Experience-Biased Symbolic Planning

SoCS 2014 Submission 2

Abstract

Weighted A* search is the foundation for state-of-the-art domain-independent symbolic planners. An important open challenge for these planners is to utilize experience with similar planning problems to speed up subsequent planning episodes. Experience graphs have recently shown promise as a technique for plan reuse in the context of weighted A* searches in robotics, by biasing them towards a subgraph of the search space, typically chosen to consist of the edges used in previous plans. In this paper, we demonstrate how to augment symbolic planners with experience graphs, using the HSP2 planner as a representative example. The resulting planners are able to trade off seamlessly between plan quality and runtime and thus have advantages over past plan-reuse techniques, such as planning by analogy. We demonstrate experimentally that on many symbolic domains, our approach reduces the number of nodes that the HSP2 planner needs to consider by a factor of 2 or more while achieving about the same plan quality.

Introduction

Modern state-of-the-art symbolic planners tend to rely on weighted A* search as their core algorithm. In forward weighted A* search, a frontier of nodes expands from the start until it hits a goal, at which point a solution is found. The performance of weighted A* depends heavily on its *heuristics*, cheaply computable approximations of the remaining distance from a given node to the goal. Heuristics focus the efforts of weighted A* toward the goal and can greatly impact the runtime of the search. Consequently, much research has been devoted to developing new heuristic functions and different techniques for combining multiple heuristic functions together, all with the aim of providing more effective guidance to the weighted A* search.

However, even with good heuristic functions, many symbolic planning problems can be very difficult to solve. A state-of-the-art domain-independent planner, or even a human faced with an unfamiliar kind of problem, might have great difficulty in solving it. After having solved several instances of a problem, humans tend to utilize their experience and learn how to solve similar problems quickly. Similarly, it is beneficial to develop planning algorithms that can learn to solve repeated planning problems faster over time.

In robotics, Experience Graphs (E-Graphs) have recently been introduced as a means of accelerating motion planning from experience (Phillips et al. 2012). Based on previously computed plans (a.k.a. experiences), this methods computes an E-Graph that models the high-level connectivity of the free space in motion planning tasks. The intuition is to remember previously generated paths so that, when a new start and goals are queried, a new path can be quickly generated by reusing subpaths from the E-Graph. A variant of weighted A* is employed which biases its focus toward the E-Graph. While this work demonstrates promising results in robot motion planning, E-Graphs have never before been applied to symbolic domain representations such as the STRIPS language.

Our present contribution is to investigate the use of E-Graphs in STRIPS domains. In particular, we take the HSP2 planner (Bonet and Geffner 2001) as a representative of the class of modern symbolic planners that are based on weighted A*, and augment its heuristic with a bias toward its E-Graph. After presenting the related works, we begin by describing the STRIPS language and the original HSP heuristic. In the following section, we formally introduce E-Graphs in technical detail. Then we combine the two approaches, resulting in the new Experience-Biased HSP planner. We present the algorithm and discuss its theoretical properties. This is followed by experiments showing promise for the application of E-Graphs to symbolic problem solving. Finally, we conclude with some thoughts on extensions worth investigating.

Related Works

Classically, case-based reasoning methods such as PRIAR (Kambhampati and Hendler 1992), SPA (Hanks and Weld 1995), MRL (Koehler 1994), and NoLimit (Veloso and Carbonell 1993) have been used to recall previous plans that appear similar to the current problem, and then adapt them to meet the new specifications. Compared to these methods, our work reuses prior experience in the context of modern weighted A*-based methods for symbolic planning. We also provide a bound on the suboptimality of the solution.

Recently, there has been work on incorporating prior

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

knowledge into modern planners (based on weighted A* and powerful heuristics). For instance, GPG (Gerevini and Serina 2000) uses planning graphs to adapt a prior path to the new scenario by searching locally within windows around plan inconsistencies. To guarantee completeness, the windows can grow if solutions are not found until they constrain the whole plan. In this work (de la Rosa, Olaya, and Borrajo 2007), case-based reasoning is used to choose the order of state evaluations of EHC (enforced hill-climbing), which is a component in modern planners such as FF.

OAKplan (Serina 2010) is a recent case-based reasoning method which converts planning instances into a compact graph representation, and then uses recent kernel methods for matching labeled graphs to select a similar case to the problem at hand. The selected case is then adapted using LPG-ADAPT (Fox et al. 2006). LPG-ADAPT is a replanning method that encourages *plan stability*: it tries to make sure that the new plan is as similar to the previous plan as possible. The method also uses a modern heuristic search approach. While E-Graphs too can encourage stability by reusing old plans, our primary purpose is efficient planning with cost suboptimality bounds, rather than stability.

