Goal Agnostic Learning and Planning without Reward Functions

Christopher Robinson CKEVINR@GMAIL.COM
Department of Electrical and Computer Engineering
University of Louisville
Louisville, KY 40292, USA

Abstract

In this paper, we present a novel learning system for solving arbitrary sequential, hierarchical planning problems under stochastic uncertainty and abstractions which requires no reward function to identify optimal policies. It implements a hypergraph–based learning model, a datastructure–driven maintenance method, and a planning algorithm based on a probabilistic application of Dijkstra's algorithm in a holistic approach to joint learning and planning. The resulting agent demonstrates analytically provable benefits, in particular combining aspects of Reinforcement Learning, Markov Decision Processes, and STRIPS-like automated planners. In this manuscript, we show that the algorithm determines optimal solutions, mathematically bound learning performance rates, and supply a mathematical model analyzing system state progression through time. This model produces explicit predictions for the form of learning curves, goal achievement rates, risk of reach dead–ends, and response to abstractions and uncertainty. To validate performance, we exhibit results from applying the agent to three archetypal planning problems, including composite hierarchical domains, and highlight empirical findings which further illustrate properties elucidated in the analysis.

1. Introduction

Traditionally, Artificial Intelligence (AI) and Machine Learning (ML) are conceptualized as solving somewhat different problems. Machine Learning generally identifies patterns in structural relationships which render problem solving straightforward. By contrast, AI leverages relationships between states and attempts to apply logical operations to convert pattern–weak representations into solutions. This is a very broad generalization, but in terms of trends algorithms in these two fields often conform to these patterns. In this paper, we will be looking particularly at the issue of sequential problem solving, the process of identifying a sequence of actions to progress a system from some initial state to a goal state, a common target for wide classes of AI and ML both.

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 signifier 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 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. The common thread of large ecosystems of adapted models suggests that both AI and ML have similar requirements in design which drive the adoption of specific frameworks and later adaptation thereof. 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.

With this paradigm the sequential planning problem, in practice, demonstrates many systematic solutions with similar design requirements and similar outcomes: constructing a problem solving agent requires designing some model to drive convergence to a solution, and the result is a wide diversity of applied models in the literature. Consequently, the models presenting the most generalized design utility (STRIPS-like, Reinforcement Learning, MDPs, PDDL, and HTNs as specific examples we highlight shortly) succeeding by virtue of their adaptability to the divergent requirements of practical problems.

However, there are still observable weaknesses present in all these algorithms which present an opportunity for improvement. There are known, well–established limitations for all of the most prevalent methodologies, and we can see that the need for modifications to system structure for adaptation is itself a limitation, introducing designed bias and reducing comparability between use cases. More generally, we can also look to the common thread- the mechanisms driving convergence, as a source of limits. The need to create representations, on any level of abstraction, to drive solution generation suggests that a more generalized approach can address these problems.

Because the limiting factors we are considering here are endemic to both machine learning and artificial intelligence algorithms, we may suspect that a resolution will incorporate features of both, as the pair of fields spans a common working domain, namely sequential planning. Further, there is a wide diversity of algorithms addressing an equally wide range of domains under this very broad umbrella- we can thus hypothesize that there are a number of elements in extant frameworks which can be combined to produce such a system, given a suitable unifying framework. Indeed, through our examination of literature it becomes clear that this is not only true, but that those features are anything but rare.

From these observations, we can conclude 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.

Herein, we present a joint learning and planning system henceforth referred to as the Goal Agnostic Planner (GAP). The originating impetus for GAP was, as indicated, to produce a sequential problem solving system which did not use a reward function or world model definition, and could find a path to any reachable state from any initial state. To achieve this objective, we draw concepts from a variety of extant learning and planning systems, and unify them with two additional features- a hypergraph datastructure for storing learned relationships, and a maximum probability path inference algorithm operating on a subgraph within the hypergraph.

In addition to achieving these goals, the GAP also demonstrates several beneficial properties which are analytically provable and empirically validated. Further, GAP agents demonstrate exceptional performance characteristics while learning complex, hierarchical problems, and by design are inherently transparent and explainable.

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.

1.1.1 Reinforcement Learning

RL agents operate in a state/action framework, learning a quality function for maximizing prospective rewards associated with state/action pairs. The use of the state/action framework as a broad approach to modeling systems has allowed for the development of many variants of reinforcement learning. As attested to by (Kaelbling, Littman, & Moore, 1996) RL can readily be applied to determining a wide range of action policies for sequential problem solving. For instance, Temporal Difference learning by (Sutton & Barto, 1987), SARSA, developed by (Rummery & Niranjan, 1994) and the classic Q-Learning by (Watkins, 1989). (Wu, Say, & Sanner, 2020) presents a more recent approach, using deep neural nets to learn transition models for planning applications in domains of nonlinear systems control.

This diversity shows that the state/action framework is highly flexible, but a common through–line for reinforcement–based systems is the necessity of reward function design for convergence, such as expressed for Q-Learning by (Watkins & Dayan, 1992). As such, performance of an RL agent is predicated on the quality of this function, a relationship explored in detail by (Matignon, Laurent, & Le Fort-Piat, 2006). An additional limitation expressed both in (Koenig & Simmons, 1996) and (Grzes, 2017) is goal orientation. Reward functions are constructed in relation to a specific objective set, which makes training of the agent specific to that set only. It is sometimes possible to transfer RL training– (Taylor & Stone, 2009) investigated conditions for transference in detail– it is a different matter to solve an already learned problem for an alternate goal state.

Of particular note is that in learning the value of the reward function, not only is a fixed design parameter relative to the goal being learned, but also additional observational data which may be pertinent to future operations is lost. With our method, we seek to store and apply the learned *state relationships* in a way which can be applied to problem solving between any pair of reachable states, preserving all such relationships for later use. We thus eliminate both goal dependence and information loss by extension of the state/action framework into a higher dimensional state/action/state space.

1.1.2 Markov Decision Processes

Markov Decision Processes analytically estimate behavior of systems under uncertainty, and enable predictions under time evolution. (White, 1985) discusses, from an early standpoint, the diversity of cases in which MDPs find use in real world applications. (Van Otterlo & Wiering, 2012) even goes so far as to discuss MDPs as the 'de facto' standard for sequential learning, a position that would be difficult to dispute in any but the most narrow fields of research. As examples, (Ding, Smith, Belta, & Rus, 2014) presents a method for solving optimal control problems; (Karami, Jeanpierre, & Mouaddib, 2009) shows an application which schedules manual and autonomous activity in a collaborative robotics task; and (Fox, Barbuceanu, & Teigen, 2001) approaches policy planning of material acquisition under an agent–based framework.

While considered more flexible than most machine learning systems, MDPs are still reliant on careful modeling of the system in question. Further, effective reward functions are critical to solve for optimal MDP policies. (Dimitrov & Morton, 2009) discusses the construction of action sets for MDP formulation as a *design methodology.* Though not the explicit point of that paper, this approach illustrates the presence of implicit optimality conditions embedded within problem structure itself. (Szepesvári & Littman, 1996) investigates the use of reinforcement learning to supplant knowledge of MDP reward functions, showing, vis-a-vis convergence properties of reinforcement learning, that design of the reward is critical to productive application of MDPs.

(Steinmetz, Hoffmann, & Buffet, 2016) discusses the problem of parametrically identifying probabilities associated with goal achievement in a Markov process, examining three cases- maximum probability, thresholded probability, and and approximate probability. Via analysis of the L1 norm of our policy-derived transition matrix, we are able to explicitly derive these probabilities for the GAP agent, as well as identify the probability of transitioning to states from which the goal is unreachable.

A special case of Markov decision processes is the Stochastic Shortest Path (SSP, or total reward un–discounted MDP) problem most notably investigated in (Bertsekas & Tsitsiklis, 1991), contains some specific relationships to the model we present for the GAP. SSP problems seek to identify the optimal stationary policy for an MDP with stochastically varying costs, similar to our objective of identifying optimal goal achievement rates with varying action results. (Guillot & Stauffer, 2020) discusses some outstanding issues with policy definition related to the implementation of SSP policy determination. We implement a bounded polynomial-time optimal solution in a special case (uniform cost) of the SSP, while generalizing the State/Action framework to account for dimensionality beyond that associated with MDPs in general. Additionally, we present a solution for the maximum probability problem which requires no iteration to identify the optimal policy.

