The TorchLight Tool: Analyzing Search Topology Without Running Any Search

Jörg Hoffmann
INRIA
Nancy, France
joerg.hoffmann@inria.fr

Abstract

The ignoring delete lists relaxation is of paramount importance for both satisficing and optimal planning. In earlier work (Hoffmann 2005), it was observed that the optimal relaxation heuristic h^+ has amazing qualities in many classical planning benchmarks, in particular pertaining to the complete absence of local minima. The proofs of this are hand-made, raising the question whether such proofs can be lead automatically by domain analysis techniques. The TorchLight tool answers this question in the affirmative.

The tool is based on a connection between causal graph structure and h^+ topology. It distinguishes between *global analysis* and *local analysis*. Global analysis shows the absence of local minima once and for all, for the entire state space of a given planning task. Local analysis determines the percentage of individual sample states not on local minima, thus allowing to make finer distinctions. Finally, *diagnosis* summarizes structural reasons for analysis failure, thus indicating domain aspects that may cause local minima.

Complementing the ICAPS'11 and JAIR papers on TorchLight (Hoffmann 2011b; 2011a), we provide a brief summary of TorchLight's workings and results, and illustrate its functionalities with example output on some IPC benchmarks.

Introduction

The ignoring delete lists relaxation is of paramount importance for both satisficing and optimal planning (e.g., Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010; Helmert and Domshlak 2009). The planners based on it approximate, in a variety of ways, the optimal relaxation heuristic h^+ which itself is **NP**-hard to compute. As was observed in earlier work (Hoffmann 2005), h^+ has strong qualities in many classical planning benchmarks. Figure 1 gives an overview of these results (omitting ADL domains and including the more recent benchmarks Elevators and Transport (without action costs).

The results divide domains into classes along two dimensions. We will herein ignore the horizontal dimension, which pertains to dead ends. The vertical dimension divides the domains into three classes, with respect to the behavior of exit distance, defined as $d - 1$ where d is the distance to a state with strictly smaller h^+ value. In the "easiest" bottom class, there exist constant upper bounds on exit distance from both, states on local minima and states on benches (flat regions). In the figure, the bounds are given in square brackets. For example, in Logistics, the bound for local minima is 0 – meaning that no local minima exist at all – and the bound for benches is 1. In the middle class, a bound exists only for local minima; that bound is 0 (no local minima) for all domains shown. In the "hardest" top class, both local minima and benches may take arbitrarily many steps to escape.

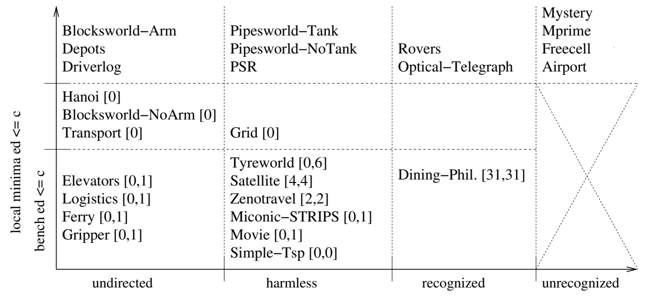

Figure 1: Overview of h^+ topology (Hoffmann 2005).

The proofs underlying Figure 1 are hand-made. For dealing with unseen domains, the question arises whether we can design domain analysis methods leading such proofs automatically. The TorchLight tool answers this question in the affirmative. The key to the analysis is a connection between causal graph structure and h^+ topology. In its most basic form, the connection is this:

If the causal graph is acyclic, and every variable transition is invertible, then there are no local minima under h^+.

The proof of this result works in two steps. Step (A) identifies circumstances under which one can deduce from an optimal relaxed plan for a state s that there exists a monotone exit path, i.e., a path from s to a state s' with $h^+(s') < h^+(s)$ and where all intermediate states s'' on the path have $h^+(s'') = h^+(s)$. Step (B) devises causal graph based sufficient criteria implying that analysis (A) will always succeed. This scheme can be used to prove results stronger than the above, allowing e.g. casual graph cycles arising (only) due to transition "side effects" that are harmless in certain ways.