ERRT-PLAN (Borrajo and Veloso 2012) presents a very different approach to reuse in planning. Similar to E-Graphs, ERRT has its roots in motion planning (Bruce and Veloso 2002). ERRT-PLAN applies this sampling-based motion planner that leverages reuse to symbolic planning. The planner randomly chooses to extend the search tree toward the goal, toward subgoals from previous plans, or to reuse an action from a previous plan. The extension operations are based on modern heuristic search planners. The stochastic nature of the direction the search progresses toward allows it to overcome local minima caused by misleading heuristics.

The primary difference between our approach and previous works is that we plan with prior experience in a way that comes with provable bounds on the suboptimality of the solution. The use of E-Graphs also allows us to formulate this in such a way that the user has control over this suboptimality bound with an intuitive parameter that trades reuse (speed) for solution quality.

The HSP2 (Bonet and Geffner 2001) planner uses the heuristic pioneered by HSP, but replaces greedy hill-climbing with weighted A* search. We test our approach on HSP2 because the latter is straightforward to describe, and because it is representative of the core techniques that are common to many state-of-the-art domain-independent symbolic planners. Some newer additions to the field include the causal graph heuristic of the Fast Downward planner (Helmert 2006) and the landmark heuristic of the LAMA planner (Richter and Westphal 2010). Our approach is orthogonal to these developments and can, in principle, be adapted to work with any search heuristic.

STRIPS Language

A STRIPS problem is a tuple $P = \langle A, O, I, G\rangle$ consisting of an atom set A, operator set O, initial state $I \subseteq A$ and goal condition $G \subseteq A$. Each operator $op \in O$ is defined by its cost, preconditions, add effects and delete effects: $Cost(op) \in \mathbb{R}^+$ and $Prec(op), Add(op), Del(op) \subseteq A$.

The problem P defines a directed state graph (V, c) where V is the power set of A (i.e. states $S \in V$ correspond to collections of atoms), and the edge costs are

$$c(S, S') = \min\{Cost(op) \mid S \supseteq Prec(op) \text{ and } S' = (S \setminus Del(op)) \cup Add(op)\}$$

By default, $c(S, S') = \infty$ when there is no operator directly transitioning from S to S'. Given a STRIPS problem P, we seek a low-cost path from the initial state $I \in V$ to any of the goal states $S \in V$ such that $S \supseteq G$. In this paper, we are particularly concerned with sequences of problems in which A and O are fixed, but various pairs $\langle I, G\rangle$ are queried.

HSP Heuristic

Many of today's state-of-the-art domain-independent STRIPS planners are based on the HSP2 planner (Bonet and Geffner 2001), which we now describe. It's common to compute heuristics by solving an easier, relaxed version of the original problem. In STRIPS, one might ignore the delete lists of operations. Since having more atoms makes preconditions and the goal condition more likely to hold, this can only make the problem easier, so the heuristic derived from relaxation never overestimates the true cost.

However, even the relaxed STRIPS planning problem includes set-cover as a special case, making it NP-hard. Intuitively, we see that the relaxed search space remains exponential in size. To reduce it, HSP2 decouples the atoms, instead estimating the cost of achieving each individual atom.

Let's say we want a heuristic estimate of the distance from state S to a goal that contains G. HSP2 estimates the cost to achieve an atom $a \in A$ from S by $g_S(a) =$

$$\begin{cases} 0 & \text{if } a \in S \\ \min_{op \mid a \in Add(op)} \left(g_S(Prec(op)) + Cost(op)\right) & \text{if } a \notin S \end{cases}$$

The HSP heuristic (also used in HSP2) is then defined by

$$h^{HSP}(S) = g_S(G),$$

the estimated cost of achieving all atoms in G. Note that $g_S(\cdot)$ is evaluated on certain atom sets, namely $Prec(op)$ and G. To keep the computations feasible, define $g_S(P) = \max_{p \in P} g_S(p)$ for atom sets P. The resulting HSP-max heuristic is an underestimate; indeed, it satisfies a stronger property called consistency.

Definition 1. *For $\epsilon \geq 1$, a heuristic h is called **ϵ-consistent** iff $h(S) = 0$ when S is a goal and $h(S) \leq \epsilon c(S, S') + h(S')$ for all $S, S' \in V$. h is **consistent** iff it is 1-consistent.*

Theorem 1. *Given an ϵ-consistent heuristic, A* without re-expansions is guaranteed to find a path costing no more than ϵ times the optimal path cost (Likhachev, Gordon, and Thrun 2003).*

If the atoms were completely independent, a more accurate estimate would be $g_S(P) = \sum_{p \in P} g_S(p)$. Despite its inconsistency in general, the latter HSP-plus heuristic is often useful in practice, as it uses more state information: it

decreases if progress is made towards *any* goal atom, instead of just the most distant goal.

From a computational perspective, the HSP heuristic must compute $g_S(a)$ for all $a \in A$ whenever a new state S is generated. This can be done by dynamic programming, iterating the recursive formula to a fixpoint in Bellman-Ford-like fashion. Pseudocode is listed in Algorithm 1.