1.1.3 Automated Planning & Learning

Automated planning is one of the oldest fields associated with artificial intelligence, and adaptive methods (including learning) have been investigated since the field's inception. The Stanford Research Institute Problem Solver (STRIPS) proposed in (Fikes & Nilsson, 1971), potentially the most iconic problem solving system, is an excellent example of this. It has been applied in various forms to many different problems, with an ecosystem of fine–tuned implementations to approach various problem domains. (Lekavỳ & Návrat, 2007) indicates that STRIPS is computationally complete- within properly formatted world spaces. They vindicate the general adoption of STRIPS, while alluding to the specific limitations leading to the development of so many focused variants.

(Geffner, 2000) proposes a system for increasing the domain expressiveness of world models to address this. (Sacerdoti, 1974) presents a modification which operates within an abstracted world model, showing substantial improvements in planning capacity by leveraging efficient representation. Retrospectively, (Lekavỳ & Návrat, 2007) indicates that this is not strictly necessary for generalized application of the STRIPS algorithm, but it speaks to the importance of model design. (Hunter & Thimm, 2017) presents a probability-based belief model for uncertainty tolerance in their Abstraction Augmentation- a notion we extend and adapt to state abstraction as a transform. Further, (Hostetler, Fern, & Dietterich, 2017) evaluates state abstractions as applied to tree search, in particular examining the function of state–abstraction as a partially observable condition on an MDP, similar to our state-mixing interpretation of abstractions.

There are also many relatives to STRIPS, the key feature linking them being the casting of solution generation as a search problem. One such competing method is the Hierarchical Task Network, originally outlined in (Sacerdoti, 1975). HTNs have a similarly large number of implementations, and are noted by (Georgievski & Aiello, 2014) as possessing many of the same limitations as STRIPS. They have been considered more expressive, at the expense of requiring even more detailed models. Further, (Lekavỳ & Návrat, 2007) demonstrates that both HTN and STRIPS are functionally equivalent in practical application, making the distinction one of design utility, rather than functional applicability.

A related standard is the Problem Domain Description Language, initially proposed in (McDermott, 2000) and later extended by (Fox & Long, 2003). Further development was made by (Younes & Littman, 2004) to incorporate probabilistic effects of actions. (Gutiérrez-Basulto, Jung, Lutz, & Schröder, 2017) discusses a class of probabilisitic description languages which extend first-order logic, relating them to temporal logic DLs, illustrating the relationship between temporal reasoning and stochastic reasoning. (Pineda & Zilberstein, 2019) further evaluate model reductions for automated planning. They note a goal-state mapping condition analogous to our convergence condition in abstracted domains, and connected component analysis to identify the presence of dead ends, a method we extend through our trap net analysis.

1.1.4 Combined Models

Consistently, we see that the need for well-designed and models drives innovation. Efforts towards the use of learning to supplement modeling and reduce reliance on design are a natural conclusion. (Zimmerman & Kambhampati, 2003) and (Jiménez, De La Rosa, Fernández, Fernández, & Borrajo, 2012) discuss a wide range of algorithms which apply learning methodologies to classical planning systems. (Jiménez et al., 2012) elucidates two major issues ubiquitous within automated planning which are well addressed by learning:
(1) 'Accurate descriptions of learning tasks', and
(2) 'failure to scale up or yield good quality solutions'.
(Bylander, 1996), though published earlier, even specifically addressed bounding requirements related to problem (2) in propositional STRIPS-like planning explicitly. (Lüdtke, Schröder, Krüger, Bader, & Kirste, 2018) discusses outstanding issues in the use and generation of abstractions for probabilistic inference.

As a contrast to typical search-based methods, (Blum & Furst, 1997) presents Graphplan, which operates as path–planning on a task graph. The author notes that this allows for much more tractable planning speeds which are polynomial time bounded. The authors later extend their work to probabilistic planning in (Blum & Langford, 1999), however they directly acknowledge limitations associated with multiple overlapping action results– a special case of problem (1) above, an issue we specifically resolve using the hypergraph learning structure. In (Konidaris, Kaelbling, & Lozano-Perez, 2018), the authors develop a system designed 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, inheriting similar limitations. Similarly, (Leonetti, Iocchi, & Stone, 2016) combines reinforcement learning with search-based planning, implementing their DARLING algorithm. (Vodopivec, Samothrakis, & Ster, 2017) directly relates reinforcement learning to tree search methods, and similarly demonstrate that incorporating machine learning with traditional search methods can produce highly effective systems.

Each of the discussed algorithms incorporates specific elements of two or more prior problem–solving systems towards the objective of alleviating limiting factors in certain problem cases. The pursuit of joint systems, and the success demonstrated with them, illustrates from a practical perspective the value in unification and consolidation of features across divergent solution–finding algorithms.

1.2 Contribution

As we have seen, a substantial limitation of machine learning systems is the need for the determination of objective functions to drive convergence of learning to an optima. This imposes constraints on these functions and on basis of those constraints, performance guarantees, such as described in (Vidyasagar, 2020) via the Bellman equation, can be derived. (Baird & Moore, 1999) presents a similar argument, following the use of Gradient Descent, as another example which illustrates the point of the optimization argument. The constraints themselves are thus intimately tied with performance.

By contrast, MDPs and planning algorithms typically rely on the quality of the world description, a property expressly investigated in (Dimitrov & Morton, 2009), where the authors explore the problem of optimal design of action sets for MDPs. Additionally, the use of heuristics or modeling restrictions for reduction of the problem space to tractable sizes, as seen in (Dicken & Levine, 2010), has been investigated towards the end of improving model representativeness while maintaining efficiently computable space sizes. Again, the limitations in design directly underwrite effectiveness.

While success has been seen with these methods, there are still inherent limits imposed by world construction and design of space reducing rules. While automated planners rarely include implicit goal dependence, they do not generally provide for direct adaptation and expansion of their state spaces, detailed as a primary challenge both early in (Sacerdoti, 1974) and substantially later still in (Jiménez et al., 2012).

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 $O(n^2)$ using Dijkstra's Algorithm, (Dijkstra et al., 1959), rather than semantic logic. This lets us address 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 loses no data, and learns a representative model by observation, without human design influence. It therefor combines aspects of prior work approaching these outstanding problems in a coherent single system: removing the reward function, implementing non–search planning, and losing no state-to-state observation data.

By implementing this holistic approach to management of the learned data and the planning algorithm we are able to derive several benefits, some as a result of the specific design of the GAP, and some emergent results of its structure. From the former, GAP agents are goal agnostic, require no design of either reward functions or world models, operate in polynomial time, and are natively explainable. With regard to the latter, we show that solutions are globally optimal and exhibit exponentially bound goal achievement rates, derive a metric which parameterizes the probability of entering states from which the goal is non–reachable, 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.

Additionally, we investigate in detail three problem domains commonly used to study machine learning and automated planning algorithms in order to demonstrate the effectiveness and practical impact of these properties in applied contexts. We apply GAP agents to a traditional sequential problem–solving task such as is solved by STRIPS, to a joint domain combining the TAXI problem with maze navigation, and finally to solving the Tower of Hanoi puzzle. In addition to showing convergent learning of these problems with the predicted learning curves, we also perform experiments under varying levels of artificial error and with abstractions to validate the analytic predictions regarding disturbance and perturbed systems.

The remainder of the paper is organized as follows: in the next section, we define the fundamental components of the system as we will be using them throughout the paper. Following that, in Section 3 we present the algorithms and data structures which comprise the GAP algorithm. Section 4 details the analysis of the algorithm, including proofs of efficacy, the dynamic behavior of GAP agents, analysis of the effect of abstractions on performance, and finally demonstration of learning convergence. In Section 5, we present our experimental procedures, demonstration cases, data collected from these experiments, and discussions thereof in the context of the prior analysis. Finally, in Section 6 we conclude the paper and discuss avenues of future research.

