Pattern Selection Strategies for Pattern Databases in Probabilistic Planning

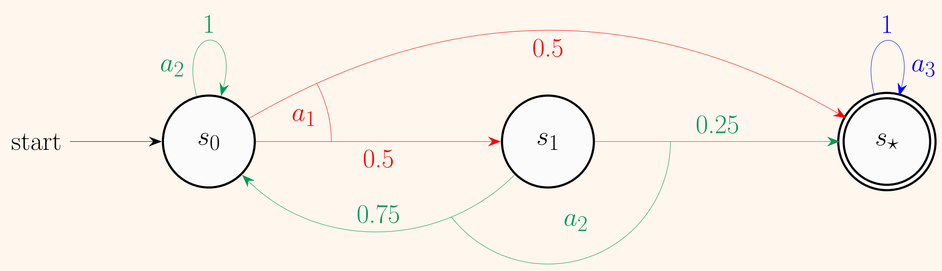
Thorsten Klößner, Álvaro Torralba, Marcel Steinmetz, Jörg Hoffmann
{kloessner, steinmetz, hoffmann}@cs.uni-saarland.de, alto@cs.aau.dk

Probabilistic Planning

Markov Decision Processes As the baseline planning model, we consider goal-oriented Markov Decision Processes (MDPs) with infinite horizon, consisting of

- A finite set of *states* S
- A finite set of *actions* A
- A *transition probability function* $T : S \times A \times S \to [0, 1]$
- An *initial state* $s_0 \in S$
- A set of *goal states* $S_\star \subseteq S$

Furthermore, for all $s \in S, a \in A$, either $T(s, a, \cdot)$ is a probability distribution or $T(s, a, \cdot) \equiv 0$



Objectives We consider to optimization objectives: *Goal-probability maximization* ("MaxProb") and *Stochastic Shortest Path Problems* (SSPs, Bertsekas (1995)).

In these settings, behaviour is specified by a policy $\pi : S \to A$. For MaxProb, policy π is optimal for state s if the probability to eventually reach the goal when starting in s and executing π is maximal among all policies. The SSP setting also assumes a *cost function* $c : A \to \mathbb{R}^+$. π is optimal for a state s if starting in s and executing π the goal state is reached with certainty and among all such policies, π has the lowest expected cost-to-goal.

Heuristic Search An optimal policy can be found by SSP heuristic search algorithms. These require a heuristic that *overestimates* the optimal goal probability or *underestimates* the optimal cost-to-goal of all states in the respective settings.

One family of such heuristics that has recently surfaced are *Pattern Database (PDB) Heuristics* for probabilistic planning (Klößner and Hoffmann (2021), Klößner et al. (2021)).

Pattern Database Heuristics

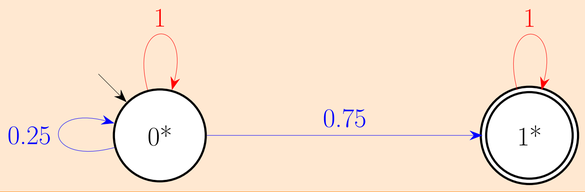
Using pattern databases requires a factored representation of the state space using *state variables* $\mathcal{V}$ with finite domains $D(v), v \in \mathcal{V}$. The states of a factored state space are variable assignments s with $s(v) \in D(v)$. Furthermore, each action a maintains a *precondition* $\mathrm{pre}(a)$ which dictates unique values that must be set for some variables in order for the action to be applicable. Each action a has multiple possible *effects* $\mathrm{Eff}(a)$, each a partial variable assignment that determines the state change by assigning values the state variables accordingly. The *factored goal* is a partial variable assignment that specifies a goal value for specific variables.

PDBs are based on *projections*, which consider only a subset of the state variables, called a *pattern*.

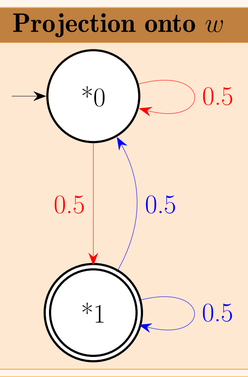
State Variables

Here, v and w can take on the values 0 and 1.

