Path-based Heuristics (Preliminary Version)

Nir Lipovetzky
Departament of Technology (DTIC)
Universitat Pompeu Fabra
08018 Barcelona, SPAIN
nir.lipovetzky@upf.edu

Hector Geffner
Departament of Technology (DTIC)
ICREA & Universitat Pompeu Fabra
08018 Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

A path in a classical planning problem is a sequence of causal links a_i, p_i, a_{i+1} that connects a given state with the goal. C3 is a recent based planner that computes paths from the initial state and uses them to recursively decompose the problem into subproblems (Lipovetzky and Geffner 2009). Paths are ranked with an heuristic that takes deletes into account and is built on top of a base heuristic, in this case, the additive heuristic. In this work, we use the resulting path-heuristic to define a state heuristic that we compare with the additive heuristic in the context of a standard greedy best first state planner. Interestingly, while the path-heuristic is an order-of-magnitude more expensive than the additive heuristic, it results in many less expanded nodes, and often appears to be cost-effective and produces better plans.

Background

We consider Strips planning problems P = $\langle F, I, O, G\rangle$ where F is a set of fluents or atoms, $I \subseteq F$ and $G \subseteq F$ are the initial and goal situations, and O is a set of (ground) actions or operators a, each with an Add, Delete, and Precondition list $Add(a)$, $Del(a)$, $Pre(a)$. For convenience, and without loss of generality, we assume as in partial order planning that O contains an End action whose preconditions are the real goals of the problem and whose only effect is a dummy goal g so that $G = \{g\}$.

Most forward-state planners use heuristic functions $h(s)$ for guiding the search in the graph. In particular, the planner HSP uses the additive heuristic $h(s) = h(g)$, where $h(p) = 0$ for atoms p, if $p \in s$, and else is $h(p) = \min_{a \in O(p)}[1 + h(a)]$, where $O(p)$ is the set of actions in O that add p and $h(a) = \sum_{q \in Pre(a)} h(q)$ (Bonet and Geffner 2001). The max heuristic h_{max} is defined in a similar way but with the addition replaced by maximization. The max heuristic is equivalent to the heuristic that can be obtained from a relaxed planning graph (Hoffmann and Nebel 2001) by assigning to each fluent p or action a the index of the lowest layer where it appears, starting from layer 0.

The *best supporters* of a fluent $p \notin s$ in either heuristic, are the actions $a \in O(p)$ with smallest h. The heuristic $h_{\text{FF}}(s)$ used in FF is given by the size $|\pi_{\text{FF}}(s)|$ of the relaxed plan computed by FF in s. This plan $\pi_{\text{FF}}(s)$ can be defined recursively in terms of the best h_{max} supporters, by collecting backwards from the goal, a best supporter a for each goal, and recursively, a best supporter for each precondition of a that is not in s (Keyder and Geffner 2008). The definition in (Hoffmann and Nebel 2001) uses instead a relaxed planning graph and NO-OPs, along with a preference for NO-OPs supporters, which amounts to a preference for best (h_{max}) supports.

The notion of paths, introduced in (Lipovetzky and Geffner 2009), builds on the notion of *causal links* developed in the context of partial order planning. A causal link a, p, b is a triple that states that action a provides the support for precondition p of b. This is taken as a constraint that implies that a must precede b in the plan and no other action that adds or deletes p can appear between them (Tate 1977; McAllester and Rosenblitt 1991).

Paths

The reasons for performing an action usually take the form of a sequence of causal links, that we call *causal chains.* When these causal chains reach the goal, we call them paths.

Definition 1 *A sequence a_0, p_1, a_1, p_2, a_2, ..., p_n, a_n of actions a_i and fluents p_i forms a* causal chain *if p_{i+1} is a precondition of action a_{i+1} and a positive effect (add) of action a_i for $i = 0, \ldots, n$.*

Definition 2 *A path is a causal chain that ends with the End action.*

Paths are not plans but rather *plan skeletons*, where actions may have to be filled in for achieving *all* the preconditions of the actions in the path.

