Learning a Heuristic Function
Automated Planning Term Project

Douglas Souza
Faculdade de Informática
Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681, Porto Alegre, RS
douglas.souza.002@acad.pucrs.br

Abstract

This paper aims to present the work developed for the final term project of the Automated Planing course. We propose the use of machine learning to learn a heuristic function to guide the planner search. We intend to achieve this result by exporting data of intermediate planning search, like literals, for example, and its associated cost to the objective state. Finally, this data is used to approximate a function that can be used as a new heuristic function. We expect this new function to present a better performance for the domain it was trained for compared to a standard heuristic function. We also show an approach to building such datasets used for training.

Introduction

We can observe a plan as a set of nodes. A node n can be viewed as a combination of the state s, its parent node, the action that resulted in this state, and the cost from the initial state to the current node. In this sense, we can relate the planing problem to a search problem, where we need to find the best path from the initial node to the goal node. In order to select the next node to visit, it is common the use of *heuristics*. Therefore, a heuristic is a function $h(n)$ that evaluates numerically each adjacent node to n (Ghallab, Nau, and Traverso 2004). Heuristics for node selection are often based on the *relaxation principle*: to define how desirable an adjacent node is, it simplifies the problem by relaxing constraints. The closer the relaxed problem is to the original problem the more informative the heuristic is. Similarly, the more relaxed the problem is, the easier to compute the heuristic. Thus, it is hard to find a good trade-off between informativeness and relaxation of the heuristic, it often requires a deep knowledge of the problem. (Ghallab, Nau, and Traverso 2004).

Machine learning can help in automated planning in two ways: learning **action models** to feed planners and **search control** to guide planners (Jiménez et al. 2012). The goal of this work is to apply machine learning help planners on the search control task. In other words, we could use machine learning to approximate a good heuristic function for each domain. A similar work was proposed by Yoon et al. in 2006 (Yoon, Fern, and Givan 2006), in which they defined an approximation of a heuristc funcion of the form $H(s, A, g) = \sum_i w_i \times f(s, A, g)$, where s is the state, A is the set of possible actions in that state, and g is the goal. It is basically a linear weighted combination of the features $f(s, A, g)$. In the context of learning a heuristic function, one of the main challenges is how to arrange the features that the learning algorithm will use as input. In the work mentioned above, their main contribution was the encoding of the features $f(s, A, g)$, where they tried to use more information than just the relaxed plan length.

Approach

To address the problem, the proposed solution consists of two main tasks: 1) export data from planner, 2) use the data to estimate a heuristic function. The idea is to run the planner for a set of different problems of the same domain, for each plan found, we export the literals of each state s and the cost for the goal g at that state. The cost is simply the number of steps necessary to reach the goal from a state s.

After exporting data from the planner, it is necessary to prepare the data before we use it to estimate the heuristic function. Here lays the difficult part, the way the data is encoded can lead to very different results. The quality of the encoding can only be measured when testing. The simple and natural form is to capture the literals of each state by their name, eg. (on d1 peg1), and encode them into a numerical form. In addition to encoding, we want the heuristic function to be domain specific, so it is necessary to come up with a feature space representation that uses the literals of different problems of a specific domain and and generalize to other different problems of the same domain.

Once the data is ready, we can then estimate a function using a regression algorithm. In the work of Yoon et. al (Yoon, Fern, and Givan 2006), they used a linear function, which may not capture the behavior of the data in the best way. In this step it would be interesting to try different regression algorithms, perform experiments not only with linear functions, but also with polynomial functions and neural networks. The idea is that this learned function will have a better performance for any problem on the domain it was trained in than a standard function.

Implementation

This section describes the implementation of the project. The implementation was performed using the Python programming language and is divided in two main parts: dataset creation and heuristic learning.

Dataset creation

To create a dataset using the literals of problems' intermediate states, we follow the following steps:

1. Obtain PDDL problems, domains and plans: we used the Planning.domain.api (Dom 2015) to obtain a set of domains, problems, and their respective plans. The API contains 126 domains and 4506 problems. Of all problems, only 3021 have computed plans available. We considered computing the plans for the remaining problems, but there are very complex problems and that turned out to be unfeasible by the time it would require to compute.
2. After obtaining PDDL domains, problems and plans, we extract literals of intermediate states using planning validator VAL (VAL 2014). VAL allows us to retrieve plan length, add and delete lists applied at each state. We run VAL for every plan and parse its ouput. For example, the output of VAL given plan for a simple problem from the Hanoi domain would be:

```
Plan to validate:

Plan size: 1
1:
(move d1 peg1 peg3)


Plan Validation details
-----------------------

Checking next happening (time 1)
Deleting (on d1 peg1)
Deleting (clear peg3)
Adding (clear peg1)
Adding (on d1 peg3)
Plan executed successfully - \
              checking goal
Plan valid
Final value: 1

Successful plans:
Value: 1
```

3. After getting the add and delete lists applied to each state, we use a PDDL parser (PDD 2015) to extract the initial state of each problem. For each state, starting with the initial state, we apply the add and delete lists of the action applied in that state to literals of that state. This way, we achieve consistent values for the literals at each state of the plan.

