Advances in Artificial Intelligence and Machine Learning; Research 3 (1) 778-815 Received 23-01-2023; Accepted 25-02-2023; Published 04-03-2023

Goal Agnostic Learning and Planning without Reward Functions

Christopher Robinson ckevinr@gmail.com
Louisville, KY
USA

Joshua Lancaster
Louisville, KY
USA

Corresponding Author: Christopher Robinson

Abstract

In this paper we present an algorithm, the Goal Agnostic Planner (GAP), which combines elements of Reinforcement Learning (RL) and Markov Decision Processes (MDPs) into an elegant, effective system for learning to solve sequential problems. The GAP algorithm does not require the design of either an explicit world model or a reward function to drive policy determination, and is capable of operating on both MDP and RL domain problems. The construction of the GAP lends itself to several analytic guarantees such as policy optimality, exponential goal achievement rates, reciprocal learning rates, measurable robustness to error, and explicit convergence conditions for abstracted states. Empirical results confirm these predictions, demonstrate effectiveness over a wide range of domains, and show that the GAP algorithm performance is an order of magnitude faster than standard reinforcement learning and produces plans of equal quality to MDPs, without requiring design of reward functions.

Keywords: Graphical models, Markov model, Probabilistic methods, Hypergraphs, Sequential planning

1. INTRODUCTION

This paper presents an algorithm created specifically to solving arbitrary planning problems without requiring a reward function or pre-defined transition model. The impetus for such agents is based on two principles: (1) the idea that crafting reward functions introduces a risk of bias; and (2) objective based learning models reduce the potential for knowledge re-use.

Many extant methods have seen broad use and are highly effective at solving a range of sequential problems in an equally wide range of domains. However, one key indicator of a common problem is the very existence of this diversity. For every parent methodology, there are dozens, if not hundreds of application–specific variants, designed to solve specific domains and addressing problems which emerge when those domains magnify limitations of the general framework. Given that the most frequently used problem solving and learning systems all have many adaptations to cover this wide

Citation: Christopher Robinson and Joshua Lancaster. Goal Agnostic Learning and Planning without Reward Functions. Advances in Artificial Intelligence and Machine Learning 2023;3(1):50.

range of circumstances, we can see that similar problems of adaptation apply- systems which find wide adoption are highly flexible and thus are able to fill a wide range of roles.

In general, when convergence to a goal policy is desired, there is a necessary component of design to align progressive algorithms towards solution convergence. In planning, this requirement can generally be summed up as either the design of a reward function (as with reinforcement learning or MDPs), or a detailed model, as with automated planning and classical artificial intelligence algorithms. It is possible to view these two systematic functions as serving the same role- to provide a model which represents the problem in soluble form- maximizing reward or satisfying modeled transition conditions.

From these observations, we can hypothesize that a problem solving system which includes both planning and learning components, integrates features from highly effective extant agent designs, and eschews both world modeling and reward functions would fit the bill for an efficient, effective system. One of the key components, then, will be the elimination of the common design based limiting factors, while integrating components in a way which does not re-introduce problems ameliorated by other factors. In doing so, we can address some ubiquitous outstanding issues through a holistic approach to the design of the agent, rather than adjustment of the agent to fit a problem domain.

The Goal Agnostic Planning (GAP) algorithm applies these principles by combining an MDP-like planner with an RL-based learning mechanism, integrated with a composite datastructure combining a hypergraph, pointer arrays, and linked lists. This datastructure is populated and updated throughout learning so that Dijkstra's algorithm may be used to find an optimal maximum probability path between an observed current state and any reachable goal state which exists. GAP agents therefore require no modification to re-use already learned domain knowledge when presented with an alternate goal state. They do not require manual construction of a transition graph or reward function.

In addition to achieving the objective of producing learning agents with the previously described properties, we are also able to analytically prove several valuable features of the algorithm, including exponentially bounded rates of goal convergence; learning rates proportional to the reciprocal of epoch number; polynomial computational complexity in both space and time; and explicit conditions and performance impacts for agents learning under state abstractions or uncertainty.

To demonstrate the effectiveness of GAP agents in practice, we explore 3 example domains to demonstrating the accuracy of the analytic predictions. On these problem domains, we evaluate the performance of the GAP agent against performance baseline set by an MDP planner, and a learning baseline set by Q-Learning.

1.1 Related Work

In this section, we will be discussing well-established methods for solving sequential problems, in particular to discuss the concepts we appropriate for our work, and present the context for the limitations of prior systems we seek to ameliorate with the GAP.

Our objective here is to demonstrate, through prior literature, that methods in Reinforcement Learning (RL), Markov Decision Processes (MDPs), and automated planning share common design–related threads that limit effectiveness, which are intimately related to world design and reward function selection. While there are immediately apparent constraints associated with designer bias and breadth of representation implicit in either building a world model or constructing a reward function, we also intend to illustrate deeper connections which cannot be directly addressed by minimizing these topical design problems on a case–by–case basis.

We seek to illustrate three primary limitations in extant approaches with this review:

1. That the need for an explicitly designed reward function or world model is intricately tied to the effectiveness of these algorithms;

2. That construction of such guiding functions recasts a problem into a form which is oriented towards narrow goal sets, losing information; and

3. That the many variants of these systems indicate these limitations, via necessitating bespoke adaptations to ameliorate them.
   In highlighting these issues, we are developing the rationale for development of a system which diverges from the prior designs.

Planning for sequential problems is a very well-studied topic, and naturally invites the question as to why another approach is necessary. In this section, we discuss well-established methods for solving sequential problems, and highlight limits and restrictions of these methods addressed by the GAP algorithm. Our objective is to demonstrate that extant methods have *design–related* limitations that are intimately related to world design and reward function selection.

RL agents operate in a state/action framework, learning a quality function for maximizing prospective rewards. A common through–line for reinforcement–based systems is the necessity of reward function design for convergence. Performance of an RL agent is predicated on the *quality of this function*, a relationship explored in detail by [1]. An additional limitation expressed both in [2], and [3], is goal orientation. Reward functions are constructed in relation to a *specific objective*, so training applies only to that goal. We remove both these limits entirely by separating learning from both rewards and specific goals.

While considered more flexible than most machine learning systems, MDPs are reliant on careful modeling of the system in question. [4], discusses the construction of action sets for MDP formulation as a design methodology, illustrating the presence of implicit optimality conditions. [5], investigates the use of reinforcement learning to supplant reward functions, showing that reward design effects the success of the planner. [6], discusses problems associated with identifying probabilities for goal achievement and reaching dead ends in the MaxProb problem. We are able to explicitly derive these probabilities for the GAP agent. A special case of Markov Decision Processes is the Stochastic Shortest Path (SSP) problem, most notably investigated in [7]. SSP problems seek to identify an optimal policy for stochastically varying costs, similar to MaxProb. [8], discusses

outstanding issues with policy definition related to the implementation of SSP policy determination. We find a polynomial-time, globally optimal solution in this special case of the SSP.

Efforts towards integrating learning and planning domains to improve performance have seen success as well. [9], presents Graphplan, which operates on a task graph, extended to probabilistic planning in [10]. However, they acknowledge limitations of overlapping action results– an issue the GAP algorithm resolves. [11], presents a probability-based belief model in their Abstraction Augmentation- a notion we extend and adapt to state abstraction as a transform. [12], combines reinforcement learning with search-based planning, implementing their DARLING algorithm. However, they still implement reward-based training and focus on semantic planning, losing the computational benefits of graph-based systems. We advance these concepts by unifying their properties and reducing detriments by eliminating design-dependence.

Use of model simplification and abstraction to reduce state space size and improve planning have also been investigated. [13], evaluates state abstractions as applied to tree search, similar to our state-mixing interpretation of abstractions but lacking the analytic power of our model. [14], evaluate model reductions for automated planning. They note a goal-state mapping condition analogous to our convergence condition for abstracted domains, and use connected component analysis to identify the presence of dead ends- a method we *simplify* through our trap net analysis. In [15], the authors develop a system to learn abstractions in a probabilistic planning domain. Their agents are designed for symbolic planning rather than graphical planning, however, and are constructed in the standard reward–based framework for MDPs, lacking the design–agnostic elements our design implements.

While success has been seen with these methods, we can see that there are still inherent limits imposed by world construction and design of reward functions. We address these problems in tandem by modeling the planning task as a lower order combinatorial problem operating on a 3–dimensional datastructure, confining the space complexity to $O(n^3)$, and the time complexity to $O(n^2)$ using Dijkstra's Algorithm. Our hypergraph data structure allows for planning of any task within a domain and learns a representative model by observation without human design influence. It combines aspects of prior work approaching these outstanding problems in a coherent single system: removing the reward function and implementing non–search planning using the maximally probable path as the action policy.

1.2 Contribution

We address these problems by modeling the planning task as a lower order combinatorial problem operating on a 3–dimensional datastructure, confining the space complexity to $O(n^3)$, and the time complexity to $O(n^2)$, rather than semantic logic. This addresses issue (1) by treating the learning task as a probability optimizing planning path, and issue (2) by retaining all the functional operations within polynomial time space. Our hypergraph data structure learns a representative model by observation without human design influence. It combines aspects of prior work on these outstanding problems: removing the reward function, implementing non–search planning, and losing no state-to-state observation data.

We are able to derive several benefits: GAP agents are goal agnostic, require no design of either reward functions or world models, operate in polynomial time, and are natively explainable. Solutions

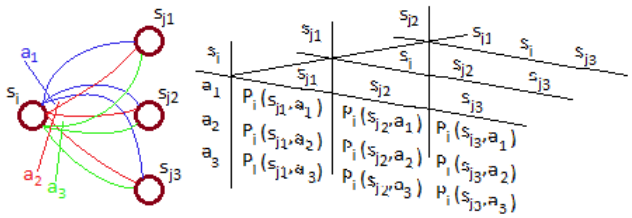

Figure 1: Hypergraph representation in a 3D array, detailing the existence of multiple overlapping edges between pairs of nodes for which differing actions may result in the same state–to–state transition.

are globally optimal and exhibit exponentially bound goal achievement rates, stochastic behavior is fully predictable, and we derive conditions for learning state–abstracted problems, present a metric for performance variance under abstractions and uncertainty, and prove that GAP training converges with reciprocal–form learning curves.

2. THE GAP ALGORITHM

In this section, we define terms related to the construction and analysis of the GAP algorithm, and present a robust analytical evaluation of its properties.

2.1 Definitions

Herein, we define all the components and terms which will be used in the succeeding sections, in particular those relating to the hypergraph modeling system, its associated auxiliary data structures, and the mechanisms used for planning.

A GAP agent registers a set of perceptual *states* (denoted $\mathcal{S}$) and may take a set of *actions* ($\mathcal{A}$), which can impact the world and possibly alter the state. At any given point in time k, the agent can observe an initial state, $s_i \in \mathcal{S}$, and subsequently take an action $a_l \in \mathcal{A}$, resulting in a state change to a final state s_f (note that s_f may be identical to s_i). Such a series is henceforth referred to as an *occasion*: $o_k = a_l(s_i) \rightarrow s_f$, conceptually treated differently than traditional state/action or state/action/state sets.

We implement a learning system for which the basic units are occasions within a 3-dimensional structure of size $|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|$, labeled INC. Cells at locations (s_i, s_f, a_l) $(INC[i, j, l])$ contain an instance count of the number of times the corresponding occasion has been observed. This data structure is conceptualized as a *directed hypergraph* (FIGURE 1): a higher dimensional analog of a graph with multiple edges between each node, corresponding to varying actions. Each action may have multiple results that overlap with other actions' results. Thus, we have multiple links between states, each connected with a different action. This structural change allows us to contend with the challenge of overlapping action results identified in [10].

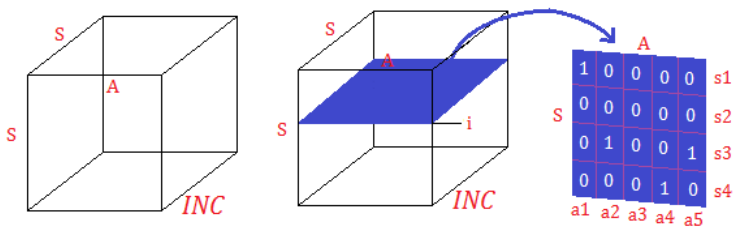

Figure 2: Extraction of an Action/State result slice from the hypergraph, illustrating the relationship between the a–priori state s_i and the potential results of taking actions from within that state.

We can calculate the probability of an occasion occurring as a ratio of the number of observed instances of the occasion to the sum of occasion counts along a slice of INC. However, as s_i is fixed but a_l and s_f are not, this presents two possibilities for probability models, one referenced against resultant states and one referenced against actions taken.

In the first model, the *a priori* probability, the most probable *outcome* of taking an action from a given state is referenced. Calculation of the associated probabilities is thus given by the following formula:

$$P(a_l(s_i) \rightarrow s_j) := P(s_j|s_i, a_l) = \frac{INC[s_i, s_j, a_l]}{\sum_{\forall s} INC[s_i, s, a_l]} \tag{1}$$

In the second model, *a posteriori* probability, we select actions based on the most probable cause of a given state/state transition, calculated:

$$P(a_l(s_i) \rightarrow s_j) := P(a_l|s_i, s_j) = \frac{INC[s_i, s_j, a_l]}{\sum_{\forall a} INC[s_i, s_j, a]} \tag{2}$$

To effect a net change among non-adjacent states, several actions must be taken. An ordered series of these transitions and the associated states we term a *sequence*. Solutions produced by the planning algorithm are sequences, represented as a pair of two ordered lists $\sigma_{og} = [\{s_1, s_2 ... s_g\}, \{a_1, a_2, ... a_{g-1}\}]$, where $a_1(s_1) \rightarrow s_2$, $a_2(s_2) \rightarrow s_3$, and so on.

