Logic-based Methods for Reasoning About Search in Classical Planning

Salomé Simon
University of Basel
Basel, Switzerland
salome.simon@unibas.ch

Overview

I am currently roughly 9 months into my PhD studies, and the final topic of my PhD thesis is not yet fully settled. My current idea is to focus on *general logic-based methods* for *reasoning about heuristics* and other kinds of knowledge exploited by today's classical planning systems.

Planning research in the last two decades has led to the development of powerful methods for deriving heuristic distance information for classical planning, based on concepts such as delete relaxation (Hoffmann and Nebel 2001), critical paths (Haslum and Geffner 2000), abstraction (Edelkamp 2001), landmarks (Richter, Helmert, and Westphal 2008) and network flows (van den Briel et al. 2007).

A limitation of much of the current work on such distance heuristics is that they are "black boxes" to the planning system that conducts the overall search for a solution: a search algorithm can feed a state to a heuristic estimator and receive a distance estimate from it, but the search algorithm does not have access to information *how* the given estimate was derived.

For example, if a distance estimator returns an infinite heuristic estimate, telling us that a solution does not exist from a given state, it might be useful to know what it is about the current state that leads to this judgment. Perhaps this information can be generalized and usefully applied to other states later encountered during a search (similar to *clause learning* in SAT solving algorithms). Perhaps we can understand the power and limitations of distance heuristics better if they could give us "explanations" of how they come up with our heuristic estimates. Perhaps we could *trust* the implementation of an optimal heuristic search algorithm more if we had a way of independently verifying that the heuristic estimates it returns are really admissible, similar to the way that many SAT solvers can emit resolution proofs showing that input formulas claimed to be unsolvable are indeed unsolvable.

I want to investigate questions of this kind by making planning heuristics and other sources of knowledge about planning tasks, such as landmarks, more transparent by representing them as logical formulas that can be used in generic logical reasoning engines. The hope is that this will lead to a better understanding of current search heuristics, generic methods for combining and generalizing heuristic information, new ways of leveraging existing SAT solving technology within planning, and in the building of bridges between current state-space search planners and SAT-based planners (Kautz and Selman 1992; Rintanen 2010)

In the following I describe two concrete ideas that exemplify this general research direction. The first, "A General Framework for Deriving and Exploiting Information in Classical Planning", describes ongoing and mostly completed work which is based on my MSc thesis, "A General LTL Framework for Describing Control Knowledge in Classical Planning" (Simon 2014). The second, "A Reachability Proof System", describes a research idea that I intend to develop further in the next months.

A General Framework for Deriving and Exploiting Information in Classical Planning

The goal of my Masters Thesis was to develop a unified formalism to derive and exploit information gathered during the search. Currently heuristics derive information in a formalism specific to the heuristic. This makes combining information gathered from different heuristics difficult. With a unified way of describing derived information we could combine it easily and thus create more powerful heuristics. Additionally, we could split up the task of deriving and exploiting information. When developing a new heuristic, we currently need to take care of both deriving information and exploiting it in a correct way. With a unified framework, a researcher that finds a new way of deriving information does not need to additionally show how to exploit it correctly, and similarly if a new way of exploiting information is found, one can just use the already existing techniques for deriving information.

Current State

Our approach for developing this unified formalism utilizes a form of temporal logic called *Linear Temporal Logic (LTL)* (Pnueli 1977). Temporal Logics have already been used extensively in the area of classical planning. Some approaches specify the entire planning task in a temporal logic language and generate plans by theorem proving (Koehler and Treinen 1995) or model construction (Cerrito and Mayer 1998). Other systems encode domain-specific control knowledge in a temporal logic formula, which allows them to prune paths that cannot satisfy the formula (Bacchus and Kabanza 2000;

Doherty and Kvarnström 2001). Another approach generates task-specific LTL formulas in a fully automated fashion from landmark orderings and uses it to enhance the FF heuristic by making the constraint from the formulas visible to the heuristic (Wang, Baier, and McIlraith 2009).

