Critical Constrained Planning and an Application to Network Penetration Testing

Marcel Steinmetz
Saarland University
Saarbrücken, Germany
{steinmetz}@cs.uni-saarland.de

Abstract

Critical constrained planning is a very interesting, though also very under explored class of classical, as well as probabilistic, planning. In a broad view, constraints allow to limit the space of solutions to solutions that satisfy certain kinds of conditions. In this work, we are going to develop techniques that are in particular suited to solve critical constrained planning instances. Automated, and semi automated network penetration testing has gotten a rise in attention in the previous decade due to the growing size of todays networks: it is not possible anymore to manually check reasonably sized networks for security threats. Planning overall, and in particular critical constrained planning appears to be a very fruitful direction in this area, although many of the previous works lack of either an accurate representation of the problem, or they lack of scalability. In this work, we will continue on Hoffmann's (2015) taxonomy model, and we will identify tractable fragments of, and we will develop efficient methods to solving the variant classes of network penetration testing.

Introduction

Critical constrained planning problems impose additional hard requirements on the solutions of the problem. We will in particular focus on resource constrained planning problems (Nakhost et al. 2012), so problems that contain some sorts of consumable resources (time, money, . . .) that cannot, or only sparsely, be refilled. Although there are many works that consider problems facing resource consumption (e. g., (Koehler 1998), (Haslum and Geffner 2001)), only a very few consider the case of a limited resources that keeps decreasing as the system progresses (e. g., (Nakhost et al. 2012), (Hoffmann et al. 2007), (Gerevini et al. 2008)). Resource constrained planning problems model a bunch of interesting real world applications. For instance, when modeling time as resource, then typically this resource cannot be reset. Other examples are a fixed money budget, or space agents with disconnected (or inactive) power supply. Even though resource constrained planning problems are only a subclass of general resource planning problems, they offer many challenges that make resource constrained planning on its own very interesting. In general, due to the non-increasing structure of the resource, the planner has to plan far ahead in order to not run out of resources before reaching the goal. Nakhost et al. have shown that with growing *resource constrainedness*, current state of the art methods are getting more and more trouble solving these problems (Nakhost et al. 2012). Our work will be focusing around the following observation: a typical phenomenon in resource constrained planning problems is the high density of dead end states – as the search progresses, more and more solutions are ruled out. Thus, dead end detection will be an important key to solve these kinds of problems efficiently.

On the other hand, automated network penetration testing (Arce and McGraw 2004), an apriori completely unrelated area to critical constrained planning, will constitute the second major part of our work. Due to the growing size of today's networks, it is getting harder (and in medium to large company networks even impossible) to identify security threats by hand. Rather, companies are using automated, and semi automated tools to analyse vulnerabilities of their networks. Using planning to simulate attacks to networks is a very promising future direction in (semi-) automated network security testing. A company called CoreSecurity (*http://www.coresecurity.com/*) is already commercially using a planner inside one of their tools to generate possible *attack plans* that are provided to an human security officer to guide the attention to particular, possible security threatening, regions of the network (Lucangeli et al. 2010). One problem of this approach is that it requires a global and exact model of the network, including all the network host configurations. In practice, this is clearly impossible to get and to maintain. Ideally, the model should start with minimal possible knowledge about the network and host configurations. This idea was followed by Sarraute et al. in their design of network penetration testing as solving POMDPs (Sarraute et al. 2011; Sarraute et al. 2012). Unfortunately, solving such POMDP models is only feasible for a very small number of hosts (Sarraute et al. 2012), and thus does not scale to real world networks. In this work we will continue with Hoffmann's work on finding feasible, yet realistic models of penetration testing (Hoffmann 2015). One very typical aspect of probabilistic models of penetration testing is that the probability of reaching a goal state will be low. In other words, the probability of reaching dead ends will be very high. So, dead end detection will play an important role to solve penetration testing problems efficiently, as well.

Learning to Recognize Dead Ends

Current state of the art heuristic search based planning approaches do not really consider the problem of dead end detection specifically. Rather, they make use of the *artifact* that when using a *safe* heuristic function h, whenever $h(s) = \infty$, then s is a dead end. If a state s is identified as dead end, it is not further considered in search (we say that s is *pruned*). Recently, a few works have been published on the topic of detecting planning problem *unsolvability* (e. g., (Bäckström et al. 2013; Hoffmann et al. 2014)). Although these techniques are designed to prove the unsolvability of a planning task, they still can be used in solvable problems: we can use them in order to identify dead ends during the search. This can lead to a significant performance advantage in problems where the number of dead ends abound (Steinmetz and Hoffmann 2016). However, these *unsolvability heuristics* are typically computed in a preprocessing step. In our work we will follow a completely different direction. Instead of computing a data structure detecting dead ends before the search, we make use of the information becoming available during the search in order to constantly refine our dead end identifier, and thus detecting more dead ends.