For each occasion we will have the conditional probability which represents the likelihood of the occasion occurring. For a sequence, we can then define a joint probability of the entire sequence being executed:

$$P(\sigma) = \prod_{\forall o_k \in \sigma} P(a_l(s_i) \rightarrow s_j) \tag{3}$$

Which serves as our primary optimization criteria. With this, it is possible to define a class of subgraphs embedded within the hypergraph which must contain all edges of a solution. One such subgraph formulation which is computationally simple to construct and maintain contains all maximally probable transitions between any state pair (s_i, s_j), stored as an $|\mathcal{S}| \times |\mathcal{S}| \times 2$ array. The component $< s_i, s_j, 0 >$ is the maximum probability associated with the $s_i \rightarrow s_j$ transition, and component $< s_i, s_j, 1 >$ is the index of the corresponding action. Thus defined, we have:

$$AFI[s_i, s_j, 0] = \frac{INC[i, j, \underset{l}{\operatorname{argmax}}\{P(a_l(s_i) \rightarrow s_j)\}]}{\sum_{\forall s} INC[s_i, s, a_l]}$$

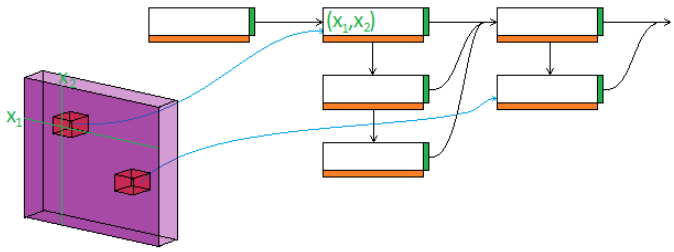

Figure 3: Array/linked list showing the indexed cell locations within the array containing pointers to the corresponding elements in the sorted linked list, which itself contains the data component associated with each array cell, and is organized into columns containing the same number of observed instances.

$$AFI[s_i, s_j, 1] = \underset{l}{\operatorname{argmax}}\{P(a_l(s_i) \rightarrow s_j)\}$$

Another such graph, prepared and maintained similarly, is one which contains maximally likely final states with respect to actions taken. This graph can be represented on an $|\mathcal{S}| \times |\mathcal{A}| \times 2$ sized array, in which members at $< s_i, a_l, 0 >$ represent the probability associated with the most likely result of taking action a_l from state s_i, and $< s_i, a_l, 1 >$ represents the index of s_f. Each of these compressed arrays represents a traditional graph, which feature we will use for efficient computation of solution sequences.

2.2 Datastructures & Algorithms

In this section, we begin by presenting the datastructures used to retain observed information, and then the algorithms which operate on these datastructures to identify solutions within the problem space.

We use a combination of an array with a linked list, as illustrated in FIGURE 3, such that each element in the array is a pointer to a member of the linked list containing that address' necessary data. In such a structure, each array element contains a pointer to a member link within the linked list and each such member, in addition to any other data, contains its corresponding location within the array.

In his way, the linked list need not be searched for member elements, and ordering of the list can be maintained using single operations on the linked list members. For our case, sorting is by incidence counts, and so we also implement the linked list in a parallel configuration with each 'column' containing instances with identical numbers of observed instances, so that each observation requires at most two operations to retain the list in sorted order.

A hypergraph may be stored in a 3-dimensional array, but for planning we choose a probability model as described above, where the planning algorithm need not evaluate all edges, only the most probable ones. We thus augment the hypergraph with a pair of array/linked lists corresponding to the sorted elements within the maximal likelihood subgraphs. The 3-dimensional array is paired with two 2-dimensional array linked lists, corresponding to the a priori and a posteriori probability

Algorithm 1 Linked list subgraph maintenence

```
function MaintainLL(INC, (s_i, s_f, a_l))
    occasionLink = INC[s_i, s_f, a_l, 0]

    if occasionLink.prev == None then
      return 1
    if occasionLink.prev.count
          > occasionLink.count then
      return 1
    if occasionLink.prev.count
          == occasionLink.count then
        occasionLink.prev = occasionLink.prev.prev
        occasionLink.post = occasionLink.current
        occasionLink.current =
          occasionLink.prev.current
```

metrics, each member pointing to a linked list which contains the sorted members of the slice along the compressed axis: the maximum likelihood subgraph.

FIGURE 4 shows how each cell in the 3-dimensional array contains two pointers, one to each subgraph compression, and each subgraph cell contains the linked list of members along the compression axis. For the a priori probability.

Because the linked list members each contain increment counts of the number of times the occasion has been observed, and are sorted by these counts, each link may only move ahead one link in the list at any time. As such, maintenance revolves around correctly rebuilding the link chain at each step, as detailed in Algorithm 1. The maximum likelihood subgraph slice of the bulk hypergraph data structure is thus perpetually embedded within the array/linked lists. Addressing to the linked

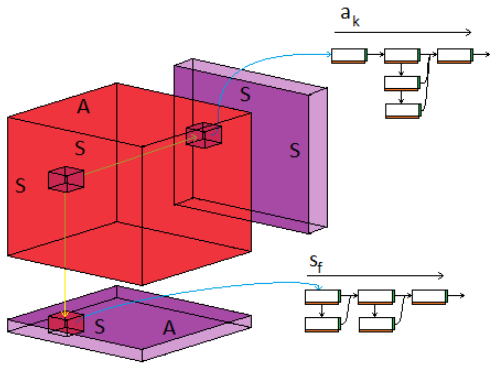

Figure 4: Augmented hypergraph data structure: a 3 dimensional array, each cell of which contains pointers to members of two Array/Linked List objects, each containing a pointer to the corresponding sorted list associated with that state/action or state/state pair, allowing for immediate retrieval.

Algorithm 2 Sequence inference algorithm

function SequenceInfer(AFI, (s_i, s_g))
  $bound \leftarrow s_i.edges$
  $perm \leftarrow [(s_i, 1.0)]$
  $edges \leftarrow []$
  while $s_g \notin permanent$ **do**
    $jointProb(s_j) := perm[bound[j]][1] \cdot bound[j].P$
    $s_{maxP} \leftarrow \underset{s_j}{\text{argmax}}(jointProb)$
    $perm \leftarrow (s_{maxP}, jointProb(s_{maxP}))$
    $bound = (bound \cup s_{maxP}.edges) - [e|e(1) = s_{maxP}]$
    $edges \leftarrow bound[s_j]$
    $solution = [edges[s_g]]$
  while $solution[-1][0] \neq s_i$ **do**
    $solution \leftarrow edges[solution[-1][0]]$
  return $solution$

list element is direct via pointer, and comparison of the increment counts require only the prior and current list members, and thus each update's complexity is $O(1)$.

To identify the maximally probable path, we implement a modified version of Dijkstra's algorithm adapted to find maximum probability (rather than minimum weight) subtrees rooted at s_i using the Array Linked Lists of AFI. Algorithm 2, formalizes this, calculating the net probability as a result of each sequential action, expressed in Equation 3. Due to the structure of the augmented hypergraph, the computational efficiency of this method is $O(|\mathcal{S}|^2)$ for the a priori probability model and $O(|\mathcal{S}| \cdot |\mathcal{A}|)$ for the a posteriori model.

Phrased by analogy to Markov Decision Processes, this algorithm produces a policy $\pi(s_i)$ such that the action taken at any step is the first action in the maximal probability sequence between s_i and s_g, or:

$$\pi(s_i) = \left(\underset{\sigma_{ig}}{\text{argmax}} \prod_{\forall o_j \in \sigma_{ig}} P(o_j) \right) \Bigg|_{k=0} \tag{4}$$

the first action in the most–probable sequence σ_{ig} from state i to state g (even if the ideal state–to–state transition was not achieved at the prior step). Note that, in this analogous formulation, no reward term exists within the policy function.

We can see from this implementation that the decision–making process of GAP agents is imminently transparent and explainable. For any plan, the actions are based on the probability-maximizing policy and the corresponding state-to-state transitions intended by the sequence of actions is readily available from the algorithm. Further, it is clear at this stage that a path between any pair of reachable states can be achieved, making the learned AFI arrays fully independent of goal choice.

2.3 Analysis of the GAP Algorithm

In this section, we analyze the performance of the GAP algorithm as defined above to show optimality and efficacy, determine agent dynamics, the effects of abstractions on performance, and learning convergence properties. For our analysis, we rely heavily on Markov process analysis techniques, with the critical difference that we have defined a generalized action policy, Equation 4, rather than derive one from analysis of a problem space.

2.3.1 Optimality of GAP plans

We begin by demonstrating the optimality of Equation 4 as a policy. We first demonstrate that the optimal path is embedded within the subgraph.

Theorem The solution with maximum joint probability within a hypergraph is embedded within the maximum likelihood subgraph.

Proof We proceed by contradiction. Presume that there exists an optimal solution sequence σ_{og} which contains an occasion not allocated to the maximum likelihood subtree. In this case, by definition the occasion must have an associated probability less than that of the corresponding transition in the subgraph. However, because probabilities are necessarily monotonically decreasing, the sequence σ'_{og} using the subgraph's instance for the given transition will have higher probability than the assumed solution, and thus σ_{og} is not optimal.

Note that this proof applies to either probability model: that each graph contains a maximal slice with respect to either projection is sufficient. Given that the optimal path is then known to be embedded within the maximum likelihood subgraph, we can demonstrate that the inference algorithm extracts the maximally likely path:

Theorem The sequence inference algorithm extracts the σ_{og} representing the maximal joint probability sequence representing a path from s_i to s_g.

Proof Consider that all probabilities are on the range $[0, 1]$, and that the joint probability function (Equation 3) is therefor monotonically decreasing. We proceed by induction on the distance from s_i. The first node selected will have the maximum probability edge of all leading from s_i to s_{i+1}, and thus any alternate path to this node is bounded by that single probability. Continuing on, at any point in the sequence, each successive joint probability is further bound by the product of the prior and current occasion. As such, any higher probability bound occasion would have to be off of the maximum probability tree in AFI, a contradiction to Theorem 1.

The action policy derived from planning on the maximal probability subgraphs is thus optimal, and the need for designation of a reward function to drive convergence is unnecessary. These proofs reflect those of [7], for SSP, however are simplified substantially by the structure of the maximum likelihood subgraph.

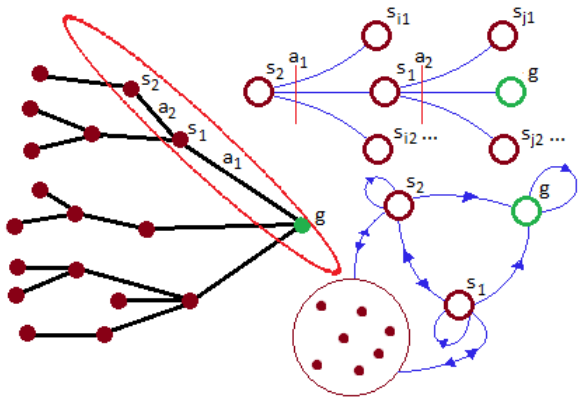

Figure 5: Conversion of Maximal Probability path tree to Markov Chain. highlighting states s_2, s_1, and g, with maximal probability actions a_2 and a_1 linking them, and additional possible results of taking those actions leading to other states besides the intended result, shown in the Markov network

2.3.2 Predictive behavior analysis

For planning purposes we proceeded by finding the highest probability expected path to the goal, but for this analysis, we will not keep a specific starting state in mind. Instead, we build the tree of maximal probability paths rooted in the (arbitrary) goal state. We denote this tree as $T_{P(g)}$. This tree will contain all maximal σ_{ig}.

Within this tree, each maximally likely transition indicates the associated action most likely to effect that transition: the action the agent will choose when in that state via Equation 4. However, because each action is assumed to be non-deterministic, it will include probabilities for arriving at non–intended states as well: $a_l(s_i) : \{(s_{j1}|P_{j1}), (s_{j2}|P_{j2}), ...\}$. We construct the transition table from these segments of INC for all states, each having a stochastic vector associated with it: $\vec{t}_i = AFI[s_i, \pi(s_i), :]$. For s_g the operation of the agent effectively terminates, and so $\vec{s_g}$'s entries are 0 excepting the entry $\vec{s_g}[g]$, which is 1; $\vec{t}_g = \left[0\ 0...\ 1..\ 0...\right]^T$, adopting the absorbing state method of [6]. From this concatenation, we have the transition table, P_g, which enables us to acquire a picture of system behavior over time. We concatenate all these states to acquire the transition table:

$$P_g = \left[\vec{t}_0\ \vec{t}_1 ...\ \vec{t}_g ...\ \vec{t}_j\right] \tag{5}$$

A graphical representation of this conversion is illustrated in FIGURE 5.

It is important to note here that the conversion does not result in tree–like structure in P_g: non-optimal transitions are also embedded so long as they result from *optimal actions*. However, sub-problem optimality is retained: the subsequent path in $T_{P(g)}$ from an incidental non–optimal state is optimal with respect to the new state.