LTL extends propositional logic by temporal operators. The two essential operators are **X** (Next) and **U** (Until). We also use three additional operators **F** (Finally), **G** (Globally) and **R** (Release), which can be expressed by using **X** and **U**, but make formulas easier to read. LTL formulas are interpreted over an (infinite) sequence of worlds, where a world is a truth assignment. This sequence of worlds describes how truth assignments change over time. Checking whether the beginning of a sequence can be extended to satisfy an LTL formula can be done incrementally by applying a *progression* step, which transforms the formula in the following way: The progression $\varphi' = \text{progress}(\varphi, w_0)$ of a formula φ is satisfied by a sequence of worlds $\langle w_1, \dots \rangle$ iff φ is satisfied by the sequence $\langle w_0, w_1, \dots \rangle$.

Our goal is to encode information about (optimal) plans in LTL, and later on exploit the information contained in these formulas. We interpret LTL formulas in the following way: The variables of the LTL formula consist of the STRIPS variables of the planning task, and we also introduce an additional proposition for each action. A world describes a node in the search space, where the variable assignment consists of the STRIPS variable assignment from the state belonging to the node, and assigning true to the action variable denoting the action which reached the node (and false to all others). A sequence of worlds is thus now a path in the search space. We use nodes (instead of states) and action variables in order to be able to incorporate path-dependent information into the LTL formula.

Within this framework, we define two types of useful LTL formulas: *globally feasible* and *locally feasible*. A globally feasible formula must be satisfied by any optimal plan. A locally feasible formula is associated to a node in the search space and must be satisfied by any path from a successor to a goal node (excluding the current node) which results in an optimal plan. We showed that a globally feasible formula is locally feasible for the initial node after progressing it with the initial state and an empty action $a_{\text{no-op}}$. From there on we can thus further create locally feasible formulas for the successor nodes by progressing the formula with the corresponding node. We also showed that the framework allows to add additional information gathered during the search in the form of locally feasible formulas. Furthermore, we defined and proved the correctness of the following rules on how one can merge formulas from different nodes denoting the same state when eliminating duplicates:

- If the g costs of the nodes are different, we can use the formula of the node with the lower g cost.
- If the g costs are equal, we can combine the two formulas by logical conjunction.

As a proof of concept for our framework, we showed on two examples how inferred information can be translated into feasible LTL formulas, and derived a heuristic which bases its goal distance estimate solely on these formulas. The first example creates a single globally feasible LTL formula from landmarks and their orderings (Richter, Helmert, and Westphal 2008), which is progressed through the entire explored search space. In contrast, the second example of unjustified actions (Karpas and Domshlak 2011; 2012) is encoded by deriving locally feasible formulas for each search node. The heuristic we derived basically flattens the node-associated LTL formula in a sense that it extracts all propositions which are currently not true but need to be true at some future point and estimates a lower bound on the cost of any path satisfying the formula by applying optimal cost partitioning to the actions that could make those propositions true.

Future Work

Our current heuristic uses only a small amount of the information contained in the locally feasible LTL formula. Specifically, it ignores all temporal order. We are currently studying different approaches on how to derive more accurate heuristic estimates which also consider temporal orderings in the formula. In the future we would also like to describe other kinds of inferred information as feasible LTL formulas, in order to strengthen heuristics which use those formulas as a base of their estimate.

A Reachability Proof System

Most state-of-the-art heuristics have mechanisms to detect dead ends during search. But when a dead end is detected, the heuristic does not give any reason as to how it came to that conclusion. This means we can neither verify that the state is actually a dead end, nor can we learn the reason why it is a dead end. One goal of my doctorate is to develop a proof system which can, independently from any heuristic, prove or refute that a state is a dead end. This is beneficial both in theoretical and practical aspects:

On the theoretical side, we could compare the ability to detect dead ends of different heuristics with the help of such a proof system by showing that some heuristics only use certain parts of the proof system. The idea is inspired by the resolution proof system in SAT: with its help, it was possible to show that clause learning can provide exponentially shorter proofs than many proper refinements of general resolution like regular and Davis-Putnam resolution (Beame, Kautz, and Sabharwal 2004). Similarly with a proof system for reachability it might be possible to show that different heuristics utilize different information and that some heuristics are strictly more powerful in detecting dead ends than others.

On the practical side we could analyze the reachability proof and extract which properties of the dead end state were actually used for the proof. From this we can conclude that any state which has the same properties is also a dead end. If we now later in the search encounter a state which has the same property we can mark it as a dead end immediately, without needing to calculate the heuristic.