Observe that the search itself gives proofs of dead ends while exploring the state space (Steinmetz and Hoffmann 2016). Whenever a state s is encountered in an open, closed list search so that each state of the search space that can be reached through s, and s itself are closed, then we know that s must be a dead end. Given this information and additionally $u(s) < \infty$, for an unsolvability heuristic u, we can refine u so that s becomes recognized under u (provided that there is a refinement algorithm for u that supports this). After this refinement, u might *generalize* to other, not yet seen, dead ends as well. For $u = h^C$ (Keyder et al. 2014; Hoffmann and Fickert 2015), this generalization happened in a large scale in the three available resource constrained benchmark domains (NoMystery, Rovers, and TPP) (Steinmetz and Hoffmann 2016). In those experiments, the search space reduction is tremendous. The geometric mean is around two orders of magnitude, and the reduction goes even up to 5 orders of magnitude. However, as the size of the set of atomic conjunctions C is continuously growing, the computation of h^C is getting computationally more complex the more dead ends are being learned. To really benefit from the search space reduction, one has to reduce the number of calls to h^C. To prefilter the calls to h^C, we extract a clause ϕ so that for all t with $t \not\models \phi$ it is $h^C(t) = \infty$, after each time when we have evaluated h^C on a state s and $h^C(s) = \infty$. Inspired by SixthSense (Kolobov et al. 2012), we compute ϕ based on greedily minimizing $\phi = \neg s$ while keeping $h^C(s) = \infty$. This already works pretty well, up to 98% (in the geometric mean) of the dead ends recognized by h^C could be filtered through these conjunctions.

In future work, we have to address open questions in three different parts of the approach: (1) search algorithm, (2) h^C dead end detection, and (3) finding other unsolvability heuristics u that can be used in the framework. (1) Our current experiments have shown that only very few dead ends are learned when running optimal search (A* (Hart et al. 1968)) instead of a depth first exploration of the state space. The intuitive reason is that the farer the distance to the initial state, the higher is the chance of finding dead ends, and thus the more can we learn. In contrast, in A*, the exploration is biased towards the initial state, meaning that fewer dead ends become known, and thus we can learn less. This brings up the question, how can we learn more dead ends while preserving the optimality of the search? One possibly promising direction might be to use IDA*(Korf 1985) instead. What about an anytime search algorithm, such as heuristic depth first search (Bonet and Geffner 2006)?

In (2), we distinguish between (a) the h^C heuristic itself, and (b) learning clauses based on h^C. Because h^C is getting so expensive to compute in the long term (learning many dead ends), there might be room for improvements in the conjunction selection algorithm. Choosing different sets of conjunctions during the refinement has a direct effect on the detection of previously unrecognized dead ends. In our current implementation, we try to greedily minimize the size of each single conjunction that is selected during refinement. Can we choose the set of conjunctions differently so that the number of selected conjunctions becomes smaller than before while recognizing at least as many dead ends? Or does it even make sense to construct larger sets of conjunctions C, e. g.because the $|C|$ by number of detected dead ends ratio becomes smaller? A way towards answering these questions is to identify the *value* of a conjunction, i. e., in how many detected dead ends s does the conjunction play a role in obtaining $h^C(s) = \infty$. But how do we find the value of a conjunction efficiently? Another idea, borrowed from the SAT community (Goldberg and Novikov 2002), do we actually need all conjunctions ever added to the set of atomic conjunctions C? Or can we *forget* some of the conjunctions after a while? Another issue of our current refinement algorithm is that it only allows to refine the set of atomic conjunctions on states s if all of their successor states are already recognized. This in particular makes it hard to use h^C learning additional to other dead end identifiers.

Regarding (2b), recall that the clauses are checked before h^C is evaluated, and h^C is only evaluated if non of the clauses matched. As previously mentioned, actually almost all dead ends are identified through the clauses, $i.e.$, h^C is barely evaluated on dead ends. So, why evaluating h^C at all? It turns out that when just using clauses to identify dead ends, one performs significantly worse than when additionally evaluating h^C. The reason for this is simple: by always skipping the evaluation of h^C, we do not learn enough clauses to cover all the recognized dead ends. Can we select when to evaluate h^C, or when to stop evaluating h^C? Or is it even possible to learn multiple clauses at once that cover all recognized dead ends, and thus getting rid of the evaluation of h^C completely?

(3) Our current results are all based on h^C. However, there might be also other heuristic functions that allow a refinement during search and which might perform equally well, or even better.

Last but not least, can we generalize this approach to a less strict definition of *dead ends*: can we learn to identify states whose only (possible) solution is via one of its ancestors?

Network Penetration Testing

We will be investigating the different classes of Hoffmann's taxonomy models (Hoffmann 2015), and we will design efficient methods to solve them. In short, the planning model simulates an attacker whose goal is to get access to sensitive parts of the network. We will mainly focus on probabilistic models as these allow to easily encode partial knowledge about the network, and even allow to express exploits that are of stochastic nature (e. g., buffer overflow attacks). The optimization criterion that we are looking at is maximizing the probability of reaching the goal as the typical question in network security is how likely it is that an attacker gets access to sensitive areas of the network. Since finding optimal solutions in MDPs (POMDPs) is notoriously hard, we will also consider weaker objectives: finding attack policies that succeed with at least θ probability, or finding policies whose success probability differs from the optimal policy by at most δ (Steinmetz et al. 2016).