2. Definitions

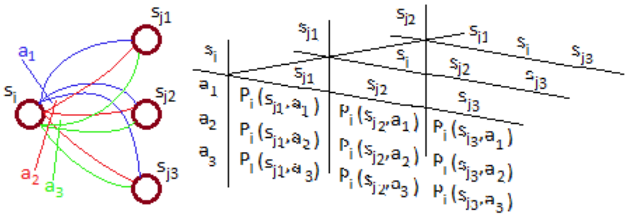
Herein, we define all the components and terms which will be used in the succeeding sections, in particular those relating to the hypergraph modeling system, its associated auxiliary data structures, and the mechanisms used for planning.

2.1 Agent & Occasions

The agent is presumed to be the portion of the system capable of making decisions and effecting the world. It is defined by the capability to register a set of perceptual *states* (denoted $\mathcal{S}$) and 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*: $o_k = a_l(s_i) \to s_f$, as distinguished from a more traditional state/action pair.

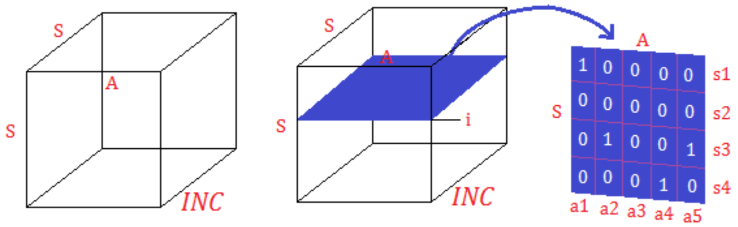
2.2 Hypergraph Learning Model

We implement a learning system for which the basic units are occasions, defined in the prior section, and are recorded within a 3-dimensional structure of size $|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|$. Within this array, cells at locations (s_i, s_f, a_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: a higher dimensional analog of a graph in which each state



s_i	s_{j1}	s_{j2}	s_{j3}
a_1	$P_i(s_{j1}, a_1)$	$P_i(s_{j2}, a_1)$	$P_i(s_{j3}, a_1)$
a_2	$P_i(s_{j1}, a_2)$	$P_i(s_{j2}, a_2)$	$P_i(s_{j3}, a_2)$
a_3	$P_i(s_{j1}, a_3)$	$P_i(s_{j2}, a_3)$	$P_i(s_{j3}, a_3)$

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.



	a1	a2	a3	a4	a5
s1	1	0	0	0	0
s2	0	0	0	0	0
s3	0	1	0	0	1
s4	0	0	0	1	0

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.

is a node and initial and final state labels express edges corresponding to out edges and in edges respectively. For each node, then, the dimensional expansion results in multiple edges between each node, corresponding to varying actions. Each action may have multiple results, and these results may overlap with other actions' results. Thus, we have multiple links between states, each connected with a different action, hence the construction of a hypergraph as opposed to a standard graph. This structural change allows us to contend with the challenge of overlapping action results identified in (Blum & Langford, 1999).

To represent the hypergraph in practice, we implement a three dimensional array, INC, in which the location $INC[i, j, l]$ contains the number of times occasion $a_l(s_i) \to s_j$ has been observed. An illustrated reference is presented in Figure 1. Using the member entries of the INC array, and sums along slices within this array (as illustrated in Figure 2), the relative probability of differing occasions can be computed.

Using this state/action/state framework as a basis for learning, we are able to obviate the need for world model design. As each transition is counted, the model created by observation encapsulates the maximal amount of information for embedding within the INC array. One way to consider the comprehensiveness of this representation is to consider an arbitrary number of observations, combined with a true probability model- amortized, the value in each entry in INC would be equal to the number of total observations weighted by the conditional probability of the occasion being observed. As a special case, in a fully representative state space with no hidden states and fully deterministic operation, each initial state/action pair would have only one resultant state with a non-zero observation count.

2.3 Probability Models

As mentioned above, 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, we elect to choose actions based on the most probable *outcome* of taking an action from a given state. Calculation of the associated probabilities is thus given by the following formula:

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

This approach conceptualizes each action in terms of which state is most likely to be observed next as a result of taking that action, and hence we refer to it as the *a priori* probability.

In the second model, we select actions based on the most probable cause of a given state/state transition, which we term the *a posteriori* probability. That is to say, we select, for each $s_i \rightarrow s_j$, the probability of the action a_l having caused this transition relative to other actions. Therefor, in this model, the probability calculation is performed as:

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

We can see the difference, for instance, if we presume a simple problem with three states and three actions, with an INC slice at s_i as demonstrated in Table 1:

s_i	a_1	a_2	a_3
s_i	3	1	7
s_{f1}	2	5	1
s_{f2}	9	1	2

Table 1: Illustrative example of a state/action slice

We can see that Equation 1, when applied to $o_k = a_1(s_i) \rightarrow s_{f1}$, gives a probability of $\frac{1}{7}$, whereas Equation 2 produces $\frac{1}{4}$. The former tells us that there is a 1-in-7 chance that taking a_1 from s_i will result in s_{f1}, whereas the latter expresses a 1-in-4 chance that the transition $s_i \rightarrow s_{f1}$ can be precipitated by a_1.

In the qualitative sense, then, action policies using the a priori probability model are selecting the sequence of actions most likely to result in goal achievement. This means that, in our policy consideration, each action will be a viable choice, with the most likely subsequent state as the primary outcome, even if multiple actions have the same most likely resultant state. Comparatively, a posteriori policies select the most likely sequence of state changes to reach the goal, meaning each policy choice will examine all possible state/state transitions from s_i in terms of the highest probability action precipitating that transition, even if that action is the also most likely cause of other available transitions.

2.4 Sequences

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, and maintenance of the action list which precipitates the transitions between states, and the states themselves, allow the execution of the solution by the agent. A sequence is 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:

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

Throughout this paper we will be using this joint probability formula, which necessitates the assumption that occasions are conditionally independent. If the chosen states are sufficiently disjoint, this requirement is satisfied; if not, then the probabilities are presumed mixed in the same capacity as fundamental error, which we explore in Section 4.

For each occupied state, the potential effects of an action are represented in the current state slice of the hypergraph, as highlighted in Figure 2. Each entry in $INC[s_i, :, :]$ therefor also represents an associated probability of relationship between action a_l and the state transition $s_i \to s_j$, depending on which probability model is used.

2.5 Maximal likelihood subgraph

Given a specific probability calculation, and in the context of maximally likely sequences, it is possible to define a class of subgraphs embedded within the hypergraph which contain all component edges of a solution sequence. 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. In this array, the component $< s_i, s_j, 0 >$ is the maximum probability associated with the $s_i \to s_j$ transition, and component $< s_i, s_j, 1 >$ is the index of the corresponding action. Thus defined, we have:

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

$$AFI[s_i, s_j, 1] = \operatorname*{argmax}_l \{P(a_l(s_i) \to s_j)\}$$

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.

The construction of each such subgraph from the larger hypergraph structure is accomplished simply by taking the projection of maximally probable elements from any directional slice of the hypergraph. For instance, to construct the a posteriori subgraph, the hypergraph is compressed along the $|\mathcal{A}|$ axis, retaining the maximally probable a_l associated with each state transition in the $|\mathcal{S}| \times |\mathcal{S}|$ space. Maintenance of this maximal probability property is handled with the array/linked list datastructure as described in the following section, and is presumed to proceed as the learning phase progresses, allowing for in–situ sorting and efficient updates throughout the datastructure as each occasion observation is recorded.

Each of these compressed arrays can represent a traditional graph, which feature we will use for efficient computation of solution sequences. However, the two are not identical in structure, and consequently remit different associated probability calculations with comparably diverging interpretations, as described above.

3. 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.

3.1 Array Linked List

This datastructure combines 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.

3.2 Augmented Hypergraph Datastructure

A hypergraph may be stored in a 3-dimensional array, and we retain the full $|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|$ sized collection of observation counts in such an array. For purposes of planning sequences, however, we must choose a probability model as described above. Under such a model, execution of the planning algorithm need not evaluate all hypergraph links, only the most probable ones, and thus for computational efficiency the 3-dimensional array is augmented with a pair of array/linked lists corresponding to the sorted elements within the maximal likelihood subgraphs.