Algorithm 1 ComputeG(S)

```
for all a ∈ A do
  if a ∈ S then
    g_S(a) ← 0
  else
    g_S(a) ← ∞
  end if
end for
repeat
  for all op ∈ O do
    cost_estimate ← g_S(Prec(op)) + Cost(op)
    for all a ∈ Add(op) do
      g_S(a) ← min (g_S(a), cost_estimate)
    end for
  end for
until no further change in g_S-values
```

Experience Graphs

E-Graphs can improve total search time over a series of related queries on the same graph. Formally, an E-Graph is a subgraph of the state space. It is updated between queries, typically by adding the solution path from the previous search query.

Suppose we have a consistent heuristic $h(S, S')$ for the distance between arbitrary pairs of states. That is, h obeys the triangle inequality, is never dominated by an edge cost, and $h(S, S) = 0$ for all $S \in V$.

Planning with E-Graphs is simply a matter of running weighted A* search with the **E-graph heuristic** h^E, derived from h. h^E biases the search to follow E-Graph edges instead of h by penalizing the latter by an inflation factor $\epsilon^E > 1$. To be precise, define

$$h^E(S_0) = \min_{N, S_N, \pi} \sum_{i=1}^{N} \min\left(\epsilon^E h(S_{i-1}, S_i), c^E(S_{i-1}, S_i)\right)$$

with the minimization taking place over all goal states S_N and all paths $\pi = \langle S_0, S_1, ..., S_N \rangle$ of arbitrary length.

c^E encodes E-Graph edge costs, or ∞ if the corresponding edge is not a part of the E-Graph.

In the limit as $\epsilon^E \to 1$, E-Graph edges offer no advantage because $h(S_{i-1}, S_i) \leq c^E(S_{i-1}, S_i)$. Hence, using the triangle inequality to coalesce the sum,

$$h^E(S_0) = \min_{N, S_N, \pi} \sum_{i=1}^{N} h(S_{i-1}, S_i) = \min_{S_N} h(S_0, S_N)$$

Conversely, as $\epsilon^E \to \infty$, the inflated terms dominate. Hence, the minimizing path becomes one which incurs the least estimated cost outside the E-Graph. In particular, h^E has the following property:

For sufficiently large ϵ^E, once the A frontier touches an E-Graph component which includes a goal state, no node outside the E-Graph will ever again need to be expanded.*

This means, for instance, that if the E-Graph already contains a path from the start to a goal, then A*, guided by h^E with large bias parameter ϵ^E, will never leave the E-Graph. If the E-Graph is small, a solution would promptly be discovered. In this manner, little work is needed to solve previously encountered subproblems.

We cannot afford to compute h^E according to its literal definition, as there are far too many paths to consider. Fortunately, there exists a practical means of computing it. By the triangle inequality, any consecutive pair of $h(S_{i-1}, S_i)$ terms can be merged into one. Thus, we lose no generality in restricting S_i to lie on the E-Graph for $0 < i < N$. The heuristic computation can focus on the E-Graph, using $h(S_{i-1}, S_i)$ to estimate the cost of leaving the E-Graph at S_{i-1} before reentering it at S_i.

Let V^E be the union of the goal states and the E-Graph's vertices. In addition to following E-Graph edges, we imagine it's possible to "jump" from any $S \in V^E$ to any $S' \in V^E$ at cost $\epsilon^E h(S, S')$. That is, let

$$c'(S, S') = \min\left(c^E(S, S'),\ \epsilon^E h(S, S')\right)$$

In a preprocessing stage before the main search, we apply Dijkstra's algorithm once in reverse with the costs c' to compute the estimated distance-to-goal $h^E(S)$ from every $S \in V^E$. Ignoring the computation of h (presently considered as a blackbox), this preprocessing takes $O(|V^E|^2)$ time, which is insignificant for small E-Graphs and goal sets.

Later, when A* generates $S \notin V^E$, we see that we have already precomputed costs of paths consisting of all but the first edge of the minimizing path π in the definition of $h^E(S)$. Thus, we derive the mathematically equivalent but much cheaper computation

$$h^E(S) = \min_{S' \in V^E} \left(\epsilon^E h(S, S') + h^E(S')\right)$$

Theorem 2. *If the base heuristic h is consistent, then h^E is ϵ^E-consistent (Phillips et al. 2012).*

Figure 1 illustrates the idea. Suppose the dark edges have cost 3 and make up the E-Graph. Off-E-Graph edges are not shown, as they are only referenced indirectly via h; instead, light edges represent the jump estimates $h(S_i, S_j) = 2$ (with one exception: $h(S, G) = 5$), which we inflate by a factor $\epsilon^E = 2$ to make 4 (or 10, respectively). Dijkstra's algorithm follows dark and light edges backward to compute the h^E values shown alongside each node.

