two services are ranked with the same usefulness.

Table 1: Planning results, averaged over ten runs.

Heuristic	μ_{time}	σ_{time}	μ_{step}	σ_{step}
Marker Passing	**120.0**	31.0	**20.3**	5.2
Greedy	292.0	51.9	43.3	6.0
Uniform Cost	325.4	27.5	51.4	2.6
Random	358.9	9.7	53.9	0.32

Concerning the overall result, our *Marker Passing* heuristic is at least twice as fast and looks at half the number of nodes as the other heuristics. In the next section, we will discuss the results and analyze what they mean.

[3] http://www.dfki.de/scallops/

[4] A node not appearing in the goal does not have to mean that that node is to be deleted, but just that it is not relevant for the goal.

[5] We have tried the Random experiment with 100 runs, but this did not yield any different result, thus we settled for just 10 runs for all algorithms.

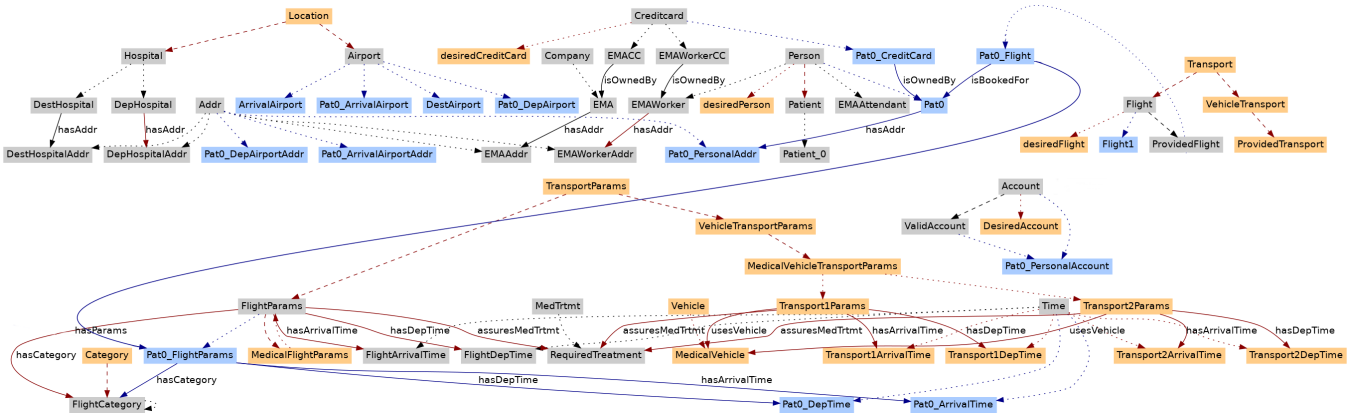

Figure 2: The Scallop health domain description as graph. Red nodes are found only the the start state, and blue nodes only in the goal state. Dashed lines are subclass and dotted lines individual relations.

6 Discussion of the results

We start by comparing the different heuristics:

The greedy heuristic is seen as the 'gold standard' for general purpose planning. It is used in most best-first search problems. As the name suggest, the services with the most overlap with the goal (the "best" ones) are tried first. This can lead to the result that in each state of the state space, the same "best" services are tried over and over again. The evaluation of the goal overlap does take about as much time as the semantic heuristic, as can be seen by comparing how many steps on average are looked at, and how much time is spent. Here we have an average step calculation time of 6.0 seconds for greedy, which is close to the 5.2 seconds the marker passing heuristic spends on each state. The standard derivation from the mean can be explained with the random selection of services with equal usefulness.

The Uniform Cost function with the same usefulness for all services creates a breadth-first search in an A^* algorithm. Here we can see that the standard derivation of the steps reduces to 2.6. This is because we are looking at all service in one state before progressing to the next; it does not matter in which order we look at the services.

The Random cost heuristic is the baseline to beat: If we are worse then this, our heuristic creates more confusion then it guides the search. In addition, this is used to ground the overall speed of our heuristics. This can be done, because the creation of a random number does almost consume as few resources as the Uniform Cost heuristic and does not give any information to the search. Here the standard derivation of steps is less then one, because the random heuristic does not speed up the search. As expected, the random heuristic has to look at the most state in the search space (see Table 1) which is at a mean of 53.9.

Of course, the relation between looking at more state and taking more time correlates. Thus, while the evaluation of the precondition and the instantiation of the effect take time, a uniform cost heuristic (which has no cost to calculate the heuristic) is still inefficient because each explored service takes some time. The use of semantics in the marker passing heuristic reduces the standard derivation from the mean, which means that we gather more useful information then by just comparing the overlap with the goal. The standard derivation from the mean is due to the random selection of services with the same heuristic value, which means that the heuristic is still not precise enough. This might change if grounded actions would be analyzed.

The problems used in academia are mostly formalized in PDDL with few semantics, e.g. input and output parameter type class hierarchies. Additionally, the problems are made 'hard' by scaling the problem up, e.g. by extending the 15-puzzle to an n-puzzle. In those toy domains, the services available are domain specific and are mostly necessary to solve the problem. Thus, the planning task is not a task of using the right services but rather to bring them in the right order. In contrast, in the general planning problem, the right services have to be selected to establish a domain. This domain includes the relevant services that help the agent to reach its goal.