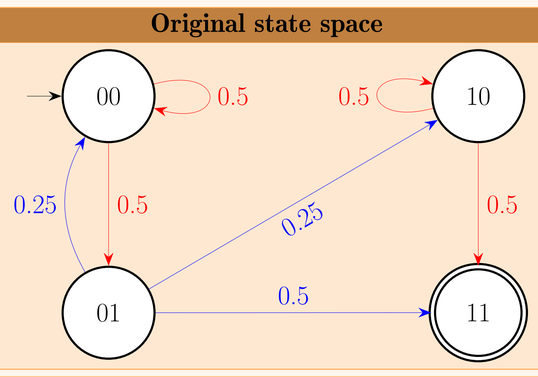
Projection onto v



Projection onto w



Original state space



To construct a PDB heuristic, a collection of patterns is needed. For each pattern of the collection, a pattern database is built, consisting of a lookup table containing the abstract state values of its projection. When a state value estimate is queried, the state value estimates for each projection are looked up and combined.

We focus on the effective construction of the underlying pattern collection.

References

Bertsekas, D. 1995. Dynamic Programming and Optimal Control, (2 Volumes). Athena Scientific.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In Howe, A.; and Holte, R. C., eds., Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07), 1007–1012. Vancouver, BC, Canada: AAAI Press.

Klößner, T.; and Hoffmann, J. 2021. Pattern Databases for Stochastic Shortest Path Problems. In Proceedings of the 14th Annual Symposium on Combinatorial Search (SOCS'21), 131–135. AAAI Press.

Klößner, T.; Torralba, Á.; Steinmetz, M.; and Hoffmann, J. 2021. Pattern Databases for Goal-Probability Maximization in Probabilistic Planning. In Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21), 80–89. AAAI Press.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In Rossi, F., ed., Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13). AAAI Press/IJCAI.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning. In Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19), 362–367. AAAI Press.

Trevizan, F. W.; Thiébaux, S.; and Haslum, P. 2017. Occupation Measure Heuristics for Probabilistic Planning. In Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17), 306–315. AAAI Press.

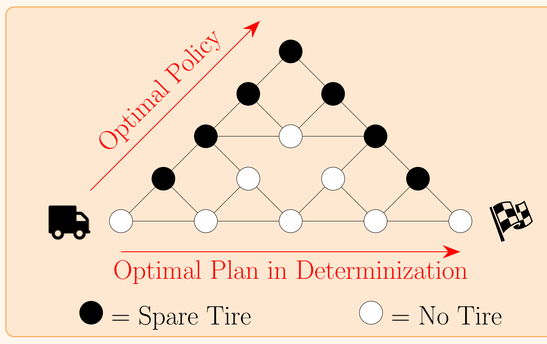
Pattern Construction by CEGAR

CEGAR In classical planning, Counter-Example Guided Abstraction Refinement (CEGAR) can be used to generate pattern collections (Rovner, Sievers, and Helmert, 2019). CEGAR in the context of PDB pattern construction is an iterative technique which starts with some pattern P. In each iteration, CEGAR computes an abstract plan for the projection of P, executes the abstract plan in the concrete state space and if the abstract plan fails, adds a new variable v which provoked a *flaw* to the pattern. A flaw occurs when the precondition of an action was not satisfied for v or the required goal value of v was not reached. CEGAR repeats until the abstract plan constitutes a concrete plan or a resource limit is reached.

CEGAR can be used to generate pattern collections by starting with a collection of all single variable patterns and iteratively using CEGAR on one of the patterns. In the probabilistic setting, we can apply it to the determinization of the task. However, this comes with drawbacks.