For the data captured and processed as mentioned above, we then store for each problem a JSON file with a structure as shown in Listing 1.

```
{
  "domain": String,
  "domain_id": int,
  "problem_id": int,
  "plan_length": int,
  "negative_goals": [String],
  "positive_goals": [String],
  "plan_literals": [
     {
       "state": int,
       "action": string
       "literals": [String],
       "add_list": [String],
       "delete_list": [String]
     }
  ]
}
```

Listing 1: Dataset structure

For example, the file for a simple problem from the Hanoi domain would look like the one shown in Listing 2.

It is easy to note that the size of JSON file grows quickly as the length of the plan and number of literals grow. For complex and large problems, this files can grow to a huge size. To avoid this, we only accept plans where the length of the plan times the number of literals in the initial state is less than 10,000. We can think of this threshold as the maximum plan size we accept is a plan of length 100 and with 100 literals in initial state, for example. This threshold, however, can be changed to include more or fewer plans depending on particular needs.

In this sense, the built dataset contains 1885 instances from 101 domains, where minimum number of instances for a domain is 1 and the maximum is 96. The dataset has the size of 214.5MB uncompressed and 6MB compressed. This high compression rate can be explained by the amount of replicated data in each file, as a literal will appear several times in different places, like states, add lists and delete lists.

Heuristic Learning

The heuristic learning consists of using the dataset described previously to approximate a function of the form $H(s, A, g)$ that tells how far the current state is far from the goal state. This function can be a linear or polynomial combination of some feature space $f(s, A, g)$. This feature space is the tricky part of the problem. Ideally, it should consider the information of the literals of each state along with information of delete lists, as considering delete lists would improve the drawback of the relaxed plan length heuristic, which ignores them.

In this work, unfortunately we were not be able to reach our goal completely as we did not learn any heuristic from the dataset created previously. Our tight schedule did not allow to explore this end of the solution, we plan to do so as a future work.

Related work

Several related works (Yoon, Fern, and Givan 2006; Petrik and Zilberstein 2008; Arfaee, Zilles, and Holte 2010; ús Virseda, Borrajo, and Alcázar 2013) have explored the use of machine learning to learn search control to provide planners a better guidance than a standard heuristic. In all of these work, the heuristic is learned from previous solved problems, as proposed in this work, and in all of them they show some improvement compared to the standard heuristics.

Conclusions and Future Work

We present a work towards using machine learning to learn stronger heuristic functions. Unfortunately we were not able to explore the learning part of the solution due to tight schedule. However, we showed a simple yet powerful way to create datasets based on previously solved problems that can be used to learn new heuristic functions. It is important to note that we did not find any publicly available dataset for this purpose.

As a future work we plan to make use of the dataset and experiment with different feature spaces and learning algorithms. Different type of learning algorithms could be used to capture the behavior present in the dataset, like linear and polynomial regressions and even neural networks. We expect to find good solutions and achieve state-of-the-art results.

References

[Arfaee, Zilles, and Holte 2010] Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2010. Bootstrap learning of heuristic functions. In *Third Annual Symposium on Combinatorial Search*.

[Dom 2015] 2015. Api.planning.domains. `http://api.planning.domains/`. Accessed: 2016-11-25.

[Ghallab, Nau, and Traverso 2004] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

[Jiménez et al. 2012] Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(04):433–467.

[PDD 2015] 2015. Pddl parser. `https://github.com/pucrs-automated-planning/pddl-parser`. Accessed: 2016-11-25.

[Petrik and Zilberstein 2008] Petrik, M., and Zilberstein, S. 2008. Learning heuristic functions through approximate linear programming. In *ICAPS*, 248–255.

[ús Virseda, Borrajo, and Alcázar 2013] ús Virseda, J.; Borrajo, D.; and Alcázar, V. 2013. Learning heuristic functions for cost-based planning. *Planning and Learning* 6.

[VAL 2014] 2014. Val. `https://github.com/KCL-Planning/VAL`. Accessed: 2016-11-25.

[Yoon, Fern, and Givan 2006] Yoon, S. W.; Fern, A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In *ICAPS*, 162–171.

```
{
  "domain": "hanoi",
  "domain_id": 124,
  "problem_id": 4269,
  "plan_length": 1,
  "negative_goals": [],
  "positive_goals": [
    "(on d1 peg3)"
  ],
  "inital_state": [
    "(smaller peg1 d1)",
    "(smaller peg2 d1)",
    "(smaller peg3 d1)",
    "(clear peg2)",
    "(clear peg3)",
    "(clear d1)",
    "(on d1 peg1)"
  ],
  "plan_literals": [
    {
      "state": 0,
      "action": "(move d1 peg1
                 peg3)",
      "literals": [
        "(smaller peg1 d1)",
        "(smaller peg2 d1)",
        "(smaller peg3 d1)",
        "(clear peg2)",
        "(clear d1)",
        "(clear peg1)",
        "(on d1 peg3)"
      ],
      "add_list": [
        "(clear peg1)",
        "(on d1 peg3)"
      ],
      "delete_list": [
        "(on d1 peg1)",
        "(clear peg3)"
      ]
    }
  ]
}
```

Listing 2: Data from Hanoi problem