A path starting with an action a that is applicable in the state s is said to be applicable in s. Such a path can be taken to suggest that a may be relevant for achieving the goal in s (Nebel, Dimopoulos, and Koehler 1997). Still, as shown in (Lipovetzky and Geffner 2009), some paths can be shown to be *inconsistent*, in the sense that no plan can make them true. For example, in Tower-n, where blocks $1, \ldots, n$ intially on the table are to be arranged so that i is on top of $i + 1$ for $i = 1, \ldots, n - 1$, the paths

$$t : pick(k), hold(k), stack(k, k + 1), on(k, k + 1), End$$

that start with the applicable actions $pick(k)$, $k = 1, \ldots, n-1$, are all *inconsistent* in the initial state, except for $k = n-1$. This means that every plan that achieves the goal and starts

with the action $pick(k)$ for $k \neq N-1$, must 'break' at least one of the links in the chain. In other words, the reasons for excluding the execution of such actions in the initial state is not only heuristic but structural too. There is an efficient and powerful but incomplete criterion for marking certain paths as inconsistent. In this work, however, we will not use such a criterion in a hierarchical planner but rather build on an heuristic for ranking paths, also developed in (Lipovetzky and Geffner 2009), and evaluate it in the context of a standard forward state planner.

A path-based heuristic

The planner C3 computes paths $a_0, p_1, a_1, \ldots, , p_n, a_n$ with $a_n = End$ from the current state, and uses them to decompose the problem into subproblems: achieving first the preconditions of a_0, then from the resulting state the preconditions of a_1 while preserving p_1, etc. Moreover, this decomposition is recursive, and thus if the preconditions of the first action in the path are true, the action is applied and otherwise, the problem of achieving them is handled in the same manner by finding a path to an open precondition of the first action, and so on.

A key point of the algorithm is the way paths t are ranked and constructed in a given state s. This is done by means of a new heuristic $h(t|s)$, built from a known base heuristic, that *estimates the cost of achieving the goal along the path* t. Since the path t is built incrementally, in order to define this heuristic on paths, we will let t range not only on full paths that reach the goal through the End action, but also over partial paths or chains that do not. The new state heuristic that we want to define and evaluate here can be roughly stated as $h_{path}(s)$:

$$h_{path}(s) = \min_t h(t|s) \tag{1}$$

where t ranges among the paths applicable in s; namely, the causal chains that connect s with the goal. This is the general idea, although later we will restrict the space of paths to be considered. We take the basic building blocks from (Lipovetzky and Geffner 2009).

Ranking Chains and Projected States

Let t be a $a_1, p_2, \ldots, p_n, a_n$ be a causal chain. For generality, we do not assume that the first action is applicable in s nor that the last action is the End action. We want the heuristic $h(t|s)$ to estimate the cost of achieving the goal along this chain. For defining this heuristic, we will estimate the sequence of states $s_1, \ldots, s_n$ induced by the chain so that s_i is the state that is estimated to be true right after the action a_i in the chain is done.

For estimating the sequence of states induced by a chain we use known ideas. We use the expression $\pi(a_i; s_i)$ to denote the relaxed plan that achieves the preconditions of action a_i from the state s_i. This relaxed plan is obtained by collecting the best supporters according to the base heuristic, recursively backwards from the preconditions of a_i (Keyder and Geffner 2008).

If $s_0 = s$ and $\pi_i = \pi(a_i, s_i)$ is the relaxed plan for achieving the preconditions of a_i from s_i, then the state s_{i+1} *projected* right after applying the action a_i is estimated as

$$s_{i+1} = (((s_i \setminus Del(\pi_i)) \cup Add(\pi_i)) \setminus eDel(a_i)) \cup Add(a_i) \tag{2}$$

where $Add(\pi_i)$ is the set of fluents added by the actions in π_i, $eDel(a_i)$ is the set of fluents e-deleted by the action a_i and $Del(\pi_i)$ refers to a *subset* of the fluents deleted by actions in π_i.[1] This subset is defined as the fluents that are deleted not just by one action in π_i that is a best supporter of some fluent p in the relaxed plan, but by *all* the best supporters of p, whether they made it into the relaxed plan or not. The reason is that the choice of best supporters in the relaxed plan is rather arbitrary, and deleting a fluent because an arbitrary best supporter deletes it turns out to be more critical than adding a fluent that an arbitrary supporter adds. So these deletions aim to be cautious.