G is on the E-Graph, though this need not be the case in general. V^E is the pentagon on the right, and its h^E values are precomputed before the search. For $S \notin V^E$, $h^E(S)$ is computed only when A* generates S.

Experience-Biased HSP

We now combine the ideas from HSP2 and E-Graphs to construct a domain-independent STRIPS planner which learns

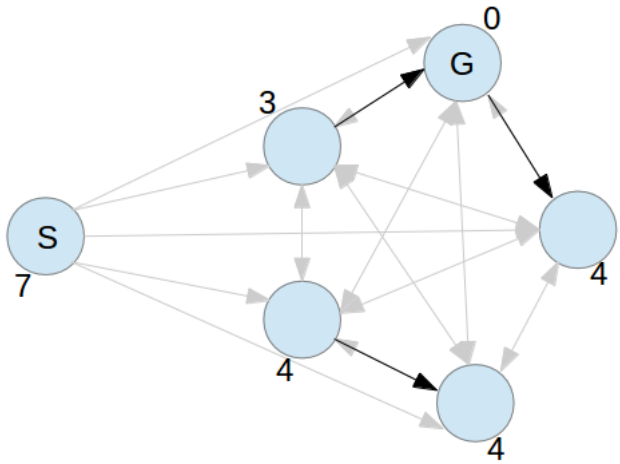

Figure 1: An example E-Graph with three edges, and corresponding h^E-values. S is not on the E-graph.

from experience. In STRIPS, a consistent heuristic $h(S, S')$ is given by the HSP-max estimate $g_S(S')$. We can compute $g_S(S')$ for all $S, S' \in V^E$ in precisely $|V^E|$ runs of Algorithm 1: each run provides us with $|V^E|$ of these values, yielding $|V^E|^2$ values in total.

As before, Dijkstra's algorithm inflates and combines these estimates with the E-Graph edges to derive $h^E(S)$ for all $S \in V^E$. Note that in STRIPS, V^E need not include every goal state: the minimal goal G suffices as it's always the easiest to reach in the heuristic relaxation.

Upon encountering a new state $S \notin V^E$, we run Algorithm 1 again to compute $g_S(a)$ for all atoms $a \in A$. Then it's a simple matter to compute the E-Graph heuristic by

$$h^E(S) = \min_{S' \in V^E} \left(\epsilon^E g_S(S') + h^E(S')\right)$$

We remark that, when E-Graphs are involved, the values $g_S(S')$ on the right-hand side range over a variety of different states S'. Compare this to the formula for h^{HSP}, which uses only $g_S(G)$. HSP2 would still compute all the g_S-values despite using them only once, so here we get to consult the E-Graph essentially for free. The only additional work is the minimization over $S' \in V^E$, which takes O($|V^E||A|$) time. A pseudocode implementation is listed in Algorithm 2. The parameter ϵ^E controls bias toward the E-Graph, while the weight ϵ provides a uniform goal-directed bias.

ComputeG(S) implements the dynamic programming computation of g_S-values. Each iteration of the outermost loop permanently fixes at least one $g_S(a)$ value, so it does at most $|A|+1$ iterations. Multiplying nested loop iterations together and dropping cardinality signs for brevity, a conservative runtime bound on Algorithm 1 is $O(\mathbf{A^2O})$. Therefore, it costs $O(\mathbf{A^2O} + \mathbf{V^EA})$ time to generate each state.

The preprocessing cost is $O(\mathbf{V^EA^2O} + (\mathbf{V^E})^\mathbf{2}\mathbf{A})$, asymptotically equivalent to preemptively generating every state in V^E. In fact, most of the preprocessing requires no knowledge of G, so it can be done in advance of the new

Algorithm 2 Search()

for all $S \in V^E$ **do**
  ComputeG(S)
  for all $S' \in V^E$ **do**
    $c'(S, S') \leftarrow \min\left(c^E(S, S'),\ \epsilon^E g_S(S')\right)$
  end for
end for
Compute h^E on V^E by reverse Dijkstra from G with c'.
Run A* on the full graph from I with heuristic ϵh^E:
for $S \notin V^E$ when generated by A* **do**
  ComputeG(S)
  $h^E(S) \leftarrow \min_{S' \in V^E}\left(\epsilon^E g_S(S') + h^E(S')\right)$
end for
if A* successfully found a path to some goal state $S \supseteq G$ **then**
  Add all edges of the solution path to the E-Graph.
  return solution path
else
  return failure
end if

planning query. Once the query $\langle I, G\rangle$ is given, the remaining work consists of completing the $c'(S, G)$ computations from $g_S(a), a \in G$, and then running Dijkstra's algorithm from G. This costs only $O(\mathbf{V^E}(\mathbf{A} + \mathbf{V^E}))$ time.