We can model statistical propagation by representing the state distribution itself as a vector, $\vec{s_k}$, where k is taken to be step number. At $k = 0$, we have the known starting state at $P(s_0) = 1.0$, and so $\vec{s_0}$ will also be zero vector excepting the s_0 element, which is 1. The state occupation distribution

as a function of step time is then given by:

$$\vec{s_k} = P_g^k \cdot \vec{s_0} \tag{6}$$

which represents the stochastic vector of probable states evolved from s_0 over time; and further the corresponding column of P_g^k represents the probability of state occupation at step k for the given starting state. Within P_g, columns represent probability vectors over states, such that $\Sigma_j P_g[i,j] = \vec{1}_{1\times|\mathcal{S}|}$. Consequently, $||\vec{s_k}||_1 = 1$, which is sensible as it is a probability vector. Now, because $\vec{s}_k = P_g \cdot \vec{s}_{k-1} = P_g(P_g^{k-1} \cdot \vec{s}_0)$, we can define the stationary state distributions by $||\vec{s}_k - \vec{s}_{k-1}||_1 < \epsilon$, or: $P_g \cdot \vec{s}_{k-1} \leq (1+\vec{\epsilon})\vec{s}_{k-1}$, which further implies that:

$$P_g^{k-m+1} \cdot \vec{s}_{k-1} \leq (1+\vec{\epsilon})^k \vec{s}_0$$

Which necessitates that any attractor state *either* be an eigenvector of P_g with $\lambda = 1$, or rbitrarily close to an initial state. $\lambda = 1$ eigenvectors of any transition matrix are also steady states. However, P_g has has no steady-state *distribution* by virtue of the presence of s_g. Presume that we arrange P_g such that s_g corresponds to the last element. We then have:

$$P_g = \begin{pmatrix} T_s & \vec{0} \\ \vec{t_g} & 1 \end{pmatrix}$$

Where T_s is the transition matrix internal to only non-goal states, $\vec{t_g}$ is the vector of transition probabilities from $\{s_i \in S | i \neq g\}$, and the final column is the stochastic vector of s_g. We can then write:

$$P_g^k = \begin{pmatrix} T_s^k & \vec{0} \\ \vec{t_g} \cdot \Sigma_{l=1}^{k-1} T_s^l + \vec{t_g} & 1 \end{pmatrix} \tag{7}$$

This expression precisely describes the probability distribution of the agent's occupation of states as a function of step time. From it, we can see that the probability of reaching the goal state at step k is given by

$$\vec{P}(s_i \rightarrow s_g | k) = \vec{t_g} \cdot \sum_{m=1}^{k-1} T_s^m + \vec{t_g} \tag{8}$$

We can further note that for a given state distribution $\vec{s}_k$, at time k we can express the probability of transition to the goal state at some future time $k' = k + \delta k$:

$$P(s_g | \vec{s}_k, k') = (\vec{t_g} \cdot \sum_{m=1}^{\delta k-1} T_s^m + \vec{t_g}) \cdot (\vec{s}_k) \tag{9}$$

Because T_g is strictly positive definite, $P(s_i \rightarrow s_g|k)$ is monotonically increasing in k, and therefor P_g has no steady state. This means that s_g must be an attractor state, as it is identical to its own start-state distribution. Further, no other state can be an attractor *unless* there is a zero probability of transitioning out from that state. Such states may be present in P_g due to the stochastic nature of a_l. We can therefore define the probability of reaching the goal state at any given step number, establish the expected number of steps to reach the goal, and calculate the probability of the agent being sequestered at non-goal attractors.

Because the columns of P_g are stochastic, we can make the following relation:

$$\vec{1}_{1\times|\mathcal{S}|} - \vec{1}_{1\times|\mathcal{S}|} T_s^k = \vec{t_g}(\Sigma_{m=1}^{k-1} T_s^m + I_{|\mathcal{S}|})$$

$$\vec{1}_{1\times|\mathcal{S}|} = \vec{1}_{1\times|\mathcal{S}|}T_s^k + \sum_{m=1}^{k-1} \vec{t_g}T_s^m + \vec{t_g} \tag{10}$$

which bounds the probability distribution of system states as a function of step time. Because there are no steady states, barring those constructed in the same form as a goal state, $< \vec{0}, 1, \vec{0} >$, and because T_s^k is positive definite, the probability distribution of goal transitions must be strictly monotonic over time.

2.3.3 Trap nets

We have assumed the form of the goal state as $< \vec{0}, 1 >$, which makes it an attractor state, and other states with unit self–transition probabilities are the only unwanted attractor states. However, it is also possible that sequences which present no path to the goal once reached to exist. We will refer to such sets as 'trap nets', illustrated in FIGURE 6.

Defining a subset, *tnet*, to represent these states, we can organize P_g for analysis by aligning the rows and columns associated with the subnet *tnet*, noting that states in *tnet* can transfer between one another, but not to other states not in *tnet*:

$$P_g = \begin{pmatrix} T_{s\notin tnet} & \mathbf{0} & \vec{0} \\ T_{s\in tnet} & T_{tnet} & \vec{0} \\ \vec{t}_{g|i\notin tnet} & \vec{0} & 1 \end{pmatrix}$$

Which we can expand into the successive probability distribution:

$$P_g^k = \begin{pmatrix} T_{s\notin tnet}^k & \mathbf{0} & \vec{0} \\ \Sigma_{j=0}^{k-1} T_{tnet}^j T_{s\in tnet} T_{s\notin tnet}^{k-1-j} & T_{tnet}^k & \vec{0} \\ \Sigma_{j=0}^{k-1} \vec{t}_{g|i\notin tnet} T_{s\notin tnet}^j & \vec{0} & 1 \end{pmatrix} \tag{11}$$

From which we can see that $P(s_{i\in tnet} \rightarrow s_g|k) = \vec{0}$ for all k. Further, define a system parameter L_{max}: the longest minimal path between any state and the goal. For any reachable state s_i, $P(s_i \rightarrow$

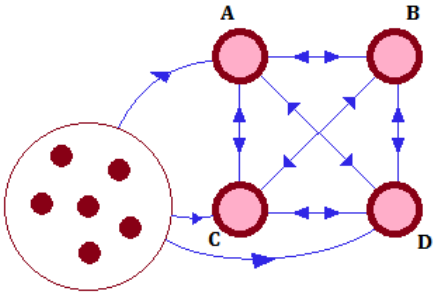

Figure 6: Illustration of subgraph segment from which no path to the goal exists, yet contains multiple transition state cycles. Such regions can present non-steady state attractors from which the agent cannot progress to the goal, hence being considered 'trapped' in the subgraph.

$s_g|L_{max}) > 0$: There is a non-zero probability that $s_i \rightarrow s_g$ has occurred after L_{max} timesteps. We can test if a state is a member of a trap net, as the final row of $P_g^{L_{max}}$ will contain only 0 probability entries in states from which the goal is unreachable. We might determine L_{max} by analysis of P_g, but it is sufficient to raise P_g to a power greater than L_{max}. The maximum path length is bounded by the size of the graph, and so any state i for which $P(s_g|\vec{s}_i, k = |\mathcal{S}|) = 0$ is necessarily a member of a trap net.

We can then use Equation 6 to determine the explicit probability at any point in time that the system has become stranded in a trap net, using Equation 11:

$$P(s_t \in tnet|k) = \vec{1}_{1\times|tnet|} \cdot \left(\sum_{j=0}^{k-1} T_{tnet}^{j} T_{s\in tnet} T_{s\notin tnet}^{k-1-j}\ T_{tnet}^{k}\ \vec{0}\right) \cdot \vec{s}_t$$

The identification of single attractor states and trap nets together provides a rigorous analysis of the reachability of the goal from all other states as a statistical distribution of time for all states, extending the connectivity approach as used in [14]. Combined with Equation 6, the goal–convergent behavior of the agent can be fully, explicitly defined, resolving the problem discussed explicitly in [6].

2.3.4 Derivation of bounded time performance

The evolution of the L1 norm of T_s^k is a useful metric, as the L1 norm is intimately related to the sum of the columns in a matrix, and thus the joint probability of stochastic vectors arranged in matrix form. As with all vector induced norms, L1 is submultiplicative, and so:

$$\|T_s^k\|_1 \leq \|T_s\|_1^k$$

Because all columns are stochastic, the maximum absolute column sum is paired with the minimum probability single step goal transition, such that:

$$\|T_s^k\|_1 \leq (1 - \min_{\forall a,i} P_g[s_i, g, a])^k$$

For many systems, $\min_{\forall a,i} P_g[s_i, g, a] = 0$, which provides little insight, but at $k = L_{max}$, all states from which the goal is reachable have a non-zero transition probability:

$$\begin{cases} \|T_s^k\|_1 = 0 & k < L_{max} \\ \|T_s^k\|_1 \leq \|T_s^{L_{max}}\|_1^{k-L_{max}} & otherwise \end{cases} \tag{12}$$

Allowing the calculation of the minimum time at which all states surpass a threshold likelihood reaching the goal, without projecting the system forward in time. In general, for some minimum transition probability threshold P_{thresh}:

$$\begin{aligned} 1 - P_{thresh} &\leq \|T_s^{L_{max}}\|_1^{k_p - L_{max}} \\ k_p &\geq \frac{log(1 - P_{thresh})}{log(\|T_s^{L_{max}}\|_1)} + L_{max} \end{aligned} \tag{13}$$

Which establishes the progression curve of states towards the goal in terms of T_s and the effective 'distance' between the starting state and the goal. Writing the relation slightly differently, as $1 - \|T_s^{L_{max}}\|_1^{k_p - L_{max}} \leq P_{thresh}$, shows the probability of transition to goal is bounded by an exponential growth rate, illustrating that even under stochastic disturbances the probability of reaching the goal rises to unity at an exponential rate.

2.3.5 Analysis of robustness under perturbation

Validation against an abstract model is often challenging, and learning algorithms often implement some level of inherent abstraction. We will consider an abstracted learning problem one with a mapping $\alpha()$ transforming a state space S into a mixed space $\alpha(S)$. α need not be strictly surjective, but for purposes of analysis, we will consider only state pairs in the domain of S and the codomain $\{\alpha(s_i)|\forall s_i\}$. This probabilistic mapping model is similar to that used by [11]. Note that a map modeling uncertainty as a mixing transfor

Define an $|\alpha| \times |\mathcal{S}|$ transformation matrix, α_T, where $\alpha[j,i]$ the probability $P(\alpha(s_i) = \alpha(j))$ that the i^{th} 'true' state is mapped onto the j^{th} abstracted state. For a state probability vector $\vec{s}_t$, the corresponding probability vector in the abstracted state space is given by $\vec{s}_{\alpha t} = \alpha_T \cdot \vec{s}_t$, or, for general time propagation: $\vec{s}_{\alpha t} = \alpha_T \cdot P_g^t \cdot \vec{s}_0$. Given an AFI subgraph for the abstracted space, P_α, we also have $\vec{s}_{\alpha t} = P_\alpha^t \vec{s}_{\alpha 0}$, and since $\vec{s}_{\alpha 0} = \alpha_T \vec{s}_0$ we can construct a relation from the equivalence $\alpha_T P_g^t = P_\alpha^t \alpha_T$:

$$\begin{cases} P_\alpha^t = \alpha_T P_g^t \alpha_T^+ \\ P_g^t = \alpha_T^+ P_\alpha^t \alpha_T \end{cases}$$

Where α_T^+ is the pseudoinverse of α_T. These relations convert the *probability space* of the abstraction into that of the grounded problem. This transform does not recover the true *state space*, even if α_T is known perfectly, as α_T^+ cannot unmix states which are combined. Mathematically, this is realized by α_T^+ not being strictly positive definite.

We can note that: $P_\alpha^t = \alpha_T P_g^t \alpha_T^+ = (\alpha_T P_g \alpha_T^+)^t$. For $t = 2$, $\alpha_T P_g^2 \alpha_T^+ = \alpha_T P_g \alpha_T^+ \alpha_T P_g \alpha_T^+$, or $\alpha_T^+ \alpha_T = I$, implying that the columns of α_T must be linearly independent. We can thus derive useful partitions by recognizing that both arrays must be stochastic transforms:

$$\alpha_T = \begin{pmatrix} \alpha_{Ts} & \alpha_{Tg} \\ \vec{1} - \vec{1}\alpha_{TS} & 1 - \vec{1}\alpha_{Tg} \end{pmatrix}, \alpha_T^+ = \begin{pmatrix} \alpha_{Ts}^+ & \alpha_{Tg}^+ \\ \vec{1} - \vec{1}\alpha_{TS}^+ & 1 - \vec{1}\alpha_{Tg}^+ \end{pmatrix}$$

Assuming that the abstracted model is convergent, we will derive conditions for the 'true' system to converge as well. Taking Equation 7 where we annotate: $t_{\vec{\alpha g}} \cdot \sum_{l=1}^{k-1} T_{\alpha s}^l + t_{\vec{\alpha g}} = V_p$:

$$P_g^k = \begin{pmatrix} T_s^k & \vec{0} \\ \vec{P}(s_i \to s_g|k) & 1 \end{pmatrix} = \begin{pmatrix} \alpha_{Ts}^+ & \alpha_{Tg}^+ \\ \vec{1} - \vec{1}\alpha_{TS}^+ & 1 - \vec{1}\alpha_{Tg}^+ \end{pmatrix} \cdot \begin{pmatrix} T_{\alpha s}^k & \vec{0} \\ V_p & 1 \end{pmatrix} \cdot \begin{pmatrix} \alpha_{Ts} & \alpha_{Tg} \\ \vec{1} - \vec{1}\alpha_{TS} & 1 - \vec{1}\alpha_{Tg} \end{pmatrix}$$

Expanding P_g^k lets us calculate the probability of goal transition in the true space:

$$\vec{P}(s_i \to s_g|k) = V_p \alpha_{Ts} - \vec{1}\alpha_{Tg}^+ V_p \alpha_{Ts} + \vec{1} T_{\alpha s}^k \alpha_{Ts} - \vec{1}\alpha_{Ts}^+ T_{\alpha s}^k \alpha_{Ts} + \vec{1} - \vec{1}\alpha_{Tg}^+ \vec{1} - \vec{1}\alpha_{Ts} + \vec{1}\alpha_{Tg}^+ \vec{1}\alpha_{Ts}$$

using the relations $\vec{1} T_{\alpha s}^k = 1 - V_p$, and $\vec{1}\alpha_{Tg}^+ = ||\alpha_{Tg}^+||$ we can re-cast this expression as:

$$\vec{P}(s_i \to s_g|k) = \vec{1} + ||\alpha_{Tg}^+||(\vec{1}\alpha_{Ts} - \vec{1}) - ||\alpha_{Tg}^+|| V_p \alpha_{Ts} - \vec{1}\alpha_{Ts}^+ T_{\alpha s}^k \alpha_{Ts}$$

We presumed that P_α is convergent, and thus we can note the limiting behavior of $T_{\alpha s}^k$ and V_p:

$$\begin{cases} \lim_{k\to\infty} V_p = \vec{1} \\ \lim_{k\to\infty} T_{\alpha s}^k = \mathbf{0} \end{cases}$$

From which the limiting behavior of $\vec{P}(s_i \rightarrow s_g|k)$ can be determined:

$$\lim_{k\to\infty} \vec{P}(s_i \rightarrow s_g|k) = \vec{1} - ||\alpha_{Tg}^{+}||\vec{1}$$

Convergence of P_g can be expressed as $\vec{P}(s_i \rightarrow s_g|k) \rightarrow \vec{1}$, so: $0 = ||\alpha_{Tg}^{+}||$, showing that convergence of the true system, given convergence of the abstracted system, is predicated on the transform between the true and abstracted goal states being *onto*. This condition is analogous to, but distinct from, the convergence conditions derived in [14], and mirrors the observability model in [13]. GAP solutions will thus retain optimality and convergence properties in transformed spaces when this condition is met.