TorchLight distinguishes between *global analysis* and *local analysis*. Global analysis shows the absence of local minima once and for all, for the entire state space of a given planning task. This is based on step (B) above. Local analysis determines the percentage of individual sample states not on local minima – we refer to this as the *success rate* – thus allowing to make finer distinctions in planning tasks where

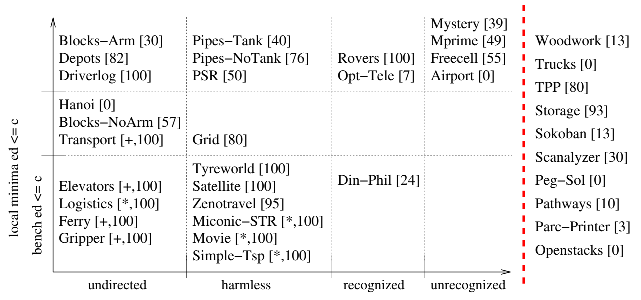

Figure 2: Overview of TorchLight domain analysis results. "*": global analysis always succeeds; "+": local analysis always succeeds if provided an optimal relaxed plan; mean success rates when sampling one state per domain instance.

local minima do exist. To analyze a given sample state s, we feed step (A) with the relaxed plan for s computed by FF's heuristic function. Since this relaxed plan is not necessarily optimal, this local analysis is approximate: if it succeeds, there is no guarantee that s is indeed not a local minimum.

TorchLight is implemented in C based on FF. Its analysis techniques rely on the finite-domain variable representation of planning. This is obtained from the PDDL input by running Fast Downward's translator (Helmert 2009). That translation is the main bottleneck in TorchLight's runtime performance. Up to 100 sample states, in more than 96% of the 1160 test instances in our experiment, the actual analysis takes at most as much time as the translator.

Figure 2 gives an overview of TorchLight's analysis results. The domains whose h^+ topology is not known are shown separately. For each domain, "*" and "+" indicate domain-specific performance guarantees that we have proved. The numbers give the per-domain average success rates when taking a single sample state per instance. Clearly, "harder" domains tend to have lower success rates.[1]

TorchLight's *diagnosis* summarizes structural reasons for analysis failure, thus indicating domain aspects that may cause local minima. Since the tested criteria are sufficient but not necessary, there is no correctness guarantee. Still, at least for local analysis, the diagnosis can be quite accurate. In Zenotravel, it always correctly identifies fuel consumption as the problem. In Mprime and Mystery, most of the time the same correct diagnosis is returned. In Satellite and Rovers, it always reports the problem to be that switching on an instrument, respectively taking an image, deletes calibration – precisely the only reason why local minima exist here. In Blocksworld-Arm and Freecell, the diagnosis identifies critical resources ("hand-empty" and "have-cellspace").

We next exemplify global analysis, local analysis, and diagnosis, with example runs on IPC benchmarks. We close the paper with a brief discussion of future work.

Global Analysis

Figure 3 gives verbatim output of TorchLight when run on the largest Logistics instance from the 1998 competition (we omit some parts of the output that are not relevant here).

The reader familiar with FF will notice FF's footprint in this output. The run of Fast Downward's translator is indicated by TorchLight near the start of Figure 3. Once translation terminates, TorchLight reads Fast Downward's intermediate output file, and matches the values of the finite-domain variables against FF's grounded facts (this involves a few subtle but uninteresting implementation details).

As visible in Figure 3, TorchLight then builds some basic data structures pertaining to the *support graph* (SG), a simple variant of causal graphs, and the domain transition graphs (DTG) as known from Fast Downward (Helmert 2006). It then sets some basic properties of these structures, for example annotating every individual DTG transition with a flag indicating whether or not the transition is invertible.

Once the basic structures are built and analyzed, TorchLight runs global analysis. This works by enumerating all *global dependency graphs (gDG)*. A global dependency graph is a sub-graph of the support graph that, starting from some goal variable x_0, recursively includes all transitive predecessors of x_0. The gDG is called *successful* if it does not contain any cycles, and satisfies a number of supplementary criteria implying that analysis (A), cf. the above, will succeed. If, and only if, all gDGs are successful – i.e., if as shown here the percentage of successful gDGs is 100% – then it is proved that the state space does not contain any local minima under h^+.[2] Further, each gDG delivers a bound on the exit distance. Maximizing this bound over all gDGs delivers a bound that is valid across the whole state space. In the shown Logistics example, that bound is 1. That same bound would be returned for any Logistics instance, i.e., TorchLight here always finds the exact bound as proved by hand (cf. Figure 1).

Note that the shown instance is huge. FF generates almost a million "action templates", i.e., instantiated actions not yet tested for (relaxed) reachability. This instance size is also reflected in the 9.78 seconds runtime for Fast Downward's translator. By contrast, the actual analysis (i.e., the part of it that we're interested in right now) takes only 0.24 seconds.