First experiments show that solving even these highly restrictive classes of probabilistic planning models is only feasible for small networks (Steinmetz et al. 2016). Due to the large number of dead ends, blindly instantiated heuristic search algorithms perform similar to state space exhaustive methods (VI) – as opposed to other domains where blindly initialized heuristic search algorithms generally perform much stronger than VI.

Unfortunately, we cannot make use of admissible heuristic functions simply because none exist. Thus, finding admissible heuristic functions for goal probability is one of our main research questions. The common approach to find admissible heuristic functions is to identify tractable fragments, i. e., fragments that allow to compute the exact goal probability in polynomial time.

Besides finding heuristic functions, we can improve the efficiency of heuristic search algorithms (as well as VI) through several state space reduction techniques. Such techniques have already been used successfully in classical planning (Pochter et al. 2011; Hall et al. 2013; Wehrle and Helmert 2012). In particular partial order reduction seems to be a very promising direction in network penetration testing: usually the search can select the next host to attack out of a large set of network hosts. However, the order in which they are attacked does not matter with respect to the probability of reaching a goal state – though, the search will still enumerate all possible permutations (an artifact of the apply-once constraint).

Finally, budget constrained network penetration testing is an interesting problem, both, from a practical as well as theoretical view point. It is relevant to network penetration testing in practice because one does not always want to consider *all* attack policies, but rather only those that can be executed in for example a reasonable amount of time, respectively by using a reasonable amount of money. This problem leads us again to the first question: how to learn to recognize dead ends during search? However, now, we are no longer considering search in a deterministic state space, but rather search in a probabilistic state space. So, can we generalize the methods developed for critical constrained (classical) planning to probabilistic problems as well?

References

Arce, I.; and McGraw, G. 2004. Why attacking systems is a good idea. IEEE Computer Society - Security & Privacy Magazine 2(4)

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In Proc. SoCS13.

Bonet, B.; Geffner, H. 2006. Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and Its Application to MDPs. In Proc. ICAPS'06.

Gerevini, Alfonso E.; Alessandro Saetti; and Ivan Serina. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. Artificial Intelligence 172.8 (2008): 899-944.

Goldberg, E.; and Novikov, Y. 2002. BerkMin: a fast and robust SAT- solver. In Design, Automation and Testing in Europe Conference, 142149, March 2002.

Hall, D. L. W.; Cohen, A.; Burkett, D.; Klein, D. 2013. Faster Optimal Planning with Partial-Order Pruning. In Proc. ICAPS'13.

Hart, P. E.; Nilsson N. J.; Raphael B. 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics SSC-4(2), 100-107.

Haslum, P.; and Geffner, H. 2001. Heuristic planning with time and resources. In Proc. ECP01, 121132.

Hoffmann, J.; Kautz, H.; Gomes, C.; and Selman, B. 2007. SAT encodings of state-space reachability problems in numeric domains. In Proc. IJCAI07, 19181923.

Hoffmann, J.; Kissmann, P.; and Torralba, A. 2014. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability, Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14), Prague, Czech Republic, August 2014.

Hoffmann, J.; and Fickert, M. 2015. Explicit Conjunctions w/o Compilation: Computing $h^{\mathrm{FF}}(\Pi^C)$ in Polynomial Time. In Proc. ICAPS'15.

Hoffmann, J. 2015. Simulated Penetration Testing: From "Dijkstra" to "Turing Test++". Invited paper in Proceeding ICAPS'15.

Nakhost, H.; Hoffmann, J.; Müller, M. 2012. Resource-Constrained Planning: A Monte Carlo Random Walk Approach. In Proc. ICAPS'12.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. Journal of Artificial Intelligence Research 50:487533.

Koehler, J. 1998. Planning under resource constraints. In Proc. ECAI98, 489493.

Kolobov, A.; Mausam; and Weld, D. S. 2012. Discovering hidden structure in factored MDPs. Artificial Intelligence 189:1947.

Korf, R. E. 1985. Depth-first Iterative-Deepening: An Optimal Admissible Tree Search. Artificial Intelligence 27, 97–109.

Lucangeli, J.; Sarraute, C.; and Richarte, G. 2010. Attack planning in the real world. In Workshop on Intelligent Security (SecArt 2010).

Pochter, N.; Zohar, A.; Rosenschein, J. S. 2011. Exploiting Problem Symmetries in State-Based Planners. In Proc. AAAI'11.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2011. Penetration testing == POMDP solving? In SecArt11.

Sarraute, C.; Buffet, O.; and Hoffmann, J. 2012. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In AAAI12.

Steinmetz, M.; and Hoffmann, J. 2016. Towards Clause Learning State Space Search: Learning to Recognize Dead Ends. In Proc. AAAI'16.

Steinmetz, M.; and Hoffmann, J.; and Buffet, O. 2016. Revisiting Goal Probability Analysis in Probabilistic Planning. In Proc. ICAPS'16.

Wehrle, M.; Helmert, M. 2012. About Partial Order Reduction in Planning and Computer Aided Verification. In Proc. ICAPS'12.