2.3.6 Impact of perturbed state on performance

We can further extend the derivations in Section 4.2 to the case of performance of an abstracted system, assuming the condition derived above. Beginning with the relation $||T_s^k||_1 \leq ||T_s||_1^k$ for the true state system:

$$||T_s||_1^k \geq ||\alpha_{Ts}^{+}T_{\alpha k}\alpha_{Ts}+\alpha_{Tg}^{+}V_p\alpha_{Ts}+\alpha_{Tg}^{+}\vec{1}-\alpha_{Tg}^{+}\vec{1}\alpha_{Ts}||_1 \geq ||\alpha_{Ts}^{+}T_{\alpha k}\alpha_{Ts}||_1+||\alpha_{Tg}^{+}||_1(1-||T_{\alpha k}||_1||\alpha_{Ts}||_1)$$

Because α_{Tg}^{+} is a vector, $||\alpha_{Tg}^{+}||_1 \geq ||\alpha_{Tg}^{+}||$, and $||T_{\alpha k}||_1$, $||\alpha_{Ts}||_1$ are submatricies of stochatic matricies, they are strictly in $[0, 1]$ (though this is not the case for $||\alpha_{Ts}^{+}||_1$, and so this derivation applies only $P_\alpha \rightarrow P_g$ and not to $P_g \rightarrow P_\alpha$: convergence of the abstracted model implies convergence of the true model, but not the converse), so:

$$||T_s||_1^k \geq ||\alpha_{Ts}^{+}T_{\alpha k}\alpha_{Ts}||_1+||\alpha_{Tg}^{+}||(1-||T_{\alpha k}||_1||\alpha_{Ts}||_1) \geq ||\alpha_{Ts}^{+}T_{\alpha k}\alpha_{Ts}||_1 = ||\alpha_{Ts}^{+}||_1\cdot||T_{\alpha k}||_1\cdot||\alpha_{Ts}||_1$$

From this inequality, we can then replicate Equation 13 for the abstracted case:

$$1 - P_{thresh} \leq (||\alpha_{Ts}^{+}||_1 \cdot ||T_{\alpha k}||_1 \cdot ||\alpha_{Ts}||_1)^{k_{p\alpha}-L_{max}}$$

$$k_{p\alpha} \geq \frac{log(1 - P_{thresh})}{log(||\alpha_{Ts}^{+}||_1 \cdot ||T_{\alpha k}||_1 \cdot ||\alpha_{Ts}||_1)} + L_{max} \tag{14}$$

Which describes how the abstraction modifies the expected efficiency of the agent. By examining this expression, we can make some inferences about the impact of α_T on performance:

$$\begin{cases} k_{p\alpha} > k_p \ ||\alpha_{Ts}||_1 \cdot ||\alpha_{Ts}^{+}||_1 < 1 \\ k_{p\alpha} \leq k_p \ ||\alpha_{Ts}||_1 \cdot ||\alpha_{Ts}^{+}||_1 \geq 1 \end{cases} \tag{15}$$

We can use the product above as a rough measure of the 'quality' of an abstraction, the degree to which it effects performance, by defining:

$$Q(\alpha_T) = \frac{1}{||\alpha_{Ts}||_1 \cdot ||\alpha_{Ts}^{+}||_1}$$

$Q(\alpha_T)$ is directly correlated to the impact α_T has on performance, resolving the metric problem brought up in [16]. Because α_T^{+} is not strictly positive definite, α_T which improve performance are possible, albeit difficult to design. An abstraction may either improve or reduce efficacy, depending on the nature of the abstraction, which may seem counter-intuitive, but consider the

way a substitution may reduce the number of steps needed to solve an algebraic equation. Certain simplifications may bias portions of the graph towards choosing equally likely but shorter paths, minimizing stochastic variance, an observation also noted by [11].

We can approximate this measure using the relation between k_p and the measured $k_{p\alpha}$ as the average number of steps to reach the goal over many iterations:

$$\frac{k_{p\alpha} - k_p}{k_p - L_{max}} = \frac{log(||\alpha^+_{Ts}||_1 \cdot ||\alpha_{Ts}||_1)}{log(||T_{\alpha k}||_1)}$$

$$||T_{\alpha k}||_1^{\frac{k_p - k_{p\alpha}}{k_p - L_{max}}} = Q(\alpha_T) \tag{16}$$

This provides a metric of the relative efficacy of the abstraction from measurable values, specifying the impact on performance with a perturbation model and underwriting the effectiveness of the GAP for operating under an abstraction or uncertainty.

2.3.7 Learning convergence

The behavior of the agent as a learning system can be modeled by treating the learned AFI matrix as an abstracted function of the true state which becomes more accurate as learning progresses. The initial transform maps true states onto a uniform distribution from which actions are initially chosen randomly:

$$\begin{cases} \alpha_{T1} = \frac{1}{|\alpha|} \cdot \mathbf{1} \\ \alpha^+_{T1} = \frac{1}{|\mathcal{S}|} \cdot \mathbf{1} \end{cases}$$

We can see that: $||\alpha_{T1}||_1 = \frac{|\alpha|-1}{|\alpha|}$ and $||\alpha^+_{T1}||_1 = \frac{|\mathcal{S}|-1}{|\mathcal{S}|}$. Further, labeling P_g as the asymptotic matrix and P_α the non-learned array, $|\alpha| = |\mathcal{S}|$. We can approximate the expected learning curves with an amortized update at each step k derived from Equation 2. An update to a single state vector in the average case has the state visited $\frac{k}{|\mathcal{S}|}$ times, and the individual visit counts can be expressed as $\frac{k}{|\mathcal{S}|}\vec{s}_{\alpha i}$, also via Equation 2. The distribution for the increase in counts can be expressed by $\vec{s}_i$ (the asymptotic learned behavior), the corresponding expectation of the column in P_g. Combining all this for $\frac{k+1}{|\mathcal{S}|}$ steps gives:

$$\vec{s'}_{\alpha i} = \left(\frac{k}{|\mathcal{S}|}\vec{s}_{\alpha i} + \frac{1}{|\mathcal{S}|}\vec{s}_i\right) \cdot \frac{1}{\frac{k}{|\mathcal{S}|} + \frac{1}{|\mathcal{S}|}} = \frac{k\vec{s}_{\alpha i} + \vec{s}_i}{k+1}$$

$$\delta\vec{s}_{\alpha i} = \vec{s'}_{\alpha i} - \vec{s}_{\alpha i} = \frac{\vec{s}_i - \vec{s}_{\alpha i}}{k+1}$$

Which, in aggregate, gives the expression across the full transition array:

$$\delta P_{\alpha k} = \frac{P_g - P_{\alpha k}}{k+1}$$

For the recurrence relation:

$$\begin{cases} P_{\alpha k+1} = \frac{kP_{\alpha k} + P_g}{k+1} \\ P_{\alpha 1} = \frac{1}{|\mathcal{S}|^2} \cdot \mathbf{1} \cdot P_g \cdot \mathbf{1} \end{cases}$$

$$P_{\alpha k} = \left[\frac{\mathbf{1} \cdot P_g \cdot \mathbf{1}}{k|\mathcal{S}|^2} + \frac{k-1}{k} P_g \right] = \alpha_{Tk} P_g \alpha_{Tk}^{+} \tag{17}$$

Which we can express in similar block fashion as above:

$$\frac{1}{k|\mathcal{S}|} \begin{pmatrix} \mathbf{1} + |\mathcal{S}|(k-1)T_s & \vec{1} \\ \vec{1} + |\mathcal{S}|(k-1)P(g) & 1 + |\mathcal{S}|(k-1) \end{pmatrix} \alpha_{Tk} = \alpha_{Tk} P_g$$

And calculate the non-goal block of each side, using the general form for α_{Tk}

$$\left(\frac{1 - k|\mathcal{S}|}{k|\mathcal{S}|} \mathbf{1} + \frac{k-1}{k} T_s \right) + \mathbf{1}\alpha_{Ts}^{+} = \alpha_{Ts} T_{\alpha s} \alpha_{Ts}^{+} + \alpha_{Tg} V_P \alpha_{Ts}^{+}$$

$$\left[\frac{\mathbf{1}_{|\mathcal{S}|-1}}{k|\mathcal{S}|} + \frac{k-1}{k} T_s \right] - \alpha_{Tg}\vec{1} + \alpha_{Tg}\vec{1}\alpha_{Ts}^{+} = \alpha_{Ts} T_s \alpha_{Ts}^{+} + \alpha_{Tg} P(g) \alpha_{Ts}^{+}$$

Because the right sides of both equations above are equivalent we can equate and simplify, and in the limit case where $k \to \infty$:

$$(\mathbf{1} - \alpha_{Tg}\vec{1})\alpha_{Ts}^{+} = \mathbf{1} - \alpha_{Tg}\vec{1} \to \alpha_{Ts}^{+} = \mathbf{I} \to \alpha_{Ts} = \mathbf{I}$$

Showing that as P_α is learned, the abstraction transform approaches the identity, and thus GAP agent training will be convergent on basis of Equations 2 and 10.

2.3.8 Derivation of learning curve form

We can also demonstrate that learning will be efficient by deriving the average form of the learning curve. Equation 17 allows us to determine the amortized form of the transition array:

$$P_{\alpha k} = \frac{1}{k} \left(\frac{\mathbf{1}}{|\mathcal{S}|} - P_g \right) + P_g$$

In which the terms $\frac{\mathbf{1}}{|\mathcal{S}|} - P_g$ and P_g are time invariant, dynamic behavior governed by the reciprocal of the timestep, $\frac{1}{k}$. The average learning curve will thus follow a reciprocal pattern $k_{p\alpha}(k) = A\frac{1}{k} + B$, where B is the asymptotic path to goal length, k_p. To determine A, we can evaluate the initial behavior of the system given the form for α_{T1} and Equation 17, $k_{p\alpha}(1) - k_p = A$:

$$A = (k_p - L_{max}) \frac{2log(|\mathcal{S}|) - 2log(|\mathcal{S}| - 1)}{log(||T_{\alpha 1}||_1)}$$

$||T_{\alpha 1}||_1$ can be directly calculated from $\alpha_{T1} P_g \alpha_{T1}^{+}$ as $\frac{|\mathcal{S}|-1}{|\mathcal{S}|}$, so $A = 2(L_{max} - k_p)$, and:

$$k_{p\alpha}(k) = \frac{2(L_{max} - k_p)}{k} + k_p \tag{18}$$

Establishing the performance of the unlearned system as a random walk with $k_{p\alpha}(1) = 2L_{max} - k_p$, and the average learning curve as an offset reciprocal function of epoch number, showing that learning will be efficient, and tied to bounded system characteristics.

3. DEMONSTRATION CASES

In this section we explore thebehavior of GAP agents in example problem classes. These tasks includes specific hierarchical components, complex and large state spaces, and have been used previously as benchmark trials for machine learning algorithms, initially in [17], for the TAXI domain, Mazes by [18], and the Tower of Hanoi by [19].

3.1 Experimental procedures

Prior to detailing the experiments themselves, we outline the common procedures used across all trials which are not specific to any one problem case.

The AFI datastructure is initialized with a uniform distribution of random values, as in Section 4. The agent proceeds in the simulation environment acting on basis of the current state of the AFI array until achieving the goal state, terminating the epoch, upon which event the simulation world is reset and the INC array persisting between epochs. We calculate plans using the a priori probability model.

To investigate system performance under uncertainty, we also artificially induce error in some trials, with a random threshold process executing a random non–planned action. Action modifications are selected because discontinuous transitions do not model real world uncertainties well, and because if there is an error in state detection, the agent will take action based on the fault state: an action independent of the current state.

Because we are intending to make a fully abstracted learning agent, every agent is built without labeled states. Simulation models output string reports when polled, and from these a hash lookup table is generated. As more states are discovered, the size of the augmented hypergraph is increased to accommodate larger lookup tables. The total state space occupation for the GAP agent will thus only include observed states, using sparsity to our advantage. We indicate the size of the accumulated state space with the $|\mathcal{S}|_{max}$ value- the largest number of states observed throughout the training process.

Demonstration of effectiveness is made by comparing accuracy of previously defined performance measures. We calculate best fit equations for the learning curves and measure of their accuracy: R^2 for the fit of $k_{p\alpha} = \frac{A}{k} + B$, and percentage off–linear ("%OL") averages of linear regressions on the plots of $(\frac{1}{k}, k_p)$ calculated for N sequential data points on the curve as: $\sum_{\forall n \in N} \frac{1}{N} \frac{|k_p[n] - (A\frac{1}{n} + B)|}{k_p[n]}$. We also compare approximations of k_p and L_{max} to correlate the analysis in Section 4 and the measured data. k_p, calculations by computation of the fit curve for Equation 18 ("k_p I"), and average performance after convergence ("k_p II"). L_{max} comparisons are made between Equation 16 ("L_{max} I") and Equation 13 ("L_{max} II"), estimated over levels of P_{thresh}, identifing the values for P_{thresh} at which the inequality is no longer valid.