The heuristic $h(t|s)$ for the chain $t : a_1, p_2, \ldots, p_n, a_n$ in a state s is defined in term of a base heuristic h, that we take to be the additive heuristic, and the sequence $s_1, \ldots, s_n$ of states induced by t according to (2)

$$h(t|s) = \sum_{i=1}^{n} [\, cost(a_i) \,+\, h(Pre(a_i)|s_i) \,] \tag{3}$$

where in the computation of the heuristic $h(Pre(a_i)|s_i)$ for $1 < i \leq n$, all the actions that interfere with the causal link a_{i-1}, p_i, a_i in t are excluded (this meaning that actions that e-delete p_i). This estimate is thus just the sum of the estimated costs of solving each of the subproblems along the path, assuming that the states s_i along the path are those projected and that causal links are respected.

A problem with this estimate, however, is that due to the use of deletes in the projection and the preservation of causal links, it's often infinite. This may reflect that the projected state sequence is inaccurate, or more often, that the decomposition expressed by the path t is not perfect. For example, if a precondition p of an action a_i cannot be established from the state s_i, yielding $h(a_i|s_i) = \infty$, it is possible that such a precondition can be established in the previous subproblem from the state s_{i-1} and maintained into the following subproblem if the action a_{i-1} does not e-delete it.

With this in mind, the estimated cost of achieving the goal through the chain $t : a_1, \ldots, p_n, a_n$ is defined as

$$h(t|s) = \sum_{i=1}^{n} [\, cost(a_i) \,+\, h_i(Pre(a_i)|s_i) \,] \tag{4}$$

where h_1 is equal to the base heuristic h, and h_{i+1} is

$$h_{i+1}(p|s_{i+1}) = \min [\, h(p|s_{i+1}) \,,\, h_i(p|s_i) + \Delta_i(p) \,] \tag{5}$$

where $\Delta_i(p)$ is a penalty term for bringing p from the subproblem i along the path t to subproblem $i+1$. We have set $\Delta_i(p)$ to a a constant, independent of i and p, except when the action a_i e-deletes p where $\Delta_i(p)$ is set to ∞. This is the same heuristic as in (Lipovetzky and Geffner 2009), even if the rational for penalizing bad decompositions is less clear in this setting.

[1] An action *e-deletes* p when p must be false after the action; i.e., when it deletes p explicitly or when it doesn't add p and has a precondition that is mutex with p. (Nguyen and Kambhampati 2001; Vidal and Geffner 2005).

		h_a					h_{path}					
Domain	I	S	D	Ex	Ev	T	S	D	Ex	Ev	T	Quality
Blocks World	50	50	0	1,912	14,834	13.89	48	45	41	402	126.18	65%
Depots	22	11	3	13,369	165,834	184.69	9	3	45	667	79.20	87%
Driver	20	16	0	548	17,679	28.70	14	6	78	995	20.36	90%
Ferry	50	50	6	65	462	0.06	50	43	29	186	0.57	95%
Grid	5	2	0	72	389	3.12	2	1	22	124	72.81	100%
Gripper	50	50	50	98	2,726	0.39	50	2	121	1,692	33.85	100%
Logistics	28	28	1	75	1,256	0.45	28	11	55	1,031	53.15	87%
Miconic	50	50	20	81	725	0.02	50	50	30	289	0.50	77%
Mystery	30	26	9	60	2,141	13.47	22	12	5	84	38.94	94%
Open Stacks	30	20	0	3,916	126,393	184.55	10	0	75	606	201.43	100%
Rovers	40	17	1	4,791	159,923	125.05	13	4	123	3,106	102.78	98%
Satellite	20	20	14	36	5,066	9.42	12	7	25	1,775	268.79	101%
Storage	30	17	3	2,065	31,449	59.24	14	5	67	739	72.12	97%
TPP	30	15	5	8,982	111,056	110.30	10	5	1,910	21,236	168.31	89%
Zeno Travel	20	19	2	58	3,601	185.12	13	6	17	415	33.28	95%
Totals	475	391	114	2,409	42,902	61.23	345	200	176	2,223	84.82	Average
Percentage		82%	24%				73%	42%				92%

Table 1: Additive heuristic vs. Path-based heuristic in a standard greedy best first search on instances from previous IPCs: *I* is number of instances, *S* is number of solved instances, *D* is number of instances where search goes straight to the goal, *Ex* and *Ev* stand for avg. number of nodes expanded and evaluated, *T* is avg time in seconds, and *Quality* is plan quality ratio; e.g. 200% means plans twice as long as those reported by h_a on avg.