As shown in Figure 2, global analysis succeeds in Logistics, Miconic-STRIPS, Movie, and Simple-TSP. In all other domains, however, the fraction of successful gDGs never attains 100%. In these cases, nothing is proved, so those gDGs that are successful may at best serve as an indication of which aspects of the domain are "good-natured".

Local Analysis

Local analysis is run on a set of random sample states. The number R of such states is an input parameter to TorchLight. Each state is sampled by executing $K * h^{\mathrm{FF}}(s_I)$ random actions, where K is another input parameter, s_I is the initial state, and $h^{\mathrm{FF}}(s_I)$ is FF's heuristic value for that state. We start in s_I and keep selecting uniformly one of the applicable actions at each state. The path length factor K is set to 5 in our experiments. We have not played much with

[1] In Driverlog and Rovers, deep local minima do exist, but only in awkward situations that don't tend to arise in the IPC instances. Hanoi and Blocksworld-NoArm are not actually easy to solve for FF, and the absence of local minima is due to idiosyncratic reasons.

[2] This is a strictly more general criterion than the one mentioned in the introduction: if the causal graph is acyclic and all transitions are invertible, then all gDGs are successful; but not vice versa.

```
./torchlight -o domains/logistics/domain.pddl -f domains/logistics/p30.pddl

TorchLight: running Fast-Downward translator to generate variables ... done.
TorchLight: creating SG and DTG structures ... done.
TorchLight: static examination of SG and DTG structures ... done.

TorchLight guaranteed global analysis:
No local minima under h+, exit distance bound 1.
Percentage of successful x0/t0 gDGs    : 100.00% (30780 of 30780)

Time spent:    0.14 seconds instantiating 912252 easy, 0 hard action templates
               9.78 seconds in FD translator generating variables
               0.24 seconds in guaranteed global analysis
```

Figure 3: Example run of TorchLight (global analysis) in the Logistics domain.

this parameter; its value makes a difference mainly in domains containing dead ends (like transportation with non-replenishable fuel), which may not be found if the random walks are too short (we get back to this below).

Given a sample state s, and a relaxed plan $P^+(s)$ for s, local analysis applies step (A) to identify whether or not $P^+(s)$ complies with a special case implying the existence of a monotone exit path from s. If so, we say that s is *successful*. If $P^+(s)$ is optimal, then this analysis is sound, i.e., for successful s an exit path as claimed is guaranteed to exist. In TorchLight, $P^+(s)$ is returned by FF's heuristic function, thus $P^+(s)$ is not necessarily optimal, thus the local analysis is approximate.[3] If s has no relaxed plan at all, then we count the state as unsuccessful.

Upon analyzing all sample states, TorchLight outputs the success rate as well as the min/mean/max exit distance bound identified. Figure 4 gives verbatim output for instances from Transport, Blocksworld-Arm, and Mystery.

Figure 4 (a) shows the output for the largest Transport instance of IPC'08. The "-s 100" in the command line gives the number of sample states (called R herein); the default value is $R = 10$. We see that all sample states are successful, indicating (rightly) the absence of local minima. The largest exit distance bound is 2, however most states have a smaller bound, as indicated by the mean 0.16. Exit distance in Transport relates to the number of vehicle moves needed in order to load/unload the next package. That number can easily be constructed to be large, however apparently this does not tend to happen in the present IPC benchmark instances. As before, we see that Fast Downward's translator constitutes by far the most costly part of the computation. Note, though, that the sampling procedure also takes considerable time (spent in the generation of applicable actions).

In Figure 4 (b), we see a domain, Blocksworld-Arm as run in IPC'00, that does contain local minima under h^+, and where, thus, global analysis is necessarily useless – it can only ever answer "sorry no success". By contrast, approximate local analysis returns interesting information, in terms of the success rate: 25% on one of the largest IPC'00 instance as run here (60 blocks). This indicates (rightly) that there are many states on local minima. Note that, for the 25% successful states, the exit distance is constantly 0, i.e., these are situations where h^+ can be decreased directly due to some simple action that is not intrusive anywhere else.

Consider finally Figure 4 (c), which illustrates the role of dead ends. The Mystery domain of IPC'98, encoding transportation with consumption of non-replenishable fuel, is a classical example of a domain containing such states. Figure 4 (c) shows the run of TorchLight on one of the largest IPC'98 instances (these are not ordered strictly by increasing size). Like in Blocksworld-Arm, the success rate is very low, 34% in this case, rightly indicating the complex nature of the search space surface. However, this time that behavior is mostly due to the presence of dead ends among the sample states, and due to the capability of relaxed planning to recognize these. As visible in the output, 57% of the sample states are recognized to be dead ends. Of the remaining 43 sample states (remember that our total is 100), 34 are successful (and 9 are not). If we sample the states less deeply, by setting K in the random path length $K * h^{\mathrm{FF}}(s_I)$ to $K = 1$ instead of $K = 5$, then only 4 sample states have no relaxed plan, and the success rate skyrockets to 78%.