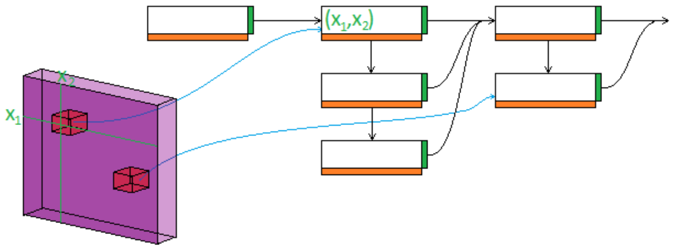


Figure 3: Array/linked list showing the indexed cell locations within the array containing pointers to the corresponding elements in the sorted linked list, which itself contains the data component associated with each array cell, and is organized into columns containing the same number of observed instances.

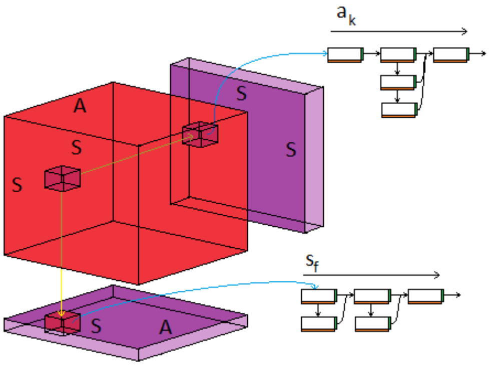


Figure 4: Augmented hypergraph data structure: a 3 dimensional array, each cell of which contains pointers to members of two Array/Linked List objects, each containing a pointer to the corresponding sorted list associated with that state/action or state/state pair, allowing for immediate retrieval.

Using the convenient sorting and addressing features of the array/linked list, each update to the optimal subgraph can be incorporated using $O(1)$ alterations to the linked list containing the pertinent occasion. To accomplish this, the 3-dimensional array is paired with two 2-dimensional array linked lists, corresponding to the a priori and a posteriori probability metrics. Each member of these arrays points to a linked list which contains the sorted members of the slice along the compressed axis, and the graph associated with this compression consists solely of the first (and thus, maximally probable) element along that axis.

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, this corresponds to all cells with constant (s_i, a_l), so the linked list contains $|\mathcal{S}|$ members along the s_j axis. Similarly, for the a posteriori graph, the linked lists are along constant (s_i, s_j), and contain $|\mathcal{A}|$ many links.

3.3 Subgraph Maintenence Algorithm

To maintain the maximal likelihood subgraphs, at each observed instance of an occasion the linked lists containing references to this set of coordinates must be updated. 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, maintenence revolves around correctly rebuilding the link chain at each step, as detailed in Algorithm 1.

Under the continual operation of this maintenance algorithm, the maximum likelihood subgraph slice of the bulk hypergraph data structure is perpetually embedded within the 2D slice corresponding to the first sorted members of each state/state or state/action list. Further, at each update step, the addressing to the linked list element is direct via the augmented hypergraph, and the only operations necessary are the direct comparison of the

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
        $occasionLink.prev = occasionLink.prev.prev$
        $occasionLink.post = occasionLink.current$
        $occasionLink.current =$
            $occasionLink.prev.current$
```

increment counts to the preceding linked objects, which on increment require only rebuilding the links to the prior and current list members, and thus each update's complexity is $O(1)$.

3.4 Sequence Inference Algorithm

With the subgraph compression as described above in place, we are then in position to infer a maximally likely sequence of occasions to achieve a transition between the given, current, state s_i, and some other goal state, s_g. 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.

In Algorithm 2, we formalize this algorithm by calculating the net probability as a result of each sequential action. Additionally, we use the array linked list to make the ordering, member checking, and set operations of the boundary list simplistic and compatible with an implementation of Dijkstra's algorithm.

Dijkstra's algorithm typically searches through the minimum cumulative path sum through nodes within a growing adjacency list of a minimum path tree. In our case, the cumulative sum of distance for each node is instead the highest cumulative probability, joined under multiplication as expressed in Equation 3. In Section 4.1, we present an explicit proof of the optimality of this algorithm. 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, though other implementations could be used with somewhat improved characteristics, or A* if appropriate heuristics are known (discussed briefly in Appendix A.4).

Phrased in terms of 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:

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
        $jointProb(s_j) \coloneqq perm[bound[j]][1] \cdot bound[j].P$
        $s_{maxP} \leftarrow \underset{s_j}{\mathrm{argmax}}(jointProb)$
        $perm \leftarrow (s_{maxP}, jointProb(s_{maxP}))$
        $bound = (bound \cup s_{maxP}.edges) - [e | e(1) = s_{maxP}]$
        $edges \leftarrow bound[s_j]$
        $solution = [edges[s_g]]$
    while $solution[-1][0] \neq s_i$ do
        $solution \leftarrow edges[solution[-1][0]]$
    return $solution$
```

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

the first action in the most–probable sequence σ_{ig} from state i to state g. Each action in the sequence σ_{ig} is selected on basis of inclusion on the maximum probability path between the current state and the goal state (even if the ideal state–to–state transition was not achieved at the prior step).

We can see from this implementation that the decision–making process of GAP agents is imminently transparent. 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.

4. Analysis of the GAP

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 will rely heavily on Markov process analysis techniques to evaluate the performance of the algorithm and demonstrate learning convergence and robustness. To do so, we are using the fact that the GAP algorithm's time evolution is readily modeled with an MDP, and the predictive power thereof allows us to define the temporal behavior of the system. The core difference, however, is that we have defined an action policy at the outset, in the form of the *maximal likelihood path* (Equation 4). This policy possesses useful properties for the Markov analysis, essentially converting the predictive power of the MDP into a problem solution without recourse to iterative policy estimates.

4.1 Optimality of GAP plans

We begin by demonstrating the optimality of the selected policy, so that the subsequent analysis of dynamic can proceed with the assumption of a properly selected policy. Because the maximum likelihood paths are maintained and planned by independent algorithms, it is simple to use the known behavior of Dijkstra's algorithm to show that the plans are optimal in the maximum probability subgraphs. In implementing the subgraph compression, though, one may question whether or not there is information lost relative to the optimal path including all possible elements within the transition space. Therefor we first demonstrate that the optimal path is embedded within the subgraph.

Theorem 1. *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, as the explicit directional slice along which the constituent maximal probability tree is compressed is nowhere referenced within the proof. That each graph contains a maximal slice with respect to either state/state or state/action projections 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 2. *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. □

Given these proofs, we can see that the action policy derived from planning on the maximal probability subgraphs is indeed optimal, and as a result the need for designation of a reward function to drive convergence is unnecessary, with the planning phase implementing a known-optimal policy and the maintenance algorithm preserving the optimal embedding of paths in the AFI subgraphs. These proofs reflect those of (Bertsekas & Tsitsiklis, 1991) for SSP, however are simplified substantially by the structure of the maximum likelihood subgraph.

4.2 Analysis of Agent Behavior Dynamics by Markov Analysis

Because we have been discussing probability–driven state/action transitions it is natural to make a comparison to Markov chains. In fact, it is possible to examine the behavior of the agent in both the planning and learning phases in detail by using a transition matrix derived from the AFI array and the previously derived policy, Equation 4.