Computing $h_{path}(s)$

We have defined the heuristic $h(t|s)$ for causal chains t. We will use now this heuristic to define a path-based heuristic $h_{path}(s)$ over states, defined as $\min_t h(t|s)$, where t ranges over a suitable class of paths applicable in s.

In (Lipovetzky and Geffner 2009), the path t is constructed by computing the measure $\min_t h(t|s)$ greedily. In this work, we use instead the A* algorithm over a graph whose *nodes* are causal chains, whose first action applies in the state s, and whose *goal nodes* represent (full) paths that reach the goal through the End action. For uniformity, we take the *source nodes* in this search to be the chains a_0, p_1, a_1 where a_0 is the dummy $Start$ action, a_1 is an action applicable in s, and p_1 is a dummy atom not deleted by any action in the problem. Last, the children of a node $t = a_0, p_1, \ldots, p_n, a_n$ such that $a_n \neq End$, are the nodes $t' = a_0, p_1, \ldots, p_n, a_n, p_{n+1}, a_{n+1}$ where p_{n+1} is a precondition of action a_{n+1} added by action a_n.

For the A* search, in this graph of causal chains, the evaluation function $f(n) = g(n) + h(n)$ is defined so that for n representing the path $t = a_0, p_1, \ldots, p_n, a_n$, $g(n) = h(t|s)$ and $h(n) = h(End|s_n)$, where $h(t|s)$ is defined as in (4) and $h(End|s_n)$ is the base heuristic with s_n being the last state projected along the chain t.

This search is not guaranteed to yield the exact minimum $\min_t h(t|s)$ over all the paths t because the heuristic $h(n) = h(End|s_n)$ is not always a lower bound, even if $h(n) = 0$ when n represents a path.

Moreover, we consider two pruning techniques for speeding up the search that affect the optimality of the resulting measure as well.

First, in order to restrict the size of the search graph, we prune the nodes n representing a path $t = a_0, p_1, \ldots, p_i, a_i$ if there is another node n' representing a path that ends in the same pair p_i, a_i such that $f(n') < f(n)$.

Second, we prune from the search all causal links a_i, p_{i+1}, a_{i+1} that do not lie along a *minimal path* from s. A minimal path $a_0, p_1, a_1, \ldots, p_i, a_i$ is one where the actions a_i are best (h_{max}) supporters of the fluent p_{i+1}.

The value of the heuristic $h_{path}(s)$ is then set to the value $f(n) = g(n)$ of the first goal node n found in the A* search. This value stands for the heuristic value $h(t|s)$ associated with some minimal path t from s, and is only an approximation of the optimal value $\min_t h(t|s)$ when t ranges over all paths.

Experiments

In order to test the quality and cost-effectiveness of the new heuristic h_{path}, we compared it with the base additive heuristic h_a in the context of a greedy best first state search. The planners were evaluated with a timeout of 1800 seconds and a memory limit of 2GB. The experiments were run on Xeon Woodcrest computers with clock speeds of 2.33 GHz.

The results are shown in Table 1. In terms of coverage, h_{path} solves 9% less problems than h_a, but at the same time, the solutions that it obtains are 8% shorter on average (35% shorter in Blocks). In terms of time, the search with h_{path} is slower than with h_a because of the higher overhead, yet, this is compensated by the additional information gained, that translates in a highly reduced number of expanded nodes. Indeed, in 42% of the problems, the heuristic h_{path} takes the plan straight to the goal without expanding any node off the solution. This is shown in the *D* column. The corresponding number for the h_a heuristic is 24%.

Table 2 shows an experiment aimed at making the search more focused by considering only the actions applicable in a state that are minimal. These are the actions that head the applicable minimal paths. Compared to table 1, the coverage is