Diagnosis

There is a variety of information sources in TorchLight that could be used for diagnosis, that is, for the identification of domain features that are good-natured/bad-natured. So far, only a first exploration of this has been made, and only in the context of approximate local analysis. We have implemented a few first-shot methods identifying which operators and variables were involved in the reasons for success/failure of such analyses, in the sample states.

Judging from our current results, the most useful one of these methods reports operators that were "harmful" in the analysis, in that they had "side-effects" preventing them from use in the special case identified by step (A). As an example, consider an operator moving a vehicle, whose intended effect is to change the position of the vehicle, but that has a harmful side effect consuming fuel. The diagnosis reports the name of the operator, along with the name of the predicate affected by the harmful effect. It maintains occurrence counts of these operator-predicate pairs, and weighs these pairs by frequency in order to provide some measure of "importance". Figure 5 gives verbatim output for instances from Mprime, Rovers, and Freecell.

[3]TorchLight also implements a version of local analysis guaranteed to be sound. This is based on a localized variant of global dependency graphs. We do not discuss this here since the empirical results are not promising – this local analysis tends to apply only in those domains successfully analyzed by global analysis anyway.

```
./torchlight -o domains/transport/domain.pddl -f domains/transport/p30.pddl -s 100

TorchLight approximate local analysis of sampled states:
Success and hence no local minima under h+: 100.00%
Dead-end states:   0.00%
Exit distance bound min:    0, mean:   0.16, max:    2

Time spent:    0.01 seconds instantiating 39304 easy, 0 hard action templates
              11.79 seconds in FD translator generating variables
               3.37 seconds sampling states
               0.52 seconds in approximate local analysis of sample states
```

(a) Transport

```
./torchlight -o domains/blocksworld/domain.pddl -f domains/blocksworld/p61.pddl -s 100

TorchLight approximate local analysis of sampled states:
Success and hence no local minima under h+:  25.00%
Dead-end states:   0.00%
Exit distance bound min:    0, mean:   0.00, max:    0

Time spent:    0.00 seconds instantiating 7260 easy, 0 hard action templates
               2.33 seconds in FD translator generating variables
               0.83 seconds sampling states
               2.38 seconds in approximate local analysis of sample states
```

(b) Blocksworld-Arm

```
./torchlight -o domains/mystery/domain.pddl -f domains/mystery/p13.pddl -s 100

TorchLight approximate local analysis of sampled states:
Success and hence no local minima under h+:  34.00%
Dead-end states:   57.00%
Exit distance bound min:    0, mean:   0.00, max:    0

Time spent:    0.03 seconds instantiating 43554 easy, 0 hard action templates
               4.83 seconds in FD translator generating variables
               0.27 seconds sampling states
               0.05 seconds in approximate local analysis of sample states
```

(c) Mystery

Figure 4: Example runs of TorchLight (approximate local analysis) in the Transport, Blocksworld-Arm, and Mystery domains.

Consider Figure 5 (a). Like Mystery, Mprime encodes transportation with consumption of non-replenishable fuel. In both domains, the available fuel units are associated with locations, rather than with vehicles (the only difference is that Mprime has an operator allowing to transfer fuel between locations). To discourage planner developers in IPC'98 from analyzing domains and designing domain-specific heuristics, the semantics of both domains was disguised behind meaningless names. One undesirable side effect of this security measure is that the verbatim output in Figure 5 (a) is, also, meaningless. According to Derek Long, more precisely to Long and Fox's (2000) synthesis of generic types, the "feast" operator in Mprime corresponds to a vehicle move, and the "locale" predicate corresponds to the level of available fuel. Thus the analysis in Figure 5 (a) correctly reports the problem to be fuel consumption.[4]

Figure 5 (b) demonstrates a case where the diagnosis identifies a very particular reason for the existence of local minima. Namely, in Rovers, the only reason for their existence is that taking an image has the harmful side-effect of deleting camera calibration. If the same camera is, without a viable alternative, required to take another image, and if re-calibrating the camera involves changing the rover position and thus incurs additional costs, then this side effect may result in a local minimum: the relaxed plan prior to taking the image did not take into account the need to re-calibrate, so after taking the image the relaxed plan length increases. The diagnosis correctly identifies exactly the culprit operator effect. Note, though, that we set $R = 1000$ here. The reason for this is that this kind of awkward situation happens only rarely, so we need a large number of sample states in order to find it. (Even with $R = 1000$, we obtain any diagnosis only in 8 of the 20 IPC'02 Rovers instances.) Note that, with such large R, the runtime advantage of analysis over Fast Downward disappears (in this example at least).