4.2.1 Predictive Behavior Analysis

To begin with, we have the maximal subgraphs, AFI. For planning purposes we proceed by finding the highest probability expected path to the goal, but for this analysis, we will not keep a specific starting state in mind. In lieu of this, we instead 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 contains also an annotation of 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: $a_l(s_i) : \{(s_{j1}|P_{j1}), (s_{j2}|P_{j2}), ...\}$. As such, we construct the transition table from these segments of INC for all of the non–goal 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; $\vec{t_g} = \begin{bmatrix}0 & 0... & 1.. & 0...\end{bmatrix}^T$, adopting the absorbing state method of (Steinmetz et al., 2016). From this concatenation, we have the

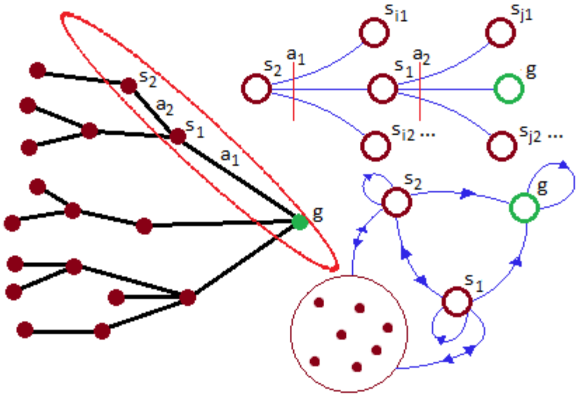


Figure 5: Conversion of Maximal Probability path tree to Markov Chain. highlighting states s_2, s_1, and g, with maximal probability actions a_2 and a_1 linking them, and additional action-caused probability links among themselves and other states shown in the Markov network

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:

$$P_g = \begin{bmatrix}\vec{t_0} & \vec{t_1}... & \vec{t_g}... & \vec{t_j}\end{bmatrix} \tag{5}$$

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; any non-optimal transitions are also embedded in the transition graph so long as they result from *optimal actions*. The optimal action choice for a non-optimal path is also embedded in this graph, as the subsequent path in $T_{P(g)}$ from an incidental non–optimal state resulting from an optimal action will still be optimal with respect to the new state.

We can model the statistical propagation from a starting state by representing the state distribution itself as a vector, $\vec{s_k}$, where k is taken to be an indicator of stepping, incremented each time an action is taken. 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.

One thing to note about P_g is that all 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:

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

Which necessitates that any attractor state *either* be an eigenvector of P_g with $\lambda = 1$, or be made arbitrarily close to an initial state vector. The $\lambda = 1$ eigenvector of any Markov transition matrix is also a steady state of the transition matrix. However, P_g can be demonstrated to have no steady-state *distribution* by virtue of the presence of s_g. Presume, w.l.g, that we order the stochastic vectors comprising 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, structured as $< \vec{0}, 1 >$ because we presume that execution terminates upon reaching the goal state. With this structure, 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, as a function of step number, of the agent's occupation of states under the GAP algorithm. From it, we can see that the probability of reaching the goal state at step k is given by

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

Which equation expressly describes the probability of any state transitioning to the goal state at a given step k. 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, T_g^k is as well, and consequently $P(s_i \to s_g|k)$ is monotonically increasing in k, and therefor P_g has no steady state. This means that s_g must, by definition, be an attractor state, as it is identical to its own start-state distribution. Second, this also means that 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 possibly leading to states not on the maximal probability tree.

We can therefore not only predict the expected behavior of the agent, but also define the probability of reaching the goal state at any given step number, and thereby establish the expected number of steps to reach the goal. Perhaps even more importantly we can also, for non-goal states which are also attractors, calculate the probability of the agent being sequestered at said states by incidental variance.

Additionally, 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 progressive magnitude of any vector representing the probability distribution across the system states as a function of step time. Because we have demonstrated that there are no steady states embedded within P_g barring those constructed in the same form as a goal state, $< \vec{0}, 1, \vec{0} >$, and because T_s^k is positive definite with all entries less than or equal to one, the probability distribution of goal transitions must be strictly monotonic over time.

4.2.2 Trap nets

We have assumed the form of the goal state as $< \vec{0}, 1 >$, which makes it an attractor state. We noted above that other states with unit self–transition probabilities would also act as unwanted attractor states from which the agent cannot reach the goal state, and that otherwise no steady states exist. However, it is also perfectly possible for state sequences

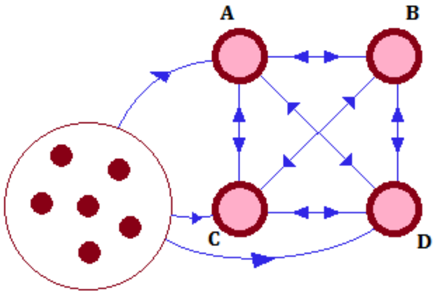


Figure 6: Illustration of subgraph segment from which no path to the goal exists, yet contains multiple transition state cycles. Such regions can present non-steady state attractors from which the agent cannot progress to the goal, hence being considered 'trapped' in the subgraph.

which are not independently attractor states, but which present no path to the goal once reached, to be non-steady attractors. Such a segments we will refer to as 'trap nets', collections of states from which the agent cannot proceed to the goal, such as illustrated in Figure 6.

We can define a subset of states, *tnet*, to represent the states associated with a structure such as this. If we presume to organize P_g such that we align the rows and columns associated with the subnet *tnet*, we can re-cast P_g in the following form, 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:

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

From which we can see that $P(s_{i \in tnet} \to s_g | k) = \vec{0}$ for all k. Further, we will define a system parameter L_{max}: the longest minimum length path between any state from which the goal is reachable and the goal itself. We then have that for any reachable state s_i, $P(s_i \to s_g | L_{max}) > 0$, or there is a non-zero probability that $s_i \to s_g$ has occurred after L_{max} timesteps. Consequently 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 use analysis of the connectivity of states in P_g to determine L_{max}, however it is sufficient to raise P_g to a power which is greater than L_{max} and examine the final row. In any graph the maximum path length between any two nodes is bounded by the size of the graph, and so it suffices to check $P_g^{|\mathcal{S}|}$: 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 probability at any point in time that the system has become stranded in a trap net, given that the trap states have been identified as above and P_g arranged as in Equation 11:

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

Which allows for measurement of the risk associated with the system progressing to an inescapable holding pattern, in addition to the probability of achieving the goal state. 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 (Pineda & Zilberstein, 2019).

With this measurement in hand we can calculate, as a function of time, both the probability of an agent becoming stuck in a trap net, as well as the time evolution of the probability of reaching the goal. Combined, they present a convergent behavior model in which the long-term behavior of the agent can be statistically parametrized, and thus fully define the goal–convergent behavior of the agent, a problem discussed explicitly in (Steinmetz et al., 2016).

4.2.3 Derivation of bounded time performance

As a further consideration, we may examine the behavior of the system proscribed by the transition matrix in terms of the evolution of the L1 norm of T_s^k. This 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. The L1 norm, as all vector induced norms, is submultiplicative, and thus we may write:

$$\|T_s^k\|_1 \leq \|T_s^{k-n}\|_1 \|T_s\|_1^n$$

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

Additionally, because all columns are stochastic, the maximum absolute column sum is paired with the minimum probability single step goal transition. Put otherwise:

$$\|T_s\|_1 = 1 - \min_{\forall a,i} P_g[s_i, g, a]$$

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

For many systems, $\min_{\forall a,i} P_g[s_i, g, a] = 0$, which provides little insight. For any reachable state, however, we may once again use L_{max}, the maximal shortest path to goal length. Then, at $k = L_{max}$, all states from which the goal is reachable have a non-zero transition probability, and L_{max} is also the first time step at which all states may possibly have transitioned to the goal state.

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

Which allows us to calculate a minimum time at which all states will surpass a certain threshold likelihood of having transitioned to the goal without projecting the system forward in time arbitrarily.

In general, for some minimum transition probability threshold P_{thresh}:

$$1 - P_{thresh} \leq \|T_s^{L_{max}}\|_1^{k_p - L_{max}}$$

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

Which establishes an expectation curve for the progression of states towards the goal in terms of both the dynamics of T_s and the effective 'distance' between the starting state and the goal. Writing the relation slightly differently, as $1 - \|T_s^{L_{max}}\|_1^{k_p - L_{max}} \leq P_{thresh}$, we can see that the probability of transition to goal is bounded by an exponential growth rate- the minimum probability threshold reached is limited by an exponential asymptotic function approaching unity. This illustrates that even under stochastic disturbances, the path planning algorithm will be highly efficient.