To put context on the effectiveness of the GAP algorithm, we compare it to two baselines- Q Learning, and Markov Decision Processes, chosen because the GAP algorithm includes both learning and stochastic optimization. Comparisons to Q Learning are made to illustrate the speed and flexibility of learning, and MDPs to illustrate the efficacy of that learning. QL and MDPs require reward

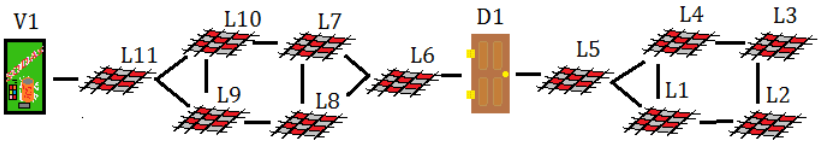

Figure 7: Illustration of a basic STRIPS-style world, containing linked location states (L_i) and multiple independent actionable states (V_1 and D_1)

functions to converge, so to ensure they reflect the same system, we implement the reward function: $R(s_i, a_l) = log(P(a_l(s_i) \rightarrow s_{i+1})) + log(P(\sigma(i+1, g)))$ which mimics the GAP algorithm's optimization. In our results, "QL k_p" is the average number of steps to reach the goal for the trained QL agent, "QL Ep." is the number of epochs for the QL agent to converge, where "NC" indicates failure to converge after 1000 epochs (1000 epochs chosen as the cut-off by being approximately 2 orders of magnitude greater than the GAP convergence period observed in all pilot experiments). "MDP k_p" is the average shortest path to the goal found by an MDP planner using Value Iteration.

3.2 STRIPS-type Problems

We begin with a STRIPS-type planning problem, schematically represented in FIGURE 7. The agent can take set of move operations which translate it, a pair of world manipulating actions (to fetch an item and open a door), an internal state (possession of an item), an external state (location) and a hidden state (status of the door), with $|\mathcal{S}|_{max} = 52$, and a random startign location. This represents a hierarchically ordered workspace, critical because expressing functionality within such problems is a key milestone for learning algorithms.

Displayed in FIGURE 8 is the average learning curve, derived over 1000 iterations beginning from no training, and across random induced error levels from 0% to 50%. On this plot we see a reciprocal

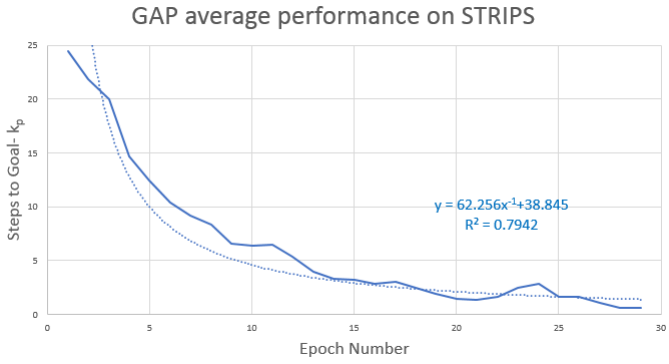

Figure 8: Average learning curve over instances of varying error from 0% to 50% in the STRIPS problem space, with the reciprocal fit curve plotted superimposed

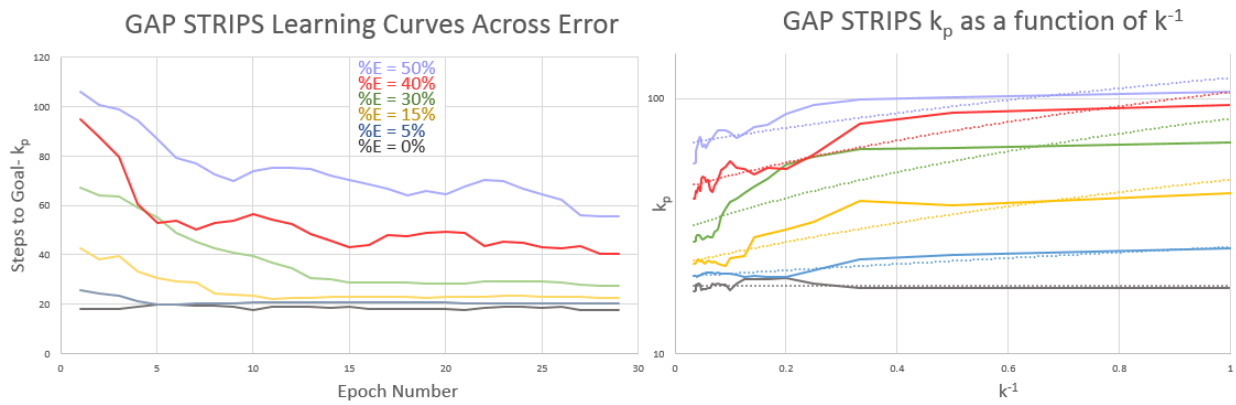

Figure 9: Learning curves for the STRIPS problem across levels of induced error from 0% to 50%, along with plots of k_p as a function of $\frac{1}{k}$ for various error levels, along with measures of the deviation from linearity in terms of % off–linear behavior, showing close correspondence to the predicted learning curve form of $k_p = A\frac{1}{k} + B$

fit curve of $k_p(k) = 62.3\frac{1}{k} + 38.9$ at $R^2 = 0.79$, and asymptotic learned performance approximately 39 steps between the starting state and the goal, compared to the no error absolute minima of 17.

FIGURE 9 showcases the learning at each error level, averaged over 50 trials. Learning tends to follow the same reciprocal pattern as the general curve, with variance in asymptotic performance due to the increase in error rate. To reinforce the reciprocal relationship, we plot linearizations and the off–linear percent labeled in TABLE I. The deviation from linear fit is relatively low, with the greatest deviation being for the 15% curve. The difference between asymptotic k_p and the fit functions ranging from 0.87% to 7.12% for induced errors up to 40%, and the difference between the the measured and predicted L_{max} is 7.8%, indicating very close correspondence between the observed performance and the predictions of Equations 18 and 12.

Of note is the 50% error case, for which the discrepancy is roughly twice other cases. Referring to Equation 16, we can see that as randomness grows, the difference between $k_{p\alpha}(k)$ and $k_{p\alpha}(0)$ shrinks: $\lim_{k\to\infty} k_{p\alpha}(k) \rightarrow L_{max}$, and so the function $k_{p\alpha}(k)$ approaches a constant function. Qualitatively speaking, as the induced error rate increases, P_α behaves more like a random process than the underlying 'true system' P_g.

In addition to the measures for the GAP algorithm's performance, we also have the comparisons to the QL and MDP agents along the same error rate panel. By contrast to the 10 to 15 epoch convergence of the GAP algorithm, the QL agents converge after 18 to 29 epochs. In addition, the convergent solutions for the 0% to 40% error rates are 20 to 10 steps slower, though the 50% case finds the QL agent solving the problem 6 steps quicker. Together, these illustrate that the GAP algorithm converges more quickly, and to higher quality solutions, than the QL agent. The MDP agent provides a baseline for performance, with the GAP agent consistently identifying solutions within 30% of the MDP-identified optima, highlighting that efficacious learning is occurring.

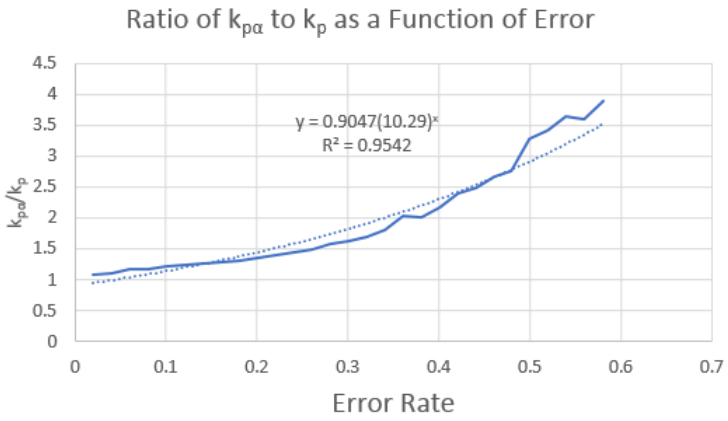

Figure 10: Plot of $\frac{k_{p\alpha}}{k_p}$ ratio for the learning curves of the GAP algorithm on the STRIPS world, along with the power law fit curve

We can use this data to confirm the predicted relationship suggested in Equations 13 and 16. Taking k_p to be the convergent performance of the 0% error case (based on the theoretical minimum case of 17 steps), we examine the ratio of $k_{p\alpha}$ at multiple error levels to this baseline k_p. Plotting $\frac{k_{p\alpha}}{k_p}$ in Figure 10, we find a high correlation exponential fit, validating the predicted relationship.

3.3 Maze/TAXI Domain

The TAXI and Maze problems are canonical study cases for machine learning systems. In the TAXI problem, the agent must visit a list of locations, pick up a 'passenger', and then deliver each passenger to a specific destination cell, and we complicate the problem by performing the navigation component inside a maze. Combining the two problems creates a complex hierarchical problem of similar character to the STRIPS implementation, but with substantially larger state spaces and far more complex learning patterns. Actions are movements between cells, and pickup and drop off actions. Inputs to the system vary depending on the abstraction mechanisms being employed,

Table 1: Comparison of measured and predicted values for analysis, calculated from the performance on the STRIPS problem learning curves, and comparisons to QL and MDPs

P_{thresh}	k_p I	k_p II	%E			Ak^{-1}	%OL	QL k_p	QL Ep.	MDP k_p
0%	18.53	18.10	2.30%	L_{max}I	25.30	54.9	7.4%	38.5	18	17.0
5%	20.04	20.21	0.87%	L_{max}II	27.29	62.1	6.8%	42.1	17	20.2
15%	22.81	22.76	2.11%	%E	7.8%	53.5	14.9%	43.5	19	20.1
30%	30.15	28.14	7.12%			25.9	5.9%	49.9	21	23.8
40%	43.82	42.07	4.15%			6.2	1.7%	53.4	29	59.2
50%	65.81	57.40	14.65%			0.08	2.9%	59.4	27	68.6

but varyingly include local observations of the maze topography, relative position of the target 'passenger', and whether a passenger is currently carried.

To engage with the goal-agnosticism of the GAP agent, we do not perform training on fixed TAXI destinations and mazes, but rather generate a random maze for each training epoch, complete with random target locations for 'pickup' and 'drop off' actions. Rather than restricting ourselves to simple mazes without interior spaces, we have allowed for the inclusion of open space regions in the maze. Such a maze is illustrated in FIGURE 11. Mazes with uniform width traversals are amenable to simple navigation algorithms, indicating inherent state space simplification, which we remove to increase problem complexity, and sensitivity of our experiments to impacts of varying error and abstraction.

In these experiments, only local, relative states are constructed- we do not use global coordinates, and the maze and goals are randomized every epoch, pressing the agent to learn the problem in a more general sense. In addition to illustrating the flexibility of the GAP, we also examine properties of learning transference and generalization. The maximal state space size is variable, however for the maze generation parameters used averaged to $|\mathcal{S}|_{max} = 18432$.

FIGURE 12, shows the average learning curves for the Maze/TAXI problem, for error ranging from 0% to 30%, with an R^2 of 0.84, greater than the STRIPS trials. A factor effecting fit quality is outlier learning cases, with the disturbances visible on FIGURE 12 as the sharp jump over epochs 4 to 7. These are due to randomization of the maze, where on occasion radically novel structures challenge the agent with an expanded problem space. The narrowness of the discrepancy indicates that the GAP algorithm is able to learn the more expansive problem after a few epochs. Further, that the scale of the disturbance is lower than initial performance implies a measure of learning transference occurs.

FIGURE 13, plots learning curves across induced error ranging from 5% to 30%, and the relationship between $k_{p\alpha}$ and $\frac{1}{k}$, also in FIGURE 13, to show continued adherence to predicted form. Asymptotic k_p proportional to error rate, and correlation between initial performance and long term performance are both present, as are further 'adaptation bumps' between epochs 4 and 8. The consistency of this range suggests that encountering a variant maze which causes innovative learning tends to happen, on average, three to four epochs after the initial learning. This range varies,

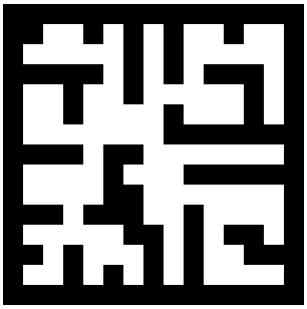

Figure 11: An example of a randomly generated ill–conditioned maze used in these Maze/TAXI problems

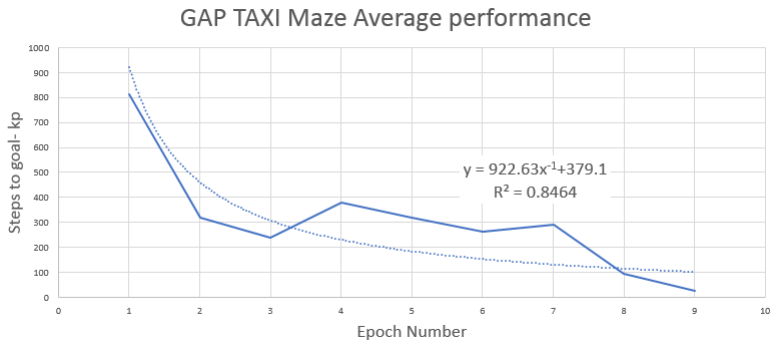

Figure 12: Average learning curve for the GAP algorithm operating on the complex Maze/TAXI problem, for all tested cases.

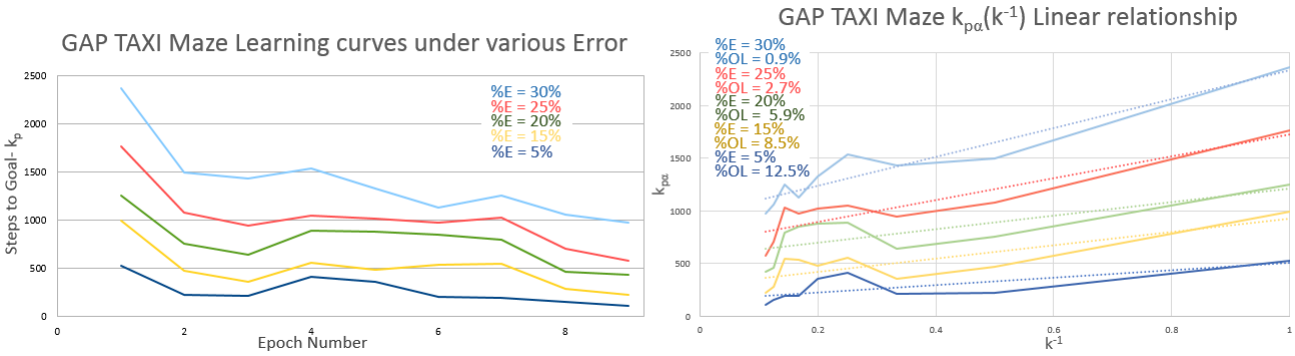

Figure 13: Performance of the GAP algorithm across multiple levels of induced error on the Maze/TAXI problem space and plot of k_p as a function of $\frac{1}{k}$, and the measures of non-linear divergence for the Maze/TAXI trials at each induced error level.

with some instances experiencing multiple small bumps, and others presenting with one substantial spike to nearly the initial performance level, followed by an on–model return to reciprocal behavior. However, long run trials (extending to 100 epochs) showed that the average case over each error level achieved asymptotic performance by 9 epochs, with no statistical outlier cases of occurring after 17 epochs across 1000 instances of training.