Consider finally Figure 5 (c), in which we demonstrate a case where weighing operator-predicate pairs by frequency is important. Obviously, a major difficulty when playing Freecell is that, when sending a card to a free cell, then the desired effect – making space where the card previously was – is countered by the undesired side-effect of consuming space where the card now is. This is reflected in the

[4]This is not always the case, due to the peculiar encoding of fuel pertaining to locations rather than vehicles. This sometimes tricks the diagnosis into thinking that it's moving away from locations, not fuel consumption, causes the local minima. This never happens in Zenotravel, where fuel pertains to vehicles as one would expect.

```
./torchlight -o domains/mprime/domain.pddl -f domains/mprime/p34.pddl -D

Top weighted non-recovered op/predicate in approximate local analysis:
100.00% of weight -- FEAST (LOCALE)

Time spent:    0.02 seconds instantiating 8964 easy, 0 hard action templates
               1.41 seconds in FD translator generating variables
               0.00 seconds sampling states
               0.00 seconds in approximate local analysis of sample states
```

(a) Mprime

```
./torchlight -o domains/rovers/domain.pddl -f domains/rovers/p19.pddl -D -s 1000

Top weighted non-recovered op/predicate in approximate local analysis:
100.00% of weight -- TAKE_IMAGE (CALIBRATED)

Time spent:    0.02 seconds instantiating 4476 easy, 0 hard action templates
               0.86 seconds in FD translator generating variables
               0.77 seconds sampling states
               0.21 seconds in approximate local analysis of sample states
```

(b) Rovers

```
./torchlight -o domains/freecell/domain.pddl -f domains/freecell/p79.pddl -D

Top weighted non-recovered op/predicate in approximate local analysis:
 58.33% of weight -- SENDTOFREE (CELLSPACE)
 33.33% of weight -- SENDTOFREE (CLEAR)
  8.33% of weight -- SENDTOFREE (ON)

Time spent:    0.05 seconds instantiating 182188 easy, 0 hard action templates
               9.63 seconds in FD translator generating variables
               0.14 seconds sampling states
               0.02 seconds in approximate local analysis of sample states
```

(c) Freecell

Figure 5: Example runs of TorchLight (diagnosis) in the Mprime, Rovers, and Freecell domains. In Mprime, the "feast" operator corresponds to a vehicle move, and the "locale" predicate corresponds to the level of available fuel.

diagnosis by the operator-predicate pair "SENDTOFREE (CELLSPACE)". However, there are many other operator-predicate pairs in the diagnosis that are not that sensible, or not sensible at all. For example, "SENDTOFREE (ON)" suggests that, when sending a card to a free cell, the effect causing trouble is the one removing the card from its previous location. In the example shown, this incorrect diagnosis receives a much smaller weight than the correct one.

Discussion

TorchLight is a new tool whose mission is to analyze search space topology without running any search. What renders this "mission impossible" possible is the observation that causal graphs can be used to characterize rich planning subclasses in which there exist no local minima under h^+.

Apart from furthering our understanding of what makes planning tasks amenable to current heuristic search techniques, such analysis has manifold potential practical uses. In particular, these include: the targeted generation of macro-actions by constructing the identified exit paths; planner performance prediction by machine learning over the generated features; automatic planner/search configuration, even on-line during search since analyzing a single relaxed plan already delivers useful information; automatic problem abstraction by removing (some) harmful effects identified by diagnosis; automatic domain reformulation by using the generated features as reformulation guidance; and PDDL modeling support for end-users by integrating diagnosis as feedback into a modeling environment.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AI* 129:5–33.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS'09*.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AI* 173(5-6):503–535.

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

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *JAIR* 24:685–758.

Hoffmann, J. 2011a. Analyzing search topology without running any search: On the connection between causal graphs and h^+. *JAIR*.

Hoffmann, J. 2011b. Where ignoring delete lists works, part II: Causal graphs. In *ICAPS'11*.

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *AIPS'00*, 196–205.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.