For comparison, standard HSP2 without E-Graphs takes $O(\mathbf{A^2O})$ time (with essentially the same constant factors) to generate each state, and does not incur the cost of generating a state unless the A* frontier encounters it. When HSP2 is augmented with small E-Graphs, the same asymptotic bound continues to hold, provided that the number of E-Graph vertices is limited to $O(\mathbf{AO})$. Furthermore, we retain completeness and the ability to configure suboptimality bounds by adjusting the bias parameters:

Theorem 3. *If there exists a path from initial state I to a goal state $S \supseteq G$, the experience-biased HSP-max search algorithm finds a path which is at most $\epsilon\epsilon^E$-suboptimal.*

This follows directly from Theorems 1 and 2, since the weighted heuristic ϵh^E is $\epsilon\epsilon^E$-consistent. However, E-Graphs make no guarantee of improving the search time. Indeed, biasing toward bad regions could trap the planner inside large local optima. Potential gains depend entirely on choosing a good small set of edges to form our E-Graph. In our experiments, we simply keep the previous solution path; different choices remain a topic for future investigations.

Experiments

We augmented the original HSP2 planner code (Bonet and Geffner 2001) with E-Graphs. Since HSP-plus performs better in practice than HSP-max, we take the former heuristic as our baseline. The purpose is to measure how E-Graphs affect the number of states generated by the planner in various STRIPS domains. We prepared two experiments, each presenting scenarios in which we hope for the planner to learn from experience. The STRIPS problems are taken from the HSP2 test suite; we note that every operator in these problems has cost 1.

Experiment 1: Reusing Fragmented Experiences

In this experiment, we solve several hundred problem instances spanning 18 symbolic domains. For each problem, we first plan without experiences, using the HSP-plus heuristic in an A* search with weight $\epsilon = 5$. This search acts as our scientific **control**. If the control fails to find a solution within 5 seconds (on an Intel Core i7-2600), we discard the problem instance. Otherwise, we record the plan cost and the number of states that were generated during the search. Thus, we end up recording data for 321 problem instances.

For each of these, we run 3 additional searches using E-Graphs, setting $\epsilon^E = 5$ but reducing ϵ to 1, thus maintaining the same theoretical suboptimality bound as planning with no E-Graph weighted by $\epsilon = 5$. (Note that although $\epsilon = 1$, some inflation is implicit in the use of HSP-plus. HSP-plus may overestimate by a factor equal to the number of atoms in the goal in the worst case, but only if the atoms are extremely highly coupled.)

The E-Graph consists of 20%, 50% or 80% of the edges selected at random from the control's solution path, over the 3 respective searches. For each of these, we again record the solution cost and number of nodes generated, and compare these figures against the control. We define **speedup** as the ratio of the number of nodes generated in the control to the number generated in the E-Graph search. Likewise, we define the **cost improvement** as the ratio of solution cost in the control to cost in the E-Graph search.

Results are summarized in Table 1. To estimate the typical range of speedups, we present the 25th and 75th percentile speedups among the instances of each domain. The full data from Experiment 1 is plotted in Figures 4 to 6. We see that, while performance is virtually unaffected in some domains, it's never substantially worsened and, in certain domains, the experience helps greatly. For example, results are highly encouraging in the Blocks domain. Furthermore, Figure 2 shows that the solution quality remains competitive; indeed, we see that when E-Graphs are employed, the path cost is slightly reduced on average.

Note that using 100%, or perhaps even 80%, of the E-Graph would seem to trivially grant us the solution, as we could simply follow the remembered path depth-first. In practice, since we fixed the bias parameter to $\epsilon^E = 5$ rather than raising it to an extremely large value, the search expands a fair number of states even if we remember most or all of the path. This is necessary to ensure the suboptimality bound, as the planner does not know whether the E-Graph can be "trusted" to yield an efficient path. E-graphs are used as suggestions, not as a trusted form of knowledge; the planner remains responsible for proving suboptimality.

Experiment 2: Reusing Nearby Experiences

The previous experiment tested our ability to complete partial plans that are fragmented by gaps; however, the experimental searches solved identically the same problems as the control used to generate the E-Graph. Now, we try changing the start and goal states by moving each of them a small distance within the state space. In this manner, we test how well E-Graphs enable the planner to generalize to nearby but non-identical queries.

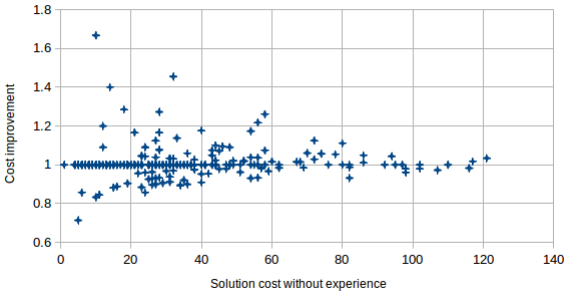

Figure 2: Relative plan costs using 50% of the control experience, displaced 0 steps from the control query.