We also investigate the impact of abstractions on the GAP algorithm, using L_{max} for a group of abstractions used *in combination*. Three different kinds of abstractions are selected: AI constructs a vector representing the 8 neighborhood cells to the agent's current cell; AII is similar to AI, but includes only the 4 cells above, below, and to the sides of the current cell; and wA, or 'with Action', adds the additional information of the most recent action the agent has taken to the full state vector, inspired by the colloquial 'right hand rule' for naive maze navigation. These produce four different state generation methods: 'AI wA', using AI and wA together, 'AII wA', and AI and AII both without wA (nominally 'AI w/oA' and 'AII w/oA'). By composing the abstractions, we can compare the relative impact of each in accordance with the form of Equation 14.

On TABLE 2, the measured and predicted k_p for each joint abstraction are listed, with the average percent error. 'AI wA', 'AII wA', and 'AII w/oA' have relatively low errors at 4-8%, with 'AI w/oA' being an outlier at 14.5% error and k_p substantially higher than the other cases. TABLE 3 presents the values for L_{max}, and reciprocal fit characteristics. We find that the pairs of values are within the scale of correspondence observed as typical for the GAP algorithm thus far, and on the appropriate scale for the performance values observed in TABLE 2.

TABLE 3 also presents the comparison parameters for the QL and MDP learning instances. For both 'wA' cases and the 'AII w/oA', the MDP algorithm finds solutions in the 20-30 step range, and the GAP algorithm solutions occupying the same range. However, in the outlier case, 'AI w/oA', the average GAP algorithm performs 33% less effectively than the MDP system. The error between k_p is twice that of the next greatest discrepancy, and both L_{max} measures are greater than either- suggesting that the variance is much higher than for the other cases. In both the 'AII wA' and 'AI w/oA' cases, the QL agents fail to converge within the time limit. In the cases where the QL agent does converge, it takes about 15 times longer, and the identified solutions are 4.2 to 5.8 times less efficient than GAP.

FIGURE 14 illustrates the linearized performance curves across the four joint abstractions, with their off–linear errors, of which all are less than 10%. We can also observe the sharp chance in performance, the 13-fold reduction in performance for 'AI w/oA' highlighting the sensitivity in Equation 16.

These performance changes can be examined empirically: P_α can be constructed as a product of transform arrays, with effects determined via Equation 14, thus fit functions for $k_{p\alpha}$ as a function of $log(1 - P_{thresh})$ can estimate $||\alpha^+_{Ts}||_1 \cdot ||T_{\alpha k}||_1 \cdot ||\alpha_{Ts}||_1$. On FIGURE 15, we have plotted $k_{p\alpha}$ as a function of $log(1 - P_{thresh})$, to establish that Equation 14 has the correct linear form for this analysis. Using the linear fit for these curves, we derive the values in TABLE 4: estimated L1 norms (denoted $|\alpha^+ T\alpha|$ for compactness) at each error level. We see narrow statistical variance, expected given that the abstraction matrices are constant.

Table 2: Predicted $k_{p\alpha}$ versus measured k_p across error and abstraction for Maze/TAXI

P_{Thresh}		AI wA	AII wA	AI w/oA	AII w/oA
1%	Meas:	30.19	23.40	396.17	38.00
	Pred:	28.27	22.42	452.29	37.19
5%	Meas:	54.16	24.58	272.13	30.88
	Pred:	54.71	24.86	231.61	29.66
10%	Meas:	52.49	31.72	285.07	37.76
	Pred:	53.39	31.69	234.67	39.68
15%	Meas:	67.30	35.43	418.00	35.83
	Pred:	71.19	37.33	492.38	40.62
20%	Meas:	42.12	31.66	729.29	36.64
	Pred:	40.83	34.89	788.94	39.97
25%	Meas:	58.45	29.70	464.91	51.31
	Pred:	54.78	30.39	399.60	58.65
Avg. %E:		4.03%	3.88%	14.45%	7.98%

Table 3: Comparison of measured and predicted L_{max} across abstractions for the complex Maze/TAXI domain with joint abstractions, along with QL and MDP performance baselines.

	k_pI	k_pII	%E	Ak^{-1}	%OL	L_{max}I	L_{max}II	%E	QL k_p	QL Ep.	MDP k_p
AI wA	30.2	28.3	6.7%	69.9	9.2%	61.7	69.1	10.6%	127.3	265	30.4
AII wA	23.4	22.4	4.5%	8.1	6.3%	32.2	36.8	12.5%	NC	NC	23.5
AI w/oA	396.0	452	12.4%	505	7.2%	527.1	573.1	8.0%	NC	NC	297
AII w/oA	38.0	37.2	2.2%	24.9	8.4%	40.4	43.8	8.3%	221.4	269	33.5

The similarity of the L1 norms indicate successful estimation of this parameter, but for the full mapping which may contain other factors aside from the abstractions (such as implicit bias). However, because we created joint abstractions from composition, we estimate the impact across paired subsets via ratios of the pairs' measures, relying on the submultiplicity of the L1 norm. On TABLE 5, these ratios are listed, with each variant transition presenting similar scale changes. To judge the scale of these deviations, we can combine the two transitions which transfer 'AI wA' to 'AII w/oA' and compare these to the actual proportional difference between 'AI wA' and 'AII w/oA'. Doing so, we find the estimates to be 0.958 and 0.791; respectively 10% and 9% off of the actual ratio of 0.871 confirming the analysis in Section 2.3.6.

3.4 Tower of Hanoi Domain

The Tower of Hanoi puzzle is a perennial favorite for mathematical analysis. The puzzle consists of a number of disks stacked on three or more pegs, labeled by an ordinal index which must be preserved when disks are transferred between pegs.

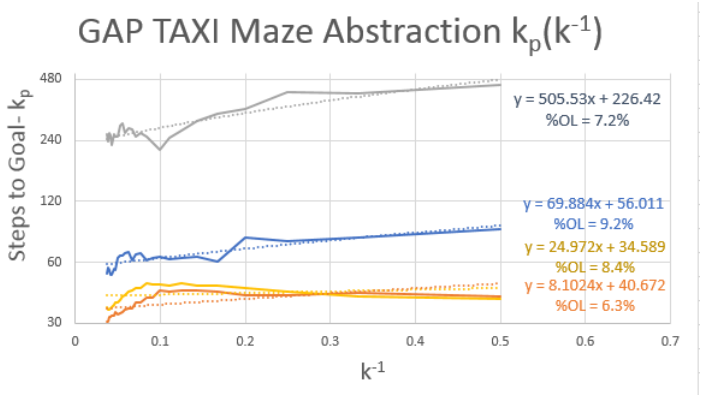

Figure 14: Linearized plot of the learning curves for the Maze/TAXI learning under abstraction with corresponding measures of off–linear performance for each of the four combinations of abstractions implemented

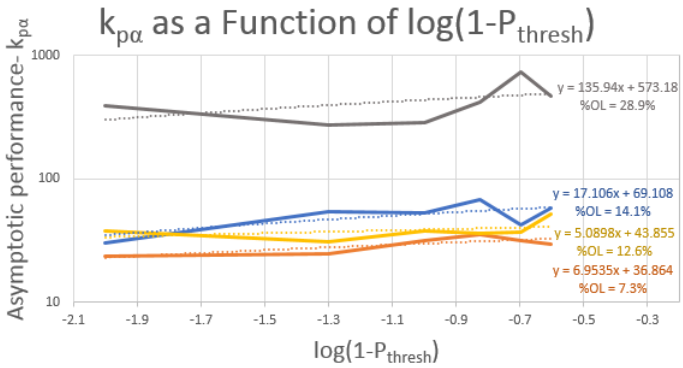

Figure 15: Convergent behavior of the four abstractions as a function of $log(1 - P_{thresh})$, as in Eq. 14, along with the percent off–linear deviations for each curve, corroborating the use of Equation 14 as a proxy for calculating the remaining components of $|\alpha^{+}T\alpha|$

Table 4: Measured $k_{p\alpha}$ and corresponding $|\alpha^{+}T\alpha|$ estimates

	AI		AII	
P_{Thresh}	$k_{p\alpha}$	$\|\alpha^{+}T\alpha\|$	$k_{p\alpha}$	$\|\alpha^{+}T\alpha\|$
1%	30.19	1.05	23.40	1.15
5%	54.16	1.09	24.58	1.11
10%	52.49	1.06	31.72	1.21
15%	67.30	1.57	35.43	1.77
20%	42.12	1.02	31.66	1.14
25%	58.45	1.05	29.70	1.08
wA	1.144 (±12.5%)		1.249 (±14.1%)	
1%	396.17	1.01	38.00	0.99
5%	272.13	1.00	30.88	0.99
10%	285.07	1.00	37.76	1.00
15%	418.00	1.01	35.83	0.99
20%	729.29	0.99	36.64	1.00
25%	464.91	1.01	51.31	0.00
w/oA	1.004 (±0.3%)		0.996 (±0.2%)	

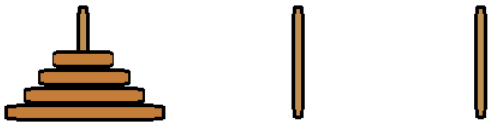

Figure 16: Illustration of a traditional Tower of Hanoi (ToH) problem: the objective is to move all disks from the first peg to the third, by only moving disks between pegs, and under the constraint that a disk may only be moved on top of a larger disk or to an empty peg. This graphic shows the 3-peg, 4-disk, variant of the problem, $ToH_{3,4}$

Table 5: Calculated $|\alpha^{+}\alpha|$ ratios across abstractions and predicted transform measure, derived from the entries in TABLE 6 and Equation 14

Q(α)	AI	AII	$I \rightarrow II$		AIwA→AIIw/oA
wA	1.144	1.249	1.091	Meas:	0.871
w/oA	1.004	0.996	0.992	Pred 1:	0.958 (+10%)
wA → w/oA	0.877	0.798		Pred 2:	0.791 (-9%)

The canonical implementation has 3 pegs, and 3 to 7 disks, but both can be changed. Problems are represented as $ToH_{p,d}$, where p is the number of pegs, and d is the number of disks. As a well-defined problem, with the mathematics of the 3-peg case being particularly well studied, the Tower of Hanoi presents opportunities for direct performance comparisons, with greater numbers of pegs presenting wider state spaces with lower net complexity, and increased numbers of disks representing increases in complexity. The scope of the state space is $|\mathcal{S}|_{max} = p^{d}$, and the action space $|\mathcal{A}|_{max} = p^{2}$, though only a small subset is reachable, and graph complexity is lower than the upper bound.

FIGURE 17 plots the average learning curves for $ToH_{3,3}$, $ToH_{3,5}$, and $ToH_{4,5}$ over error rates ranging from 0% to 25%, and the reciprocal best fit curves for each. As with the prior two domains, we see close fit to the reciprocal form. TABLE 7 presents the results from these tests in tabular format. Here, we can see steady deviations from the near optimal performance of the $ToH_{3,3}$ and $ToH_{3,5}$ cases as error level increases.

We also include the QL and MDP performance levels on TABLE 7, across each version and error level. As with the prior experiments, we observe the QL agents converging at much slower rates, up to and including non-convergent learning which increases in frequency as the problem complexity class and error rates increase (a phenomenon we explore in more detail shortly). Additionally, we can confirm continued efficacy of the produced solutions, with the GAP plans approaching the

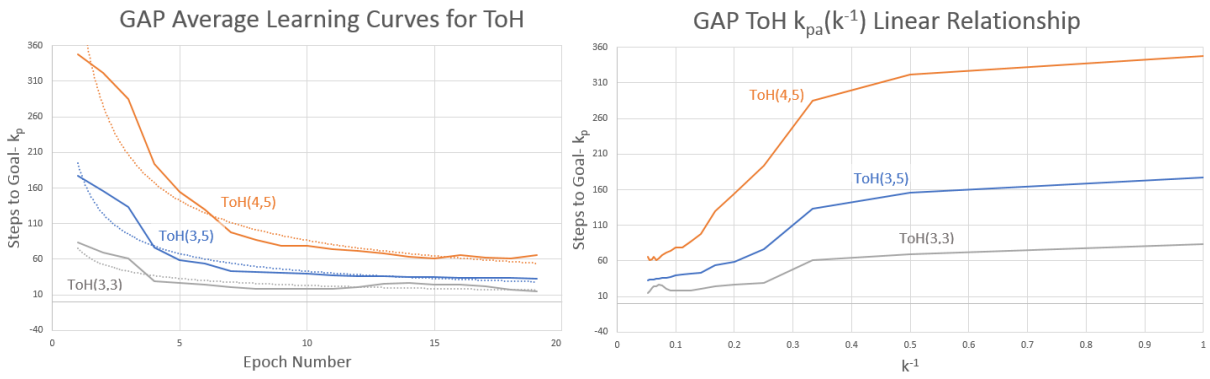

Figure 17: Average learning curves for the GAP algorithm over the three investigated ToH domains, $ToH_{3,3}$, $ToH_{3,5}$, and $ToH_{4,5}$ at varying error levels, along with reciprocal fit curves

Table 6: Chart of the correlation measures for the GAP Algorithm learning the Tower of Hanoi problem, across error level and problem complexity class

	P_{thresh}	k_pI	k_pII	%E	Ak^{-1}	%OL	QL k_p	QL Ep.	MDP k_p
ToH(3,3)	5%	16.5	15.6	5.5%	110	19.3%	19.2	141	11.4
	15%	47.3	47.8	0.9%	117	7.8%	36.7	138	35.2
	20%	61.9	63.4	2.4%	126	1.8%	NC	NC	39.9
ToH(3,5)	5%	32.7	30.1	11.4%	35	5.4%	37.2	95	30.8
	15%	34.8	37.1	6.4%	144	5.8%	100.2	137	34.3
	20%	44.9	48.7	8.2%	388	16.1%	NC	NC	51.2
ToH(4,5)	5%	206.4	201.5	2.4%	1678	12.3%	287.4	135	175.4
	15%	696.7	707.5	1.6%	1336	2.3%	NC	NC	599.1
	20%	2052.2	2006.9	3.0%	133	1.2%	NC	NC	1766.6

performance levels indicated in the MDP planner, rising also to a maximum 16% deviation at peak complexity and error rate.