This concludes our experiments. We will now take a step back and look at our results with some mental distance. Since the heuristic always returns a value between zero and one, it seems still admissible because until the goal is reached, at least one facts remains unsatisfied and thus at least one service need to be executed.

In addition the optimal selection of the weights w_1 and w_2 has to be analyzed further in future work. The heuristic seems to benefit of using semantic information, but it remains to be shown that the effort describing the services with additional semantic description is worth the effort. Further, we plan to use the executability as cost and adopt the goal by removing fulfilled subgoals for the calculation of the heuristic. Also, we are currently implementing a fast-forward planner by removing the reasoning for consistent states during the search.

References

Alves, J., and Marchi, J. 2017. Web Service Composition: An Agent-Based Approach. In *2017 Brazilian Conference*

on Intelligent Systems (BRACIS), 121–126. IEEE.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11(4):625–655.

Crestani, F. 1997. Application of Spreading Activation Techniques in Information Retrieval. *Artificial Intelligence Review* 11(6):453–482.

Du, Y.; Yang, B.; and Hu, H. 2018. Model checking of timed compatibility for mediation-aided web service composition: A three stage approach. *EXPERT SYSTEMS WITH APPLICATIONS* 112:190–207.

Fähndrich, J.; Weber, S.; and Ahrndt, S. 2016. Design and Use of a Semantic Similarity Measure for Interoperability Among Agents. In *Multiagent System Technologies*, 41–57. Springer International Publishing.

Gregory, P., and Lindsay, A. 2016. Domain model acquisition in domains with action costs. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, 149–157. AAAI Press.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proceedings of the 5th Internat. Conf. of AI Planning Systems (AIPS)*.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. *AAAI*.

Helmert, M.; Röger, G.; and Karpas, E. 2011. *Fast Downward Stone Soup: A baseline for building planner portfolios*. ICAPS 2011 Workshop on Planning and Learning.

2016. Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016. In Coles, A. J.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*. AAAI Press.

Kambhampati, S., and Hendler, J. A. 1992. A Validation-Structure-Based Theory of Plan Modification and Reuse. *Artif. Intell.* 55(2-3):193–258.

Karpas, E., and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. *IJCAI*.

Katz, M., and Domshlak, C. 2008. Structural Patterns Heuristics via Fork Decomposition. *ICAPS*.

Keyder, E., and Geffner, H. 2007. Heuristics for Planning with Action Costs. *CAEPIA* 4788(Chapter 15):140–149.

Klusch, M., and Gerber, A. 2006. Fast Composition Planning of OWL-S Services and Application. In *Fast Composition Planning of OWL-S Services and Application*, 181–190. IEEE.

Klusch, M.; Gerber, A.; and Schmidt, M. 2005. Semantic web service composition planning with owls-xplan. 55–62.

Korf, R., and Felner, A. 2007. Recent Progress in Heuristic Search - A Case Study of the Four-Peg Towers of Hanoi Problem. *IJCAI*.

Kuzu, M., and Cicekli, N. K. 2012. Dynamic planning approach to automated web service composition. *Applied Intelligence* 36(1):1–28.

Laleh, T.; Paquet, J.; Mokhov, S.; and Yan, Y. 2018. Constraint verification failure recovery in web service composition. *Future Generation Computer Systems* 89:387–401.

Marinescu, L., and Coles, A. 2016. *Heuristic guidance for forward-chaining planning with numeric uncertainty*. AAAI Press.

Markou, G., and Refanidis, I. 2016. Non-deterministic planning methods for automated web service composition. *Artif. Intell. Research ()* 5(1):14.

Nedunuri, S.; Cook, W. R.; and Smith, D. R. 2011. Cost-based learning for planning. *ICAPS 2011 Workshop on Planning and Learning*.

Osman, I. H., and Laporte, G. 1996. Metaheuristics: A bibliography. *Annals of Operations Research* 63(5):511–623.

Pearl, J. 1985. Heuristics. Intelligent search strategies for computer problem solving. *The Addison-Wesley Series in Artificial Intelligence*.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2014. On the Extraction, Ordering, and Usage of Landmarks in Planning. *Sixth European Conference on Planning*.

1986. Finding a Shortest Solution for the NxN Extension of the 15-PUZZLE Is Intractable. *AAAI*.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks Revisited. *AAAI*.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. *International Planning Competition 2011*.

Russel, S., and Norvig, P. 2002. *Artifical Intelligence: A Modern Approach*, volume 2. Prentice Hall.

Wilbur, W. J., and Sirotkin, K. 1991. The automatic identification of stop words. *Journal of information science* 18(1):45–55.

Yampolskiy, R. V. 2012. AI-Complete, AI-Hard, or AI-Easy - Classification of Problems in AI. *MAICS* 94–101.

Zhang, L.; Wang, C.-J.; and Xie, J.-Y. 2015. Cost optimal planning with multi-valued landmarks. *AI Communications* 28(3):579–590.