Domain	Count	20% exp	50% exp	80% exp
blocks	35	1.08-1.82	1.37-3.73	1.90-5.84
driverlog	14	1.00-1.49	1.11-1.58	1.15-1.60
elevators	30	1.00-1.62	1.00-2.84	1.04-3.19
freecell	34	1.00-1.06	1.00-1.12	1.00-1.08
grid	2	1.07-1.18	2.16-3.57	2.22-3.75
logistics00	28	0.91-1.04	0.98-1.13	1.03-1.19
logistics98	5	0.95-1.01	1.09-1.62	1.24-3.00
mprime	19	1.00-1.00	1.00-1.07	1.00-2.46
pegsolitaire	29	1.00-3.57	1.13-4.49	1.68-6.63
pipesworld-notankage	15	1.00-3.27	1.00-7.07	1.00-9.95
pipesworld-tankage	8	1.00-1.00	1.00-1.02	1.00-1.45
rovers	13	0.97-1.23	1.23-7.09	1.23-7.17
satellite	16	0.99-1.00	1.00-1.26	1.00-1.26
scananalyzer	16	1.00-1.15	0.93-1.20	1.00-1.36
sokoban	11	1.12-2.10	1.32-3.02	1.56-7.80
tpp	10	1.00-1.00	1.00-1.23	1.00-2.31
transport	23	1.00-1.14	1.09-1.86	1.19-1.89
zenotravel	13	1.02-1.15	1.08-1.44	1.14-1.58
TOTAL	321	1.00-1.29	1.00-1.91	1.02-2.68

Table 1: Experiment 1 speedup statistics, summarized in the form (1st quartile)-(3rd quartile).

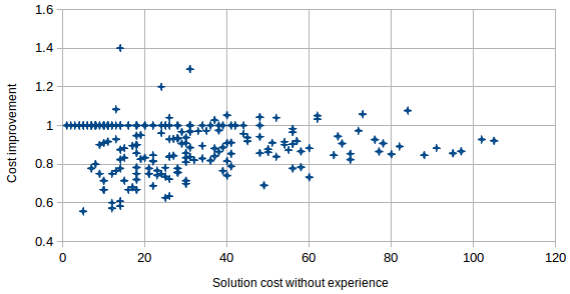

Figure 3: Relative plan costs using 100% of the control experience, displaced 20 steps from the control query.

Domain	Inst	5 steps	20 steps	50 steps
blocks	35	2.12-7.92	2.12-8.70	2.21-6.25
driverlog	14	0.86-2.42	0.85-1.30	0.82-1.20
elevators	30	1.09-4.04	0.99-2.76	0.90-2.06
grid	2	2.21-3.71	1.11-1.34	1.10-1.30
logistics00	28	1.00-1.15	0.90-1.11	0.84-1.18
logistics98	5	1.01-2.95	0.82-2.06	0.80-2.41
pipesworld-notankage	15	1.09-5.30	1.00-5.46	1.00-2.88
pipesworld-tankage	8	1.00-1.20	0.99-1.00	1.00-1.13
rovers	13	1.00-1.50	1.00-1.00	1.00-1.00
satellite	16	0.91-1.25	0.87-1.15	0.90-1.43
scananalyzer	16	0.74-1.52	1.00-1.00	1.00-1.00
transport	23	1.26-1.91	0.93-1.87	0.98-1.46
zenotravel	13	0.80-1.27	0.84-1.02	1.00-1.00
TOTAL	218	0.99-2.72	1.00-2.07	1.00-1.88

Table 2: Experiment 2 speedup statistics, summarized in the form (1st quartile)-(3rd quartile).

We narrow our focus to 13 domains, and take from each of them the same set of instances as in Experiment 1, yielding 218 problem instances in all. This time, we save 100% of the solution path found by the problem's corresponding control from Experiment 1. Then, for each problem, we form 3 modified problems where the start and goal are each displaced by 5, 20 or 50 steps, respectively, of random walk along the graph. Note that, due to the unpredictable nature of random walks, the shortest-paths distances of the moved nodes from their initial positions is only bounded from above by the number of steps taken, and may vary widely.

Now, on each of the displaced problems, we run 2 searches: the first uses no E-Graph and acts as our new control, while the second uses the E-Graph created using the undisplaced original problem, consisting of the path found by the Experiment 1 control. Speedup and cost improvement ratios are taken with respect to Experiment 2 controls; this is to provide an apples-to-apples comparison, as the displaced problem is different from the original.

Results are summarized in Table 2, with the full distributions plotted in Figures 7 to 9. Again we observe encouraging speedups in some domains. As one would expect, the advantage provided by the E-Graph decreases as we walk farther distances from the original query. Figure 3 is a representative plot to show that, once again, path quality is typically not made much worse by using E-Graphs. Nonetheless, this time we do observe a slight cost degradation on average.