TABLE 7 shows that $ToH_{4,5}$ levels deviate substantially with error, suggesting that the expanded state space is more vulnerable to impacts of variance (a property we will interrogate further shortly). We also present the errors associated with the reciprocal fit curves, showing generally strong fits, excepting the outlier of the $ToH_{3,3}$ case at 5% error. However, the low error between k_p and $k_{p\alpha}$ suggests that this is likely due to rapid convergence, and indeed the $ToH_{3,3}$ case converges at approximately 5 epochs rather than the 20 sampled. Algorithmic solutions are available from [20], enabling comparisons to the theoretical optima. For the 3 peg cases, the optimal number of moves is $2^d - 1$, giving 7 moves for 3 disks and 31 moves for 5 disks. The corresponding agents take 15 and 33 steps respectively across the full error range, with the 0% error cases naturally achieving the optimal performance level after one epoch.

We can also use the substantial state space and restricted set of elements to we develop and implement four abstractions: AI- Direct conversion of lists of disks on each peg to a numerical state: the sum of products of disk indices on each peg; AII- Encoding of disk placement as a list of the sums of disk indices on each peg; AIII- Listing pairs of the number of disks currently stacked on each peg and the index of the topmost disk; AIV- Listing the number of disks on each peg. Each produces incrementally compressed state spaces.

TABLE 7 presents the measurements for the battery of experiments on all four abstractions across error rates from 0 to 20%. The AIII and AIV cases for the $ToH_{3,5}$ case unilaterally converge to the optimal number of steps, precluding analytical estimation of L_{max}. This is remarkable as this convergence is not strictly seen in the simpler $ToH_{3,3}$ case. Further, the highest performing abstractions vary among the problem classes; for $ToH_{3,3}$ AI and AIV perform most strongly; AIII and AIV for $ToH_{3,5}$; and finally AI and AII for $ToH_{4,5}$.

TABLE 7 also lists comparison cases alongside the measurements for the GAP. As before, convergent performance levels approach those for the MDP based agents, contingent on problem complexity and abstraction density. The QL agent consistently fails to learn solutions for the AIII and AIV abstractions, though the AI and AII versions perform relatively close to the GAP and MDP agents,

Table 7: k_p and L_{max} comparisons for the GAP algorithm learning the ToH problem with various abstractions and across complexity classes

	Abst.	k_pI	k_pII	%E	$L_{max}I$	L_{max}II	%E	QL k_p	QL Ep.	MDP k_p
$ToH_{3,3}$	AI	16.5	15.6	5.6%	15.6	17.5	11.4%	27.2	139	17.1
	AII	27.8	31.5	13.5%	8.1	8.8	7.3%	37.1	123	26.9
	AIII	21.6	17.3	20.2%	17.22	14.8	16.3%	N/A	N/A	20.0
	AIV	17.0	15.3	9.7%	31.5	35.1	10.3%	N/A	N/A	12.9
$ToH_{3,5}$	AI	35.3	34.2	3.1%	62.1	59.4	4.7%	41.8	115	32.6
	AII	31.4	35.1	11.8%	64.9	69.0	5.9%	36.1	163	37.5
	AIII	31.0	31.0	0%	31	31	0%	N/A	N/A	31.8
	AIV	31.0	31.0	0%	31	31	0%	N/A	N/A	31.1
$ToH_{4,5}$	AI	254.5	256.9	0.9%	112.06	104.5	6.8%	226.9	72	162.9
	AII	278.1	241.3	13.2%	101.9	112.6	10.5%	4068	43	273.6
	AIII	1267.2	1354.6	6.9%	391.9	371.5	5.2%	N/A	N/A	1076
	AIV	2017.7	1843.2	8.6%	719.9	749.8	4.1%	N/A	N/A	1899

albeit taking 11 to 16 times more epochs to converge. In the $ToH_{4,5}$ QL experiments, convergence times are approximately a half of those for the other problems, and the AI solution is marginally superior to the GAP algorthim, but for the AII case, a relatively brief 43 epoch converges to a solution 14 times less efficient than the GAP and MDP solutions. Additionally, on TABLE 8 are presented the metrics for the reciprocal fit of each trial set, curves matching the predicted form with errors on the order of 2-8%.

There is a telling comparison in the $ToH_{4,5}$ cases for AIII and AIV, the convergent k_p are substantially larger than those for the AI and AII. Further, the average, no–abstraction $ToH_{4,5}$ curve converges at $k_p = 56$, indicating the GAP can learn this problem. Because the trials on $ToH_{4,5}$ perform substantially worse than the non-abstracted case, and other classes perform similarly, we hypothesize unsuitability of the AI-AIV models to represent the $ToH_{4,5}$ problem space, specifically. Considering AI, increasing the number of pegs, the number of abstracted states remains nearly constant, yet real states increase exponentially. We can consider this ratio of abstracted to real states as a metric component which puts an upper bound $Q(\alpha_T)$. Similar complications thus exist at even more substantial levels for the other abstractions, with more severe impacts due to the greater reduction in the size of the abstracted state space.

Table 8: Curve fit metrics for $k_{p\alpha} = \frac{A}{k} + k_p$ in the ToH trials, across Abstractions I-IV, with % off–linear measures for each best fit line.

	AI		AII		AII		AIV	
	Ak^{-1}	% OL	Ak^{-1}	% OL	Ak^{-1}	% OL	Ak^{-1}	% OL
$ToH_{3,3}$	130.9	4.1%	84.9	6.1%	116.9	8.0%	131.6	6.7%
$ToH_{3,5}$	144.1	1.6%	195.6	3.1%	N/A	-%	N/A	-%
$ToH_{4,5}$	721.5	8.2%	439.7	6.5%	3886.6	7.5%	989.4	1.6%

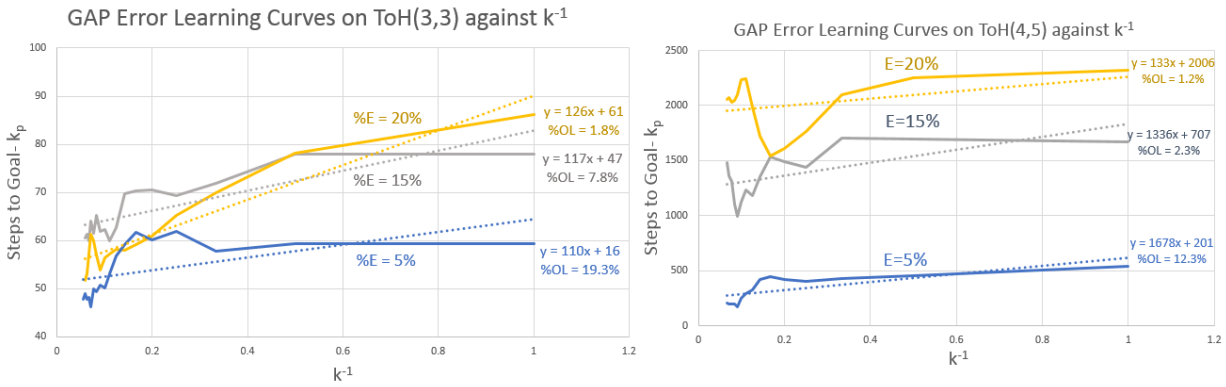

Figure 18: Plots of the linearized GAP learning curves for the $ToH_{3,3}$ (left) and $ToH_{4,5}$ (right) problems across multiple levels of induced error and associated off–linear error measures

This effect can be examined using error as a proxy for performance reduction. In FIGURE 18 (left) are linearized curves for $ToH_{3,3}$ on error rates 5, 15, and 20%. The k_p congregate around the 5% and 15% error levels, and with a ratio of 3.8x between the 20% and 5% k_p. In FIGURE 18 (right), by contrast, are the error impacts on the $ToH_{4,5}$ case, where there is a substantially higher susceptibility to error, with proportional change between 5% and 20% being a factor of 10x, and the AI and AII cases, with k_p around 260, are near to the 5% error level, with the AIV case approaching the 20% error level. This shows how relative size of an abstracted state space can impact performance, with similar results to changing error level, implying a direct relationship between the ratio of sizes of the native and abstracted state spaces that has a substantial limiting impact on the overall performance of the GAP algorithm [4].

Combined with validation of the composition model in Section 3.3, we have a learning system in which each element obscuring a problem- perturbations, error, abstractions, and inaccuracy, can be modeled as probabilistic transforms for which an information budget is present, and the quality of the resulting solutions is a result of the total cost of those impediments, as indicated in Equation 15, . This allows the GAP agent to address one of the principle limitations discussed by symmetry breaking- in a grounded way, tying similarities between performance under uncertainty and abstraction together.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a hypergraph–based learning and planning algorithm, the Goal Agnostic Planner, designed for solving hierarchical planning problems without the need to construct a reward function or world model and the capacity to plan between any pair of states. This algorithm uses a 3-dimensional array modeling a hypergraph data structure, augmented by two 2-dimensional composite data structures comprised of arrays containing ordered linked lists. These data structures retain information observed information for planning. We claimed that this structure and the associated algorithms possessed several benefits as a joint system: optimal solutions, exponential goal

convergence and bounded failure rates, tolerant of abstracted and uncertain model perturbations, and follows a reciprocal learning curve.

Planning uses 2-dimensional slices of the hypergraph, a space complexity bounded by $O(n^3)$. We proved that these slices contained the path with the greatest joint probability between any pair of states embedded in the hypergraph. We also developed, to accompany this data structure, an in–situ maintenance algorithm operating in $O(1)$, a sequence inference algorithm based on Dijkstra's algorithm, and proved that this algorithm extracts the greatest probability path in the hypergraph between any pair of states in $O(n^2)$ via the maximal probability subgraph. This information was used to construct a dynamic performance model from the predetermined policy from the planning algorithm, predicting the time evolution, showing probability of goal achievement was monotonic, describing the probability that the agent becomes unable to transition to the goal, that the convergence rate is bounded above by an exponential function, and illustrating that the transition to the terminal states is efficient.

Based on these results, we introduces a model for perturbations and abstractions, and derive the conditions under which a path planned in those spaces will converge in the true space. We analyzed the impact of the abstractions on system performance, and a metric for the 'quality' thereof. A specialized one-to-one transform model for incremental learning allowed us to show that GAP agents will demonstrate progressive learning, and determine that learning curves for GAP agents will, on average, follow a reciprocal trend.

To investigate the performance of the GAP algorithm on actual problem cases, we performed trials on three problem cases in Section 3: a traditional STRIPS problems, a combined maze navigation and the TAXI domain, and the Tower of Hanoi puzzle. In each case, we examined the performance of the GAP algorithm during learning, demonstrating that the predicted reciprocal form of the convergent learning curve persists throughout all experiments, and grounded the effectiveness of GAP agains MDP and QL systems. To validate our analysis, we made proxy measurements derived from the best fit learning curves to compare to our predictions, and applied varying levels of artificial error in each experiment to study the impact of disturbances. Convergent behavior was shown in the face of these factors. We also used the increased complexity and larger state spaces of the Maze/TAXI and Tower of Hanoi problems to investigate the effects of various abstractions on learning performance. We also used individual experiments to explore specific properties: the STRIPS problem for the power law relationships predicted for convergent performance under varying uncertainty; the Maze/TAXI experiments, to confirm theoretical predictions about performance under composed transforms; and the Tower of Hanoi experiments to investigate the properties of the abstraction quality metric.

4.1 Limitations

The size of the INC array, which scales as $O(|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|)$, is a pragmatic limit. For instance, in a hierarchical problem with two divisions, we can represent each subproblem space by a vector. If each vector contains three binary members, then the total size of the state space will be 64, and thus the state/state space will be of size 4096 elements, multiplied by however many actions are present, highlighting the ease with which the hypergraph can become very large. One way we might look at this is as an extension of problem (2) from [21], wherein addressing the issue of combinatorial

explosion of paths through a state space, we have shifted the computational burden from the time domain to the space domain. Abstractions naturally provide a means to reduce the size of the INC array, as well as compression in a stable, well-defined manner. Unlike the traditional 'curse of dimensionality', sparsity can be leveraged for improved performance. Additionally, alternative methods of representing INC which can reduce the size of the array are discussed in Appendices A.3 and A.4.

We have used a naive implementation of Dijkstra's algorithm which has worst–case time order of $O(|\mathcal{S}|^2)$. While this is not especially bad, more advanced, faster path finding algorithms can be applied to improve overall performance. The maintenance of subgraphs is independent of path–planning, so any algorithm compatible with optimization of path costs on a graph would be viable. In Appendix A.4, a model for integration with A* is presented as a means for adapting the GAP algorithm to use a faster planning approach.

The algorithm has phase associated with random actions for exploration of the state space. One may even deliberately program a phase of execution with entirely random actions to save computation time until enough paths are known. Approaches to examining the impact of, and amelioration for, this phenomena are discussed in Appendices A.1, A.3, and A.4. A method we explored in pilot experiments is the use of Tabu search in the exploration phase. We implemented a policy by which selection of actions was from only previously unused actions in each state. One very particular address to this problem, which presents the possibility of a powerful augmentation to both learning speed, and learning transference, is investigated in Appendix A.3- the possibility of using structural and archetypal patterns in problems to pre-train systems which can then more quickly be tuned to solve a more specific iteration.