The thoughts and ideas of this section are closely related to the planning as satisfiability approach (Kautz and Selman 1992). In this area of research, a planning problem is defined as a *symbolic transition system (STS)* $\mathcal{S} =$

(Σ, I, G, T), where Σ is a signature, I is the initial formula, G is the goal formula and T is the transition formula over $\Sigma \cup \Sigma'$. A plan of length n exists if the formula $I \wedge T \wedge T' \wedge \cdots \wedge T^{(n-1)} \wedge G^{(n)}$ is satisfiable.

Preliminary Work and Ideas

To prove that a state is a dead end, we need to show that there exists no action sequence (including the empty sequence) which can be applied to the state in question and result in a state satisfying all goal conditions. Since the empty sequence is also included, this also implies that the state is not a goal state. For certain types of dead ends, it is easy to find a proof:

1. A state s with no successors and which does not satisfy the goal condition is a dead end, since *no* action sequence except the empty one is applicable and applying the empty action sequence results in s which does not satisfy all goal conditions.
2. A state s which only has dead ends as successors is a dead end. Since all successors are not goal states, we know that any action sequence $a = \langle a_0, \ldots, a_n \rangle$ reaching a goal from s must have at least length 2. This sequence can be divided into the first applied action a_0, which leads to the successor state s_{succ}, and an action sequence $a^+ = \langle a_1, \ldots, a_n \rangle$ which leads from s_{succ} to the goal state. Since all s_{succ} are dead ends, there is no such sequence a^+ and thus there can also be no such action sequence a.

When only using those two mechanisms, we can detect dead ends recursively by first marking the states that have no successor as dead ends, and then successively marking all states that only have marked states as successors. However, this dead end detection is very limited. It basically simply checks all applicable action sequences, which must all be of finite length, and detects that the last state reached through the sequence is not a goal state. But as soon as we have applicable action sequences of infinite length – which is the case as soon as action sequences with cycles are applicable – the reasoning of the proof lacks its "anchor" (in the form of a state with no successor). For example, in a package delivery problem where the destination of a package is not reachable but at least one truck can move back and forth on the same road indefinitely, we have no state which has no successors.

A more general way of proving that a state is a dead end is to enumerate all reachable states and show that no state fulfills the goal condition an no new state can be reached from any of the states in the enumeration. Assume we have a formula φ describing (at least) all states reachable from the potential dead end s, the transition system of the planning task is T and the goal description G. Proving that the following two formulas are unsatisfiable is sufficient to prove that s is a dead end:

1. $\varphi \wedge T \wedge \neg\varphi'$
2. $\varphi \wedge G$

Formula 2 merely checks if any of the reachable states satisfies the goal condition. If formula 1 is unsatisfiable, this means that by applying an action (which is equal to one step of the transition system), we cannot reach a state which is not already covered in φ. This also means that even with an infinite amount of action applications we will never reach a state that does not satisfy φ, and together with the refutation of formula 1 which shows that φ cannot satisfy the goal condition, we can conclude that we will never reach a state that does satisfy G.

Currently I see two important open questions with this approach:

1. How do we find φ?
2. What can we learn from the refutation proofs?

During my research I found two techniques from the model-checking community which could be helpful in answering these questions: *Property Directed Reachability* and *Interpolants*.

Property Directed Reachability (Suda 2014), originally known as IC3 (Bradley 2011), is a hybrid between explicit and symbolic search which proves the nonexistence of plans of length $0, 1, \ldots$. To this purpose it builds up layers L_k describing an overapproximation of states that reach the goal in k steps. This overapproximation is refined incrementally. If the algorithm reaches a point where a layer L_i is equal to L_{i+1}, it can conclude that no plan of any length can exist.