Discussion

Figures 4 to 9 display the individual speedups achieved over all of our experimental trials. We observe that cases in which E-Graphs mislead the planner into generating more states (i.e. speedup < 1) are quite rare, so it seems experiences can only help. We set the bias parameters arbitrarily; tuning them by domain or problem size may well yield even better results. Interestingly, we observe a clear positive correlation between problem size or difficulty, as measured by the number of nodes generated in the control, and the speedup factor. This suggests E-graphs may be especially helpful on the hardest problems.

On the other hand, there is some overhead to E-Graph usage. Although we explained in the theoretical analysis that the overhead is not too large, constant factors do matter in practice, particularly when the speedups are still in the single-digits. We measured the number of nodes generated instead of the time taken to complete each search, so as to produce deterministic results which are independent of most engineering details. Our experiments serve as a proof-of-concept for our approach.

However, some additional work remains before state-of-the-art performance can be achieved. Aside from engineering optimizations and integration with newer state-of-the-art heuristics, future work will need to consider more intelligent ways to generalize from the experiences encoded by E-Graphs. This presents a unique challenge in symbolic planning, as distinct states, even if separated by a large distance in the state transition graph, may still be conceptually related. Thus, a planner could potentially benefit by learning to transfer experiences between parts of the state space which are somehow related. With such an approach, it would be essential to limit overgeneralization, so as not to waste too much time on false leads.

Finally, we remark that the Experience-Biased HSP algorithm is highly parallelizable, even if A* is retricted to expanding nodes sequentially. Recall that the sum, min and max functions are computable in logarithmic time using linearly many processors. Therefore, with minor adjustments, ComputeG(S) has parallel depth $O(\mathbf{A} \log \mathbf{A} \log \mathbf{O})$. With $\mathbf{A} \max(\mathbf{O}, \mathbf{V^E})$ processors, the total cost of generating a state is $O((\mathbf{A} \log \mathbf{O} + \log \mathbf{V^E}) \log \mathbf{A})$. Using even more processors, the cost of *expanding* a state can be reduced to the same by generating its successors in parallel. If still more processors are available, some may be dedicated to the task of preemptively computing h^E for states that are likely to be generated in the future, e.g. near the E-graph or A* frontier.

Conclusions

We saw that, by reusing pieces from past plans, E-Graph-based planners can find solutions more quickly as they learn and acquire experiences. Many interesting directions remain for future investigation. It remains unclear how the bias parameters should be set. In addition, more advanced schemes will be needed to expand and prune E-Graphs if they grow

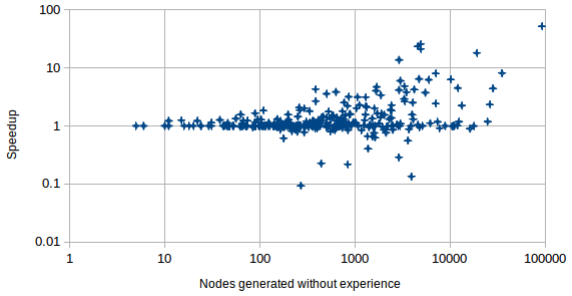

Figure 4: Speedups when planning with 20% of the control experience, displaced 0 steps from the control query.

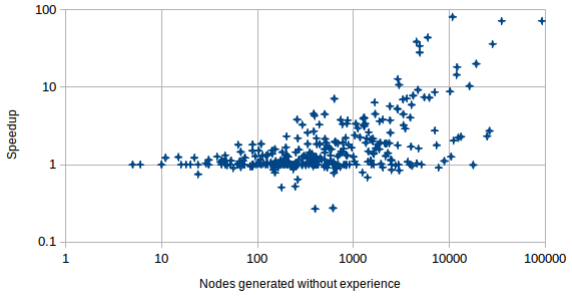

Figure 5: Speedups when planning with 50% of the control experience, displaced 0 steps from the control query.

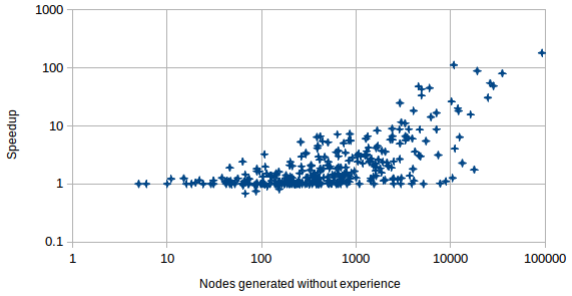

Figure 6: Speedups when planning with 80% of the control experience, displaced 0 steps from the control query.