4.3 Analysis of Robustness under perturbation

Validation of a learning algorithm against an abstract model is often challenging due to the difficulty of parameterizing abstractions, and the fact that most learning algorithms themselves implement some level of state abstraction inherent to the learning structure. Defining abstraction is a complex topic, and settling on metrics for measuring it even more so. As our system retains all observed information in a probabilistic fashion, it presents a distinct opportunity for evaluation of abstraction as a disturbance model. Towards this end, we will be modifying the derivations in the prior section by adapting them for a transform–based model which mixes and conflates state transition values.

For our purposes, we will consider an abstracted learning problem to be one in which there is some mapping $\alpha()$ which transforms a large state space S into a more compact 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 $\{\alpha(s_i)|\forall s_i\}$, and discrepancies of incompleteness will be modeled in terms of the states which are present in either set. Note that by definition $|\alpha| \leq |\mathcal{S}|$. This probabilistic mapping model is similar to that used by (Hunter & Thimm, 2017).

Though we are focusing on the concept of a representative, state–compressing abstraction, the perturbation we will introduce here also applies to an uncertainty model. In such a model, an uncertainty perturbation acts as a mixing function which skews the probability of state measurements, which alters the transition probabilities themselves as recorded by the GAP agent. As such, we can note that a map modeling uncertainty as a disturbance transformation can be constructed in the same manner, with $|\alpha| = |\mathcal{S}|$.

Presume that we have an $|\alpha| \times |\mathcal{S}|$ transformation matrix, α_T which contains in each cell $\alpha[j,i]$ the probability $P(\alpha(s_i) = \alpha(j))$ that the i^{th} 'true' state is mapped onto the j^{th} abstracted state. With a given state probability vector $\vec{s}_t$, then, 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 a learned 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. This pair of transformations allows the conversion from the *probability space* of the abstraction into the probability space for the problem. It is notable that this transform does not allow for conversion into the true *state space*, even if α_T is known perfectly, as α_T^+ cannot unmix states which are combined, whether stochastically or deterministically. 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 $t = 2$, we have that $\alpha_T P_g^2 \alpha_T^+ = \alpha_T P_g \alpha_T^+ \alpha_T P_g \alpha_T^+$, or $\alpha_T^+ \alpha_T = I$, implying that the columns of α_T must be linearly independent, a natural conclusion given the definition of $\alpha()$ being a surjection on stochastic vectors.

We can thus derive an expanded form for the abstracted transition array in terms of the true transitions by taking the partitions $\alpha_{Ts} = \alpha[:, : -1]$, $\alpha_{Tg} = \alpha[:, -1]$, $\alpha_{Ts}^+ = \alpha^+[: -1, :]$, and $\alpha_{Tg}^+ = \alpha^+[-1, :]$, recognizing that both arrays must be stochastic transforms, due to the action on $\vec{s}$:

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

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

Transforms between the probability spaces, however, allow us to apply the analysis in Section VI to convergence in the abstracted space. We assume that the abstracted model is convergent, and wish to show that the 'true' system will converge as well. Taking Equation 7 where we annotate: $\vec{t_{\alpha g}} \cdot \sum_{l=1}^{k-1} T_{\alpha s}^l + \vec{t_{\alpha g}} = V_p$:

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

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 $\vec{P}(s_i \to s_g|k)$ can be determined:

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

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

Convergence of P_g can be expressed as $\vec{P}(s_i \to s_g|k) \to \vec{1}$, so:

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

$$0 = ||\alpha_{Tg}^{+}||$$

Which shows that the convergence of the true system to the goal, given convergence of the abstracted state, is predicated on the transform between the true goal states and the abstracted goal states being *onto*, analogous to, but distinct from, the convergence conditions derived in (Pineda & Zilberstein, 2019). Note that this does not preclude early convergence due to canceling factors between the dynamics of V_p, $T_{\alpha s}^k$, and α_{Tg}^{+}, but rather determines the asymptotic behavior of the system. This demonstrates that the GAP solutions will retain their optimality and convergence properties when learning in perturbed spaces when this condition is met. Note that this mixing condition mirrors the observability model utilized in (Hostetler et al., 2017).

4.4 Impact of Perturbed State on Performance

Given this condition on the abstraction function, and presuming again that the abstracted P_α is convergent, we can further extend the derivations in Section 4.2 to the case of solving an abstracted system.

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 $P_\alpha \to P_g$ and not to $P_g \to P_\alpha$– that is, 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 the prior analysis for the abstracted case:

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

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

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

Which describes how the inclusion of the abstraction modifies the minimum expected time to achieving the goal state relative to the timescale predicted by P_α alone, or put alternatively, presuming the learned state space fully represents the behavior of the system without hidden variables.

By examining the expression above, we can make some inferences about the impact of α_T on convergence 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:

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

So that $Q(\alpha_T)$ is directly correlated to the impact α_T has on performance, resolving the metric problem brought up in (Lüdtke et al., 2018). It is worth mentioning that because α^{+}_{T} is not strictly positive definite, conditions under which $Q(\alpha_T)$ improve system performance are possible, albeit difficult to design. This suggests that the introduction of an abstraction may either improve or reduce efficacy, depending on the nature of the abstraction. Improvements may seem counter-intuitive, but consider the way a substitution may reduce the number of steps needed to solve an algebraic equation. In the context of planning, certain simplifications may indeed bias portions of the graph towards choosing probabilistically identical, but shorter paths, minimizing the outlier chances of stochastic variance increasing average path-to-goal length, an observation also noted by (Hunter & Thimm, 2017).

Empirically, we can also approximate this measure by calculating the constituent components of the relation between k_p as predicted for the system under abstraction and the measured $k_{p\alpha}$ as the average number of steps to reach the goal over many iterations:

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

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

Which allows us to calculate a measure of the relative efficacy of the abstraction from the learned transition array, maximum shortest path, and measured minimum and average path lengths across samples. Using this metric, we can thus specify the conditional impact on performance with a perturbation model, underwriting the effectiveness of the GAP for operating under an abstraction or uncertainty. Further, in Section 5, this relation will allow us to examine the expected learning curves for the agent under training. This allows the GAP agent to address one of the principle limitations discussed by (Lüdtke et al., 2018)- symmetry breaking- in a grounded way, tying similarities between performance under uncertainty and abstraction together in Section 5.4.1, and via heuristic construction discussed in Appendix A.4.

4.5 Learning Convergence

We can evaluate the behavior of the agent as a learning system by modeling the learned AFI matrix as an abstracted function of the true state which becomes more accurate as learning progresses. We may begin by presuming a transform which maps the true states onto a distribution which reflects the initial assumptions of the learning model– namely, a uniform distribution from which actions are initially chosen randomly:

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

With these, we can see that:

$$||\alpha_{T1}||_1 = \tfrac{|\alpha|-1}{|\alpha|} \quad ||\alpha^{+}_{T1}||_1 = \tfrac{|\mathcal{S}|-1}{|\mathcal{S}|}$$

Further, presuming that P_g is the asymptotic learning goal matrix and P_α is the non-learned array, we have that $|\alpha| = |\mathcal{S}|$. We can approximate the expected learning curves over many samplings by presuming, from random choice prevailing at non-sampled occasions, an amortized update at each step k derived from Equation 2. Examining an update to a single state vector, we can note that in the average case at step k the state has been visited $\frac{k}{|\mathcal{S}|}$ times. Given the probability vector $\vec{s}_{\alpha i}$ from P_α, then, the individual counts can be expressed as $\frac{k}{|\mathcal{S}|}\vec{s}_{\alpha i}$, also via Equation 2. Further, the probability 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 the prior occasions with the new, 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 $k \to \infty$:

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

$$\alpha_{Ts}^{+} = \mathbf{I} \to \alpha_{Ts} = \mathbf{I}$$

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

4.5.1 Derivation of learning curve form

In addition to proving that GAP agent training is convergent, we can also demonstrate that the learning rate will be tractable by deriving the average form for these curves over many instances. Equation 17 allows us to determine the amortized form of the transition array over time- we can express it as:

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

In which the terms $\frac{\mathbf{1}}{|\mathcal{S}|} - P_g$ and P_g are clearly time invariant, with the dynamic behavior explicitly governed by the reciprocal of the timestep, $\frac{1}{k}$, and thus the average learning curve will follow a reciprocal pattern $k_{p\alpha}(k) = A\frac{1}{k} + B$. B is naturally the asymptotic average 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 $\frac{|\mathcal{S}|-1}{|\mathcal{S}|}$, and thus:

$$A = 2(L_{max} - k_p)$$

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

Which establishes an expected average for the performance of the unlearned system as a biased random walk on P_g, with $k_{p\alpha}(1) = 2L_{max} - k_p$, as well as the general form for the learning curve of the GAP being an offset reciprocal function of step number, showing that the learning will progress in an expeditious manner, with the rate determined by pertinent constants relating to problem structure.