The Problem with Determinization-based CEGAR

The problems of determinization-based CEGAR become visible in the domain *triangle-tireworld*. Here, a truck must arrive at a goal by driving through a triangular roadmap. When driving, the truck may lose a tire with some probability. Luckily, spare trees are scattered at some locations which can be used to change the broken tire. To maximize the goal probability, the agent should use the outermost path on which spare tires are always present. In the determinization however, the agent may simply choose to never lose a tire, so the spare tires can be completely ignored. Determinization-based CEGAR therefore terminates immediately after starting with the pattern containing the truck variable. This pattern is very uninformative!



Policy-based CEGAR The issues in the example above appear as an artifact of determinization. We fix theses issues and recover the property that CEGAR eventually finds a solution by computing abstract *policies* during CEGAR. To find potential flaws in an abstract policy, we explore the full policy graph. We consider three exploration strategies: A simple Breadth-first search (BFS), a Monte-Carlo strategy that samples successors according to their probability (MCS) and a strategy sorting exploration candidates by likelihood of their generating path (MLP).

function FindFlawsMCS(π, s_0)
  return FindFlawsMCRec($\pi, s_0, \emptyset$)

function FindFlawsMCSRec($\pi, s, closed$)
  if $s \in closed \cup S_\star$ **then**
    return $\emptyset$
  $flaws \leftarrow$ GetFlaws(s)
  if $flaws \neq \emptyset$ **then**
    return $flaws$

  $closed = closed \cup \{s\}$
  $successors = \{t.\ T(s, \pi(s), t) > 0\}$
  while $successors \neq \emptyset$ **do**
    sample $t \leftarrow successors$ according to T
    $successors = successors \setminus \{t\}$
    $flaws \leftarrow$
      FindFlawsMCRec($\pi, t, closed$)
    if $flaws \neq \emptyset$ **then**
      return $flaws$

function FindFlawsMLP(π, s_0)
  // priority queue, sorted by
  // *ascending* priority
  open $\leftarrow \{\langle s_0, 1\rangle\}$
  closed $\leftarrow \emptyset$

  while $open \neq \emptyset$ **do**
    choose $\langle s, p\rangle \leftarrow open$ with highest p
    $open \leftarrow open \setminus \{\langle s, p\rangle\}$
    if $s \in closed \cup S_\star$ **then**
      continue
    $closed \leftarrow closed \cup \{s\}$
    $flaws \leftarrow$ GetFlaws(s)
    if $flaws \neq \emptyset$ **then**
      return $flaws$

    for all t s.t. $T(s, \pi(s), t) > 0$ **do**
      $p_{new} = p \cdot T(s, \pi(s), t)$
      InsertOrUpdate($open, t, p_{new}$)

  return $\emptyset$

Pattern Construction by Hillclimbing

For SSPs only, we also consider pattern generation by local search, which is a traditional technique used in classical planning to construct pattern collections (Haslum et al., 2007)

The idea is to treat the space of possible pattern collections as a search space. The successor collections of a collection C are those collections obtained from C by growing any pattern P in the collection by a single variable $v \notin P$, unless the resulting pattern is already present.

Hillclimbing starts with the single variable patterns. The algorithm iteratively applies the following steps:

1. Sample some states from the state space
2. For all successor collections, compute the induced PDB heuristic estimates
3. Successor with most improved sample states estimates is chosen

The algorithm only depends on the PDB heuristic that a collection induces, so instead of using hillclimbing on the determinization, we can use the canonical PDB heuristic for SSPs (Klößner and Hoffmann, 2021) during hillclimbing to evaluate the pattern collections!

Experimental Results

In our experimental evaluation, we compare our pattern construction techniques against the corresponding determinization-based approaches as well as systematic pattern generation (Pommerening, Röger, and Helmert, 2013). For SSPs we also compete against h^{roc} (Trevizan, Thiébaux, and Haslum, 2017). We draw the following conlusions:

- Systematic pattern generation is significantly beaten by both of our pattern construction algorithms
- Policy-based CEGAR is vastly superior to determination-based CEGAR in problems where the latter terminates early with a plan, predominantly in instances of triangle-tireworld
- When this is not the case, little to no benefit over determinzation-based CEGAR
- Hillclimbing in the space of SSPs performs best, expanding significantly less states than the determinization-based variant in some domains