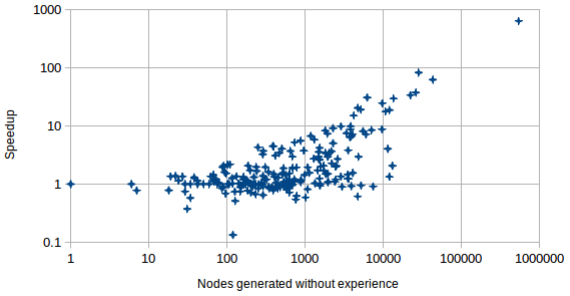

Figure 7: Speedups when planning with 100% of the control experience, displaced 5 steps from the control query.

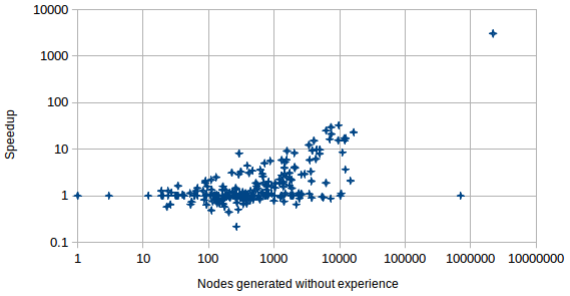

Figure 8: Speedups when planning with 100% of the control experience, displaced 20 steps from the control query.

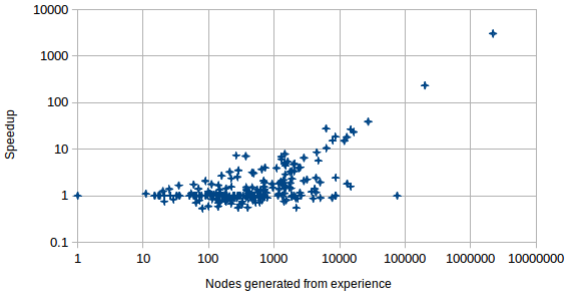

Figure 9: Speedups when planning with 100% of the control experience, displaced 50 steps from the control query.

large from collecting a lot of solutions. Keeping track of the most recently computed plans seems like a good idea intuitively, but other possibilities may be worth trying.

As mentioned in the discussion, generalization and parallelization certainly merit further investigation. Generalization is a particularly important problem in AI learning theory, and there is plenty of literature on the topic. For example, (Fikes, Hart, and Nilsson 1972) generalize plans on the original STRIPS problem solver. The problem of generalization seems related to that of abstraction, or automatically extracting structural information from a symbolic domain or problem description. Works such as (Helmert, Haslum, and Hoffmann 2007) project symbolic state graphs onto smaller spaces; one could potentially apply E-graphs on the projected space to achieve the desired generalization.

E-Graphs should be thought of not so much as a specific algorithm, but as a tool which is orthogonal to a lot of the major planning techniques out there. In principle, one could apply E-Graphs to any of the state-of-the-art A* heuristic-based planners, and one might even adapt the method to obtain anytime, incremental search using experiences as in (Phillips et al. 2013), but in symbolic domains.

Finally, E-Graphs might find other uses, such as in replanning. If an unexpected event invalidates a robot's plan mid-execution, the replanner can recall the original solution as an E-Graph to encourage reparations along the old path. In addition to speeding up replanning, the E-Graph could result in a less disruptive solution, in the sense that it avoids unnecessarily deviating from the original plan. Minimizing disruption is important when resources must be allocated in advance according to the plan, or when cognitively-burdened humans are involved with its execution.

References

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

Borrajo, D., and Veloso, M. 2012. Probabilistically reusing plans in deterministic planning. In *Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain-Independent Planning*, 17–25.

Bruce, J., and Veloso, M. 2002. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

de la Rosa, T.; Olaya, A. G.; and Borrajo, D. 2007. Using cases utility for heuristic planning improvement. In *ICCBR*, 137–148.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial intelligence* 3:251–288.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *In Proc. ICAPS*, 212–221. AAAI Press.

Gerevini, A., and Serina, I. 2000. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 112–121.

Hanks, S., and Weld, D. 1995. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research* 2:319–360.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)* 26:191–246.

Kambhampati, S., and Hendler, J. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55:193–258.

Koehler, J. 1994. Flexible plan reuse in a formal framework. In Bäckström, C., and Sandewall, E., eds., *Current Trends in AI Planning*. IOS Press. 171–184.

Likhachev, M.; Gordon, G.; and Thrun, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press.

Phillips, M.; Cohen, B. J.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience gfvraphs. In *Robotics: Science and Systems*.

Phillips, M.; Dornbush, A.; Chitta, S.; and Likhachev, M. 2013. Anytime incremental planning with e-graphs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2444–2451. IEEE.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39(1):127–177.

Serina, I. 2010. Kernel functions for case-based planning. *Artif. Intell.* 174(16-17):1369–1406.

Veloso, M. M., and Carbonell, J. G. 1993. Derivational analogy in Prodigy: Automating case acquisition, storage, and utilization. *Machine Learning* 10:249–278.