5. Demonstration Cases

In this section we explore the application case behavior of GAP-based agents' learning in three example problem classes: a classical STRIPS type problem, a complex joint domain of the TAXI domain and Maze Navigation problems, and the Tower of Hanoi puzzle. These tasks includes specific hierarchical components, complex and large state spaces, and have been used previously as benchmark trials for machine learning algorithms. For instance, the TAXI domain by (Dietterich, 1999), Mazes by (McCallum, 1995), and the Tower of Hanoi by (Knoblock, 1990).

5.1 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.

5.1.1 Training Process

To train the agent, the AFI datastructure is initialized with a uniform distribution of random values, as expressed analytically in Section 4. Upon observations of occasions, the corresponding INC cells are updated, with the AFI array modified proportionally to the actual number of measured values, as per Equations 1 and 2. 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, with the INC array persisting between epochs.

5.1.2 Error Induction

To investigate system performance under uncertainty, we also artificially induce error in some trials. Error induction is achieved in planned execution by a random threshold process which, a select proportion of time, executes a random non–planned action in lieu of that in the plan. Action modifications are selected as the medium for two reasons: first, discontinuous transitions within the world do not model real world uncertainties well. For instance, a mobile robot may accidentally slide, changing position as though it had selected a different trajectory, but it is unlikely to teleport. Second, because modeling of a state estimation error can be approximated well in most cases by an action change: if there is an error in state detection, then the agent will instead take action based on the fault state, which under a uniform distribution error model will be an action independent of the current state.

5.1.3 State Generation

Because we are intending to make a fully abstracted learning agent, every problem is built without labeled states, simply initialized to a rough estimate number of possible states. The simulation models are designed to output string type state reports when polled for information, and from these strings, a simple hash algorithm generates a lookup table for the agent to use. As more states are discovered, the implementation software increases the size of the SSA arrays and corresponding array slice structures to accommodate larger lookup tables. As a result, while we often will perform experiments in environments with very large numbers of states, the total state space occupation for the GAP agent will only include those states observed in learning, using sparsity to our advantage rather than detriment.

5.1.4 Calculation of Metric parameters

Throughout this section, we will be demonstrating the effectiveness of the GAP algorithm by measuring a suite of parameters related to performance characteristics. First and foremost of these are the metrics which represent the measured learning curves in the form of reciprocal functions as predicted in Section 4, Equation 18. Towards this end, we calculate best fit equations and measure of their accuracy: R^2 for the fit of $k_{p\alpha} = \frac{A}{k} + B$, and percentage off–linear averages of linear regressions on the plots of $(\frac{1}{k}, k_p)$ calculated for N sequential data points on the curve as:

$$\sum_{\forall n \in N} \frac{1}{N} \frac{|k_p[n] - (A\frac{1}{n} + B)|}{k_p[n]}$$

In addition to these measures to validate the predicted learning curve, we also calculate and compare approximations of k_p and L_{max} to establish a correlation between the analytic predictions in Section 4 and the measured data. For k_p, calculations are made both by computation of the fit curve for Equation 18, and by averaging performance levels after system convergence. L_{max} comparisons are made between two predicted relationships, Equation 16 directly, and Equation 13 by estimation over multiple probability levels of induced error for P_{thresh}. As L_{max} is a problem constant, the inequality Equation 13 can be used to empirically identify the values for P_{thresh} at which the inequality is no longer valid, and thus infer an estimate of L_{max} based on measurements of $||T_s||$.

5.2 STRIPS-type Problems

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

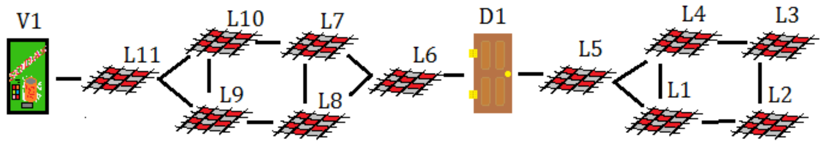


Figure 7: Illustration of a basic STRIPS-style world, containing linked location states (L_i) and multiple independent actionable states (V_1 and D_1)

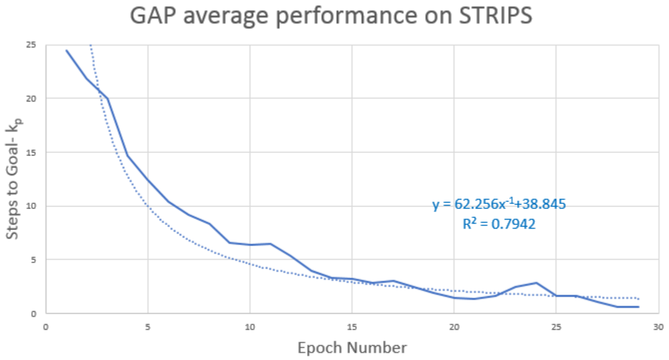


Figure 8: Average learning curve over instances of varying error from 0% to 50% in the STRIPS problem space, with the reciprocal fit curve plotted superimposed

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%. This curve is for an online implementation, with random starting locations in the right half of the world. On this plot we see a reciprocal 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 curves of the GAP algorithm on this problem at each induced error level independently. Each curve is the average performance as averaged over 50 trials. We can see from these curves that the learning tends to follow the same reciprocal pattern as the general curve, with variance in asymptotic performance shifting due to the increase in error rate elevating the expected number of steps to reach the goal.

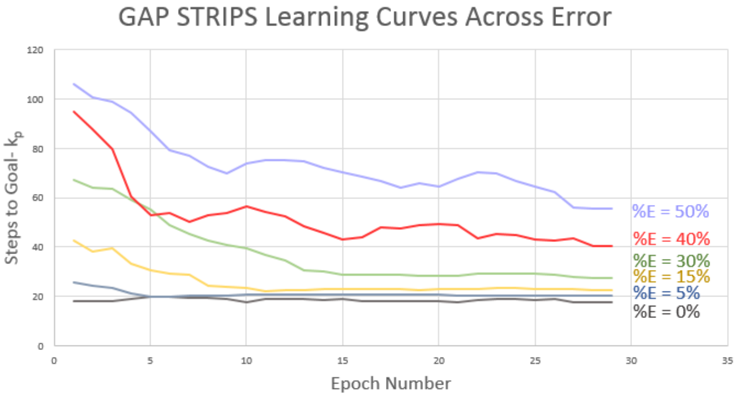


Figure 9: Learning curves for the STRIPS problem across levels of induced error from 0% to 50%

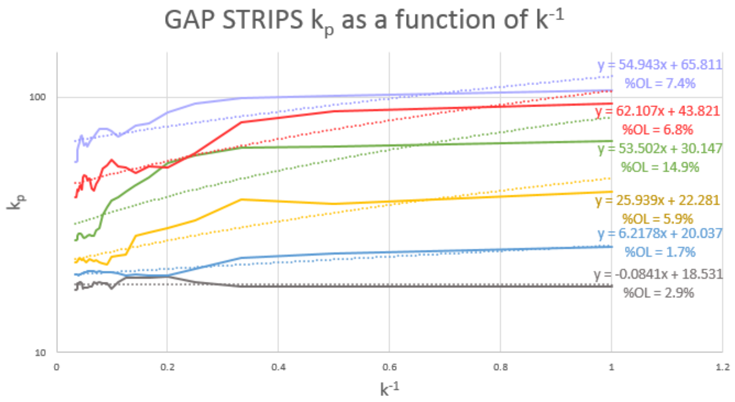


Figure 10: Plots of k_p as a function of $\frac{1}{k}$ for various error levels, along with measures of the deviation from linearity in terms of % off–linear behavior, showing close correspondence to the predicted learning curve form of $k_p = A\frac{1}{k} + B$

To reinforce the reciprocal relationship, we also plot the linearization of these curves (in Figure 10) along with the off–linear percent labeled for these curves. For each plot but one, the deviation from linear fit is in the single digits, with the greatest deviation being for the 30% curve, with a 15% average off–linear error. These measurements serve to validate the prediction of Equation 18 that the GAP algorithm will express reciprocal learning curves.