		h_a					h_{path}					
Domain	I	S	D	Ex	Ev	T	S	D	Ex	Ev	T	Quality
Blocks World	50	49	0	2,145	13,931	15.22	43	30	87	889	35.18	65%
Depots	22	12	2	5,670	45,741	50.11	16	4	43	535	50.51	88%
Driver	20	18	1	3,693	30,405	152.34	14	6	3,080	23,318	157.50	93%
Ferry	50	50	4	68	284	0.04	50	49	29	100	0.08	97%
Grid	5	2	0	39	142	1.65	4	1	79	347	276.54	117%
Gripper	50	50	50	101	1,426	0.22	50	50	77	807	1.75	76%
Logistics	28	28	2	70	557	0.22	28	1	219	3,318	25.08	86%
Miconic	50	50	12	46	221	0.01	50	48	35	199	0.05	100%
Mystery	30	25	12	7	49	1.87	21	11	8	58	65.42	96%
Open Stacks	30	20	0	1,195	28,894	99.60	17	0	289	1,259	233.29	100%
Rovers	40	34	4	185	3,664	102.63	26	5	64	955	85.38	97%
Satellite	20	20	9	38	617	1.07	19	17	36	697	11.36	100%
Storage	30	16	4	821	3,996	3.47	8	4	2,346	12,875	228.91	111%
TPP	30	16	5	4,137	28,489	68.00	12	4	564	3,483	27.36	84%
Zeno Travel	20	20	5	59	1,211	81.30	18	9	34	656	171.36	94%
Totals	475	410	110	1,203	10,539	38.29	376	239	466	3,300	91.32	Average
Percentage		86%	23%				79%	50%				94%

Table 2: Additive heuristic vs. Path-based heuristic in a standard greedy best first search considering only *minimal actions* applicable in a state: *I* is number of instances, *S* is number of solved instances, *D* is number of instances where search goes straight to the goal, *Ex* and *Ev* stand for avg. number of nodes expanded and evaluated, *T* is avg time in seconds, and *Quality* is plan quality ratio; e.g. 200% means plans twice as long as those reported by h_a on avg.

improved, h_{path} solving 50% of the problems straight to the goal. The minimality requirement increases the number of expanded nodes by h_{path} in Driver and Storage while it decreases in TPP, suggesting how suitable it is such a focused search for certain domains. On the other hand, h_a expands half of the nodes compared to table 1.

There is one additional experiment that we want to consider and include in the final version of the paper if accepted. It is a variation of the first experiment where the minimality requirement on causal chains, in the computation of the path heuristic is dropped. This is likely to result in an additional overhead but may serve in domains like FreeCell or Sokoban where the restriction to minimal paths renders many problems unsolvable. Finally, the experiments presented can be combined with these experiments to be carried out by using the ideas of Fast Downward for using helpful (minimal) actions in the context of a complete search.

Discussion

We have presented a new heuristic for forward state-based planner and preliminary but encouraging empirical results. The heuristic is defined on top of a delete-relaxation heuristic and yet takes deletes into account by means of the notion of paths. The other heuristics that take deletes into account are the admissible h^m heuristics (Haslum and Geffner 2000), used mainly in the context of optimal planning, and the causal graph heuristic (Helmert 2004), that is closely related to the additive heuristic but is defined over multivalued variables and keeps track of side effects pertaining to each variable's parents (Helmert and Geffner 2008). The path heuristic h_{path} introduced in this paper, keeps track of side effects through the states that are projected along paths. From this perspective, it is fruitful to look at the Enforced Hill Climbing (EHC) search procedure in FF, not as a search procedure at all, but as lookahead device for producing a more informed heuristic value for the seed state. This look-ahead is common in chess playing programs where the backed-up value is assumed to be more reliable that the root value. From this point of view, while the EHC lookahead considers paths in the (local) state space that until a state with a better value is found, in the path-based heuristic, the lookahead considers paths in an abstraction, where all but one of the preconditions and positive effects are thrown away, but which are evaluated with all the information available in the original problem. The paths in the state-space considered in EHC have the benefit that the local states are true, reachable states. On the other hand, projected states along the abstract paths considered in the paper, may represent 'unreal' states that can't be reached. However, while EHC looks over real states in the local neighborhood, the abstract paths are forced to reach the goal, and thus, can probe much deeper.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1–2):5–33.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, 70–82.

Helmert, M., and Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proc. 18th Int. Conf. on Automated Planning and Scheduling (ICAPS-2008)*, 140–147.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *Proc. ICAPS-04*, 161–170.

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

Keyder, E., and Geffner, H. 2008. Heuristics for planning with

action costs revisited. In *18th European Conference on Artificial Intelligence (ECAI-08)*.

Lipovetzky, N., and Geffner, H. 2009. Inference and decomposition in planning using causal consistent chains. In *Proc. 19th Int. Conf. on Automated Planning and Scheduling (ICAPS-2009)*.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI-91*, 634–639.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *Proc. ECP*, 338–350.

Nguyen, X. L., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc. IJCAI-01*.

Tate, A. 1977. Generating project networks. In *Proc. IJCAI*, 888–893.

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Proc. CP-05*.