McMillan (2003) described a technique for model checking which is based on the notion of *interpolants*. Given an unsatisfiable formula $\varphi = \varphi_1 \wedge \varphi_2$, an interpolant P is an implication of φ_1 such that $P \wedge \varphi_2$ is unsatisfiable (it is practically speaking the part of φ_1 responsible for φ not being satisfiable). The algorithm presented in the paper can prove for a given k that a plan of length k exists or (if k is large enough) that no plan of any length can exist. To this end it iteratively evaluates a formula $\Phi = R \wedge T \wedge T' \wedge \cdots \wedge T^{(k-1)} \wedge G^{(k)}$, where R is initially I. If the formula is satisfiable in the first iteration (where $R = I$), a plan of length k has been found. If the formula is unsatisfiable, an interpolant P is calculated, with the following partition of Φ: $\varphi_1 = R \wedge T$, $\varphi_2 = T' \wedge \cdots \wedge T^{(k-1)} \wedge G^{(k)}$. This interpolant P represents an overapproximation of the states reachable from R in one step, and is (in an unprimed version) added to R. Thus, R represents an ever growing overapproximation of the states reachable from I. If for a later iteration, Φ is satisfied, the algorithm aborts (since R is an overapproximation, it cannot be sure that a plan actually exists) and we need to start it with a bigger k. If at some point, the derived interpolant P implies R, the algorithm can conclude that no plan of any length can exist.

Both approaches build some sort of overapproximation of reachable states and can conclude that no plan can exist if at some point this overapproximation does not change anymore. This makes them related to my suggestion, since they basically prove in some form that $\varphi \wedge T$ implies φ', which is only the case if $\varphi \wedge T \wedge \neg\varphi'$ is unsatisfiable. Thus it should be possible to transfer some of the knowledge and ideas of these two techniques into the setting described above. It also shows that the formula describing the reachable states does not need to be exact in the sense that it describes *only* these states, it can also be an (appropriate) overapproximation.

Acknowledgements

I would like to thank Prof. Malte Helmert and Dr. Gabriele Röger for their guidance during my Masters Thesis and during my subsequent work in Prof. Helmert's research group. Furthermore I would like to thank Prof. Fahiem Bacchus for his input and ideas which inspired parts of the subsection "Preliminary Work and Ideas" form the section "A Reachability Proof System".

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1–2):123–191.

Beame, P.; Kautz, H.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research* 22:319–351.

Bradley, A. R. 2011. SAT-based model checking without unrolling. In Jhala, R., and Schmidt, D., eds., *Proceedings of the 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2011)*, volume 6538 of *Lecture Notes in Computer Science*, 70–87. Springer-Verlag.

Cerrito, S., and Mayer, M. C. 1998. Using linear temporal logic to model and solve planning problems. In Giunchiglia, F., ed., *Artificial Intelligence: Methodology, Systems, and Applications (AIMSA 98)*, volume 1480 of *Lecture Notes in Computer Science*, 141–152. Springer-Verlag.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. *AI Magazine* 22(3):95–102.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, 140–149. AAAI Press.

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

Karpas, E., and Domshlak, C. 2011. Living on the edge: Safe search with unsafe heuristics. In *ICAPS 2011 Workshop on Heuristics for Domain-Independent Planning*, 53–58.

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 92–100. AAAI Press.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In Neumann, B., ed., *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992)*, 359–363. John Wiley and Sons.

Koehler, J., and Treinen, R. 1995. Constraint deduction in an interval-based temporal logic. In Fisher, M., and Owens, R., eds., *Executable Modal and Temporal Logics*, volume 897 of *Lecture Notes in Computer Science*, 103–117. Springer-Verlag.

McMillan, K. L. 2003. Interpolation and SAT-based model checking. In Jr., W. A. H., and Somenzi, F., eds., *Proceedings of the 15th International Conference on Computer Aided Verification (CAV 2003)*, volume 2725 of *Lecture Notes in Computer Science*, 1–13. Springer-Verlag.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, 46–57.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982. AAAI Press.

Rintanen, J. 2010. Heuristics for planning with SAT. In Cohen, D., ed., *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming*, 414–428. Springer-Verlag.

Simon, S. 2014. A general LTL framework for describing control knowledge in classical planning. Master's thesis, University of Basel.

Suda, M. 2014. Property directed reachability for automated planning. *Journal of Artificial Intelligence Research* 50:265–319.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In Bessiere, C., ed., *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *Lecture Notes in Computer Science*, 651–665. Springer-Verlag.

Wang, L.; Baier, J.; and McIlraith, S. 2009. Viewing landmarks as temporally extended goals. In *ICAPS 2009 Workshop on Heuristics for Domain-Independent Planning*, 49–56.