4.2 Further Work

Though we developed a means to detect and quantify the risk associated with dead–ends (trap nets and non–goal attractors), none of the problem cases in this paper contain any such networks. Similarly, though the concept of multiple attractor states is discussed, all systems demonstrated thus far possess singular goal states. We have explored non-deterministic action of the GAP algorithm vis-a-vis error induction, and abstractions, but none of the problem cases presented in the set of validation experiments above is *inherently* non-deterministic. A future body of work would include these problem types.

While out of scope for the matter of this paper, it is simple to conceptualize a model in which the goal is not expressly a singular state, or set of states, but rather a metric function of some kind. Though this does re-introduce some issues associated with the use of bespoke objective functions, the potential for learning cost improvements opens the possibility of process optimization. A specific case of this would be an expected-cost formulation of the GAP algorithm, minimizing weighted cost rather than maximizing probability- addressed in Appendix A.2.

A further complex issue is learning transference. In the randomized worlds used in the Maze/TAXI experiments, some evidence of transferred learning was presented. However, we have analyzed transferred performance beyond a special case of the general reciprocal curve. This condition was alluded with the 'adaptation bumps' observed in Maze/TAXI learning. The value of a model for

transference learning beyond this is clear from the observed consistency in the rate of appearance of these outlier bumps.

Perhaps the most interesting direction is related to transference and the application of patterns observable in the INC and AFI arrays of trained agents. These patterns are discussed in appendices A.2, A.3, and A.4. Our interest is in leveraging the possibility of archetypal problem classes, and implementing statistically derived models representing those agents, to pre-train GAP agents, and allow for inference between joint, complex, or synthesized problems. For instance, one might prepare a statistical model of a a problem which can be scaled (for instance the Tower of Hanoi's disks, or the size of a maze) and generate a baseline AFI array for an up-scaled problem. Or, more esoterically, it may be possible to develop structural representations of how hierarchical domains are combined, as well as representative models of the subproblems, and define a new brain by applying those composition rules to the subproblems to generate a fitting AFI.

Another direction to investigate along these lines is the utility in identifying problems as joint domains. Considering the Maze/TAXI domain- is it possible that one could train an agent on the TAXI problems and Maze problems independently, develop a statistical model of each, and generate an efficient pre-train model by combining the two? Such methods, if effective, would provide an extraordinary amount of design power, inference potential, and learning transference value to GAP agents. Perhaps even the use of other unsupervised learning methods on a body of pre-existing AFI templates could be used to generate AFI arrays directly from descriptions or initial observed data. We find these directions, while admittedly ambitious, to be worthy of ongoing investigation.

References

[1] Matignon L, Laurent GJ, Le Fort-Piat N. Reward Function and Initial Values: Better Choices for Accelerated Goal-Directed Reinforcement Learning in International Conference on Artificial Neural Netw.2006:840-849.

[2] Koenig S, Simmons RG. The Effect of Representation and Knowledge on Goal-Directed Exploration With Reinforcement-Learning Algorithms Mach Learn. 1996;22:227-250.

[3] https://www.cs.kent.ac.uk/people/staff/mg483/documents/grzes17goals-in-pbrs.pdf

[4] Dimitrov NB, Morton DP. Combinatorial Design of a Stochastic Markov Decision Process in Operations Research and Cyber-Infrastructure. Springer; 2009:167-193.

[5] Szepesvári C, Littman ML. Generalized Markov Decision Processes: Dynamic-Programming and Reinforcement-Learning Algorithms in Proceedings of International Conference of Machine Learning. 1996;96.

[6] Steinmetz M, Hoffmann J, Buffet O. Goal Probability Analysis in Probabilistic Planning: Exploring and Enhancing the State of the Art. J Artif Intell Res. 2016;57:229-271.

[7] Bertsekas DP, Tsitsiklis JN. An Analysis of Stochastic Shortest Path Problems. Math Oper Res. 1991;16: 580-595.

[8] Guillot M, Stauffer G. The Stochastic Shortest Path Problem: A Polyhedral Combinatorics Perspective. Eur J Oper Res. 2020;285:148-158.

[9] Blum AL, Furst ML. Fast Planning Through Planning Graph Analysis. Artif Intell. 1997;90:281-300.

[10] Blum AL, Langford JC. Probabilistic Planning in the Graphplan Framework. In: Biundo, S., Fox, M. (eds) Recent Advances in AI Planning. ECP 1999. Lecture notes in computer science. 2000 Jan 1;1809:319-332.

[11] Hunter A, Thimm M. Probabilistic Reasoning With Abstract Argumentation Frameworks. J Artif Intell Res. 2017;59:565-611.

[12] Leonetti M, Iocchi L, Stone P. A Synthesis of Automated Planning and Reinforcement Learning for Efficient, Robust Decision-Making. Artif Intell. 2016;241:103-130.

[13] Hostetler J, Fern A, Dietterich T. Sample-Based Tree Search With Fixed and Adaptive State Abstractions. J Artif Intell Res. 2017;60:717-777.

[14] Pineda L, Zilberstein S. Probabilistic Planning With Reduced Models. J Artif Intell Res. 2019;65:271-306.

[15] Konidaris G, Kaelbling LP, Lozano-Perez T. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. J Artif Intell Res. 2018;61:215-289.

[16] Lüdtke S, Schröder M, Krüger F, Bader S, Kirste T. State-Space Abstractions for Probabilistic Inference: A Systematic Review. J Artif Intell Res. 2018;63:789-848.

[17] Dietterich T. State abstraction in MAXQ hierarchical reinforcement learning. Advances in Neural Information Processing Systems. 1999;12.

[18] McCallum RA. Reinforcement Learning. Adv Neural Inf Process Syst. 1995;7:377.

[19] Knoblock CA. Abstracting the tower of Hanoi in Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions. Citeseer. 1990: 4976: 1-11.

[20] Van Zanten AJ. The Complexity of an Optimal Algorithm for the Generalized Tower of Hanoi Problem. Int J Comput Math. 1990;36:1-8.

[21] Jiménez S, De La Rosa T, Fernández S, Fernández F, Borrajo D. A Review of Machine Learning for Automated Planning. Knowl Eng Rev. 2012;27:433-467.

Appendix A. GAP Algorithm Modifications

The structure of the GAP algorithm presents itself to a set of modifications which present various advantages. We have experimented with each of these in our test cases, but none in sufficient detail for an expansion of the prior analysis.

A.1 Implicit learning rate

The GAP algorithm does not incorporate an explicit learning rate parameter, but can be viewed as an averaging function over the number of samples. Consider the impact of one fluke observation at different times: say we have observed a transition nine times, such that: $\sum_{\forall k} INC[s_i, s_j, a_l] = 9$ and that all such observations have been precipitated with action a_1. If the tenth observation is precipitated by a_2; the associated probabilities then were previously: $P(a_1(s_i) \rightarrow s_j) = 1.0$ and $P(a_2(s_i) \rightarrow s_j) = 0.0$ whereas after, they are: $P(a_1(s_i) \rightarrow s_j) = 0.9$ and $P(a_2(s_i) \rightarrow s_j) = 0.1$, a relative shift of 10%. If the total observations, were 99 prior to the anomalous result, then the probability shift would be only 1%.

This illustrates how learning is tied to the reciprocal function, which includes learning inertia: early anomalous results take longer to correct. If learning is online this may bias the agent. An alternative is training with an artificial learning rate, implemented as a fixed moving average, where probabilities are calculated as proportions of a fixed window.

A.2 Alternative policy functions

Rather than uniformly selecting the most probable action choice, and agent may instead use a weighted expectation of each. For instance, a local probability: $P_{s_i,a_l}(s_f) = P(a_l|AFI)P_{a_l(s_i)\rightarrow s_f}$, where $P(a_l|AFI)$ represents the probability that a_l is chosen. We can write an alternative method of hypergraph compression based around the probability of each state transition: $P(s_i \rightarrow s_f) = \Sigma_{\forall a_l} P(a_l|AFI)P_{a_l(s_i)\rightarrow s_f}$ This equation compresses the hypergraph along the action slice by coupling all action results together with the $P(a_l|AFI)$ function. If we define:

$$P(a_l|AFI) = \begin{cases} 1 \; a_l = \underset{l}{\operatorname{argmax}} P(a_l(s_i) \rightarrow s_f) \\ 0 \; otherwise \end{cases}$$

Then the compression resolves to the maximally probable subgraph.

As an alternative, however, consider that we let

$$P(a_l|AFI) = \frac{P(a_l(s_i) \rightarrow s_f)}{\Sigma_{\forall a_l} P(s_f|s_i, a_l)}$$

The probability of taking action a_l is proportional to the relative likelihood of a_l resulting in s_f *relative to other actions*. Such a system will converge less aggressively, but an envelope function using Equation A.2 would enable selection of policy to favor exploitation or exploration as an explicit, bounded system property.

A potentially more powerful application of this concept would be to minimize the *expected cost* of a sequence rather than probability. One would combine two INC arrays- one for observed costs and one for probabilities, and compute minimizing paths by calculating the net expected cost at each step. Such a system would require a more sophisticated sorting system for the AFI array linked-lists, but even using an in-place $O(S^2)$ algorithm maintain the overall computational complexity of the agent.

A.3 In situ transfer functions

One alternative to learning a full state space for a problem would be to represent AFI as a transfer function. In this paradigm, some function $I(s_i, a_l)$ would either produce a statistical distribution over s_f, or $I(s_i, s_f)$ a distribution over a_l. Such a distribution might be, for instance, a rule which eliminates potential actions, such as a non-movement action only producing states which possess the same physical location as s_i. This function would then be called during the planning stage, specifying the associated probabilities by 'generating' entries in INC. For instance, in the Maze/TAXI problem case, one could define a simple function describing the effect of move actions based on the known local topology of the maze. While a manual function like this would re-introduce design bias, automated analysis could instead be used to derive $I(\cdot)$, tantamount to autonomous hierarchical decomposition via identification of an abstraction transform.

A concept discussed in the paper's conclusion is the use of extant models for AFI arrays, combined under principles derived from the analysis of joint and hierarchical problems, to generate pre-training AFI for GAP agents based on topical observations or analysis of novel domains. Use of weights or compositions could provide a mechanism for emulating generalized transfer functions for GAP agents to plan on. These potentialities all descend directly from the transfer function-like abstraction modeling described in sections 4.3 and 4.4 and validated throughout the main paper.

A.4 Generalized Heuristics and statistical models of AFI

Previously, mention was made of alternative planning algorithms for use within the Algorithm 2, such as A*. Defining generalized heuristics based on the structure of GAP, instead of a specific problem, is possible. We can illustrate this with an example, since states were assigned hashed numerical labels in the order of discovery of the state. Under random exploration, a structural relationship appears in AFI: adjacent states are most likely to be determined within relatively short time periods, so transition probabilities are clustered around the primary diagonal. FIGURE 19 demonstrates this exact phenomenon for one of the $ToH_{3,5}$ agents and one of the Maze/TAXI agents.

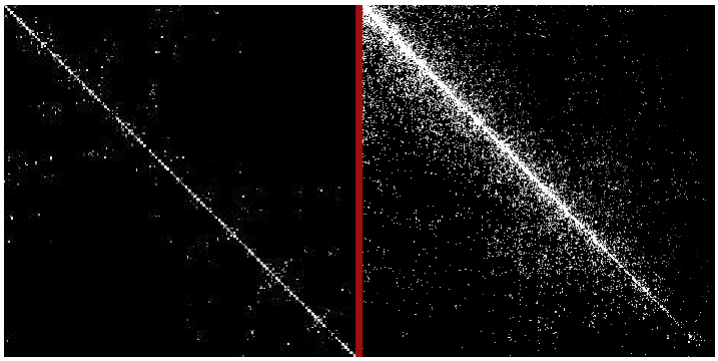

Figure 19: Visual representation of example AFI arrays for learning on the $ToH_{3,5}$ problem (left), and the Maze/TAXI problem (right) showing the relationship between structure in the AFI array and state discovery order

Based on this observation, say we build a rough statistical model, with a Gaussian distribution $s_i = s_f$, variance decearsing linearly from v_1 (measured empirically) as $s_i \rightarrow |\mathcal{S}|$. Use geometric inference to construct:

$$P_e(s_i, s_f) \approx \frac{1}{v_1\sqrt{2\pi}} \cdot \frac{2|\mathcal{S}|}{2|\mathcal{S}| - (s_f + s_i)} \cdot e^{\frac{-1}{v_1} \cdot \left(\frac{|\mathcal{S}|(s_f - s_i)}{2|\mathcal{S}| - (s_f + s_i)}\right)^2}$$

extrapolating probabilities for indirect transitions. With starting state being s_0, we can write an heuristic function $h(s_i)$ as $h(s_i) = P_e(s_i, s_g)$,

$$f(s_i) = P_e(s_i, s_g) \cdot \prod_{\forall j \in \sigma_{0,i}} P(s_j \rightarrow s_{j+1})$$

which would fit the bill for A*.

Not all problems will necessarily allow for this exact formulation, but it illustrates how a statistical model of AFI may be created and applied. It is also entirely possible to conceptualize a similar process of identifying a distribution over AFI to represent a problem *archetype*. This approach may provide a design mechanism to analytically reduce the state space size without introducing substantial compression loss, or with probabilistically bounded loss rates.

The presence of this representative structure is what inspires the previously discussed research tracks based around analyzing AFI arrays for the purposes of pre-training, inference, and synthesis. It may be possible to use derived models, such as the example above, as input to machine learning techniques to turn feature spaces for problems into AFI arrays, analysis of partially learned spaces for training acceleration, archetypal models for adaptive systems, and classification of problems by weighted decomposition of learned AFI. Moreover, it may be possible to use them as a knowledge base for descriptive and generative typologies which allow for knowledge transference, re-use, and synthesis in novel problems. All these methods would provide unique power for the GAP algorithm to be an effective agent for problem solving which requires quick learning, effective solutions, and the capacity to adapt and infer.