In addition to these linearized plots showing correlation between $\frac{1}{k}$ and steps to goal, we also highlight the correlations between k_p as predicted by the asymptotic behavior of the data itself and the fit reciprocal curve. We also calculate L_{max} from Equation 18 and as predicted by the threshold in Equation 12. Both measures are presented on Table 2, along with the corresponding percent errors. Here, we can see that the differences between the asymptotic k_p and the fit function are small, 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.

P_{thresh}	k_p Meas:	k_p Pred:	%E		L_{max}
0%	18.53	18.10	2.30%	Meas:	25.30
5%	20.04	20.21	0.87%	Pred:	27.29
15%	22.81	22.76	2.11%	%E	7.8%
30%	30.15	28.14	7.12%		
40%	43.82	42.07	4.15%		
50%	65.81	57.40	14.65%		

Table 2: Comparison of measured and predicted values for systematic form analysis as in Section 4, as calculated from the performance on the STRIPS problem learning curves

What the successive curves illustrate at the higher levels of induced error is that there is a close hew to Equation 18, even as the learning process is disturbed by a persistent error signal which distorts the true model– essentially an abstraction with a substantially small $Q(\alpha_T)$.

Of note is the 50% error case, for which the discrepancy is larger, roughly twice the level of the next greatest deviation. However, an introduction of 50% error into the action of the agent is extremely substantial, and it is reasonable to expect that the learning performance will degrade. Qualitatively speaking, as the induced error rate increases, P_α behaves more and more like a random uniform stochastic process than the underlying 'true system' P_g. Referring back to Equation 16, we can see that for a certain magnitude of error (expressed in terms of $||\alpha^+_{Ts}||_1$), the limit of $k_{p\alpha}(k)$ will grow to the point at which the difference between $k_{p\alpha}(0)$ and the asymptotic performance is effectively negligible. Because of the exponential form of Equation 16, the level will be highly sensitive to the exact value of $||\alpha^+_{Ts}||_1$, but it represents a threshold at which learning is no longer effectual. In more rigorous terms, $\lim_{k\to\infty} k_{p\alpha}(k) \to L_{max}$, and so the function $k_{p\alpha}(k)$ no longer properly behaves as a reciprocal, but as a constant function, exactly the expected behavior of an attempt to learn a uniform random process.

In the late stages of learning for each of the error variant curves the performance levels are clearly tiered in proportion to the error rate, which shows that increasing uncertainty directly increases the average number of steps needed to reach the goal state, as predicted by Equation 16, while preserving the learning curve form. Further, we have the precise convergent parameters, k_p, illustrated in Table 2, both as calculated by average of terminal cases, and as the asymptotic of the fit curves, showing the consistent increase as a function of P_{thresh}.

We can use this data to confirm the predicted relationship suggested in Equations 13 and 16. In particular, if we take k_p to be the convergent performance of the 0% error case (considered reasonable on account of the problem simplicity and closeness of the measured value of 18 steps to the theoretical minimum case of 17 steps) we can examine the ratio of $k_{p\alpha}$ at multiple error levels to this baseline k_P. Plotting the ratio $\frac{k_{p\alpha}}{k_p}$ in Figure 11, the

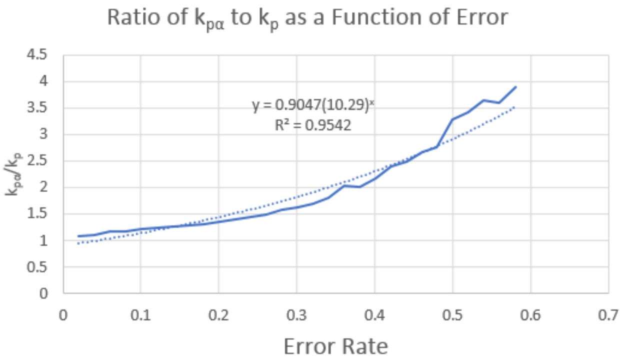


Figure 11: Plot of $\frac{k_{p\alpha}}{k_p}$ ratio for the learning curves of the GAP algorithm on the STRIPS world, along with the power law fit curve

proportional change between the asymptotic performance of the induced error case and the error free case. When we examine this ratio, we find that there is a high correlation exponential relationship between the error rate and the terminal performance of the agent, mirroring the predicted relationship.

The key observations across this entire set of results is that the GAP agent consistently learns the problem structure, even high degrees of artificially introduced error to the problem. Additionally, the learning curves closely follow the predicted forms in Section 4, and derived measures from the analytics in that section are also in close adherence to those measured within the problem space, validating these predictions for the STRIPS problem, and enabling us to investigate further on more complex problems in the next section with the knowledge that the learning performance is as expected.

5.3 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. We additionally complicate the problem by performing the navigation component inside a maze. These mazes can be well conditioned mazes, such as those lacking open fields which confound wall following algorithms, and ill–conditioned ones which possess such areas. 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, concurrent with ample opportunity for error induction and abstraction onto the learning case.

For an agent, actions are movements in each of the four cardinal directions, and pickup and drop off actions which can only be performed at explicit locations within the maze. Inputs to the system naturally vary depending on the abstraction mechanisms being employed, but in general include local observations of the maze topography in some region near to the agent, registration of the relative position of the target 'passenger', and whether a passenger is currently carried.

Of particular note for our experiments here (speaking to the goal agnosticism of the GAP algorithm) is that 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. The inclusion of both TAXI elements and Maze elements allows us to better explore the functioning of the GAP algorithm in complex hierarchical problem spaces. It is possible to see that, taxonomically, a combination of navigation and sequence ordering elements such as this is, essentially, an expanded case of the STRIPS type problem. By increasing this complexity, we are able to examine higher order behavior as an extension of the observations made in the prior experiments.

5.3.1 Simple Maze/TAXI Domain

To begin with, we examine performance in a relatively simple field of activity, where the 'maze' component is more akin to obstacles, as illustrated in Figure 12. These are areas of 15x15 cells with some number of randomly placed 'wall' cells, and three target pickup and dropoff locations. This domain thus has $|\mathcal{S}|_{max} = 1822$. Agent actions include cardinal direction movements, pickup and dropoff actions, and the states are constructed from the

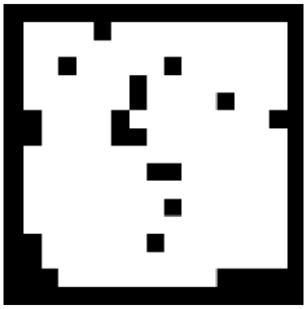


Figure 12: Example TAXI Domain world of the type used in these experiments, with sparse obstacles placed (covering 10% of the working area) and with a regional spacing of 15x15 cells

location of the agent, whether the agent currently has a passenger, whether the agent is at a passenger target, destination, or neither, and the number of remaining passengers.

This simple domain allows us first to validate that the agent will successfully learn in this sort of hierarchy with substantially more states, paving the way for the study of the more complex forms of the problem. It also presents a chance to investigate the impacts of input abstraction on learning in the absence of artificial error. State abstractions would be possible in the STRIPS-type problem, however the state space is so small that proportional changes to $|\alpha_{Tg}|$ are, in practice, difficult to compare numerically.

Towards that end, we explore three additional state construction mechanisms: reduction of both location coordinates by a factor of 2 (essentially reducing the navigation to 2x2 blocks in the greater world). Reduction of just the horizontal position coordinate by a factor of 3, and reduction of both coordinates by $1\frac{1}{2}$. Each of these abstractions reduces the state space size by compressing the navigation space, at the cost of mixing cells which are mapped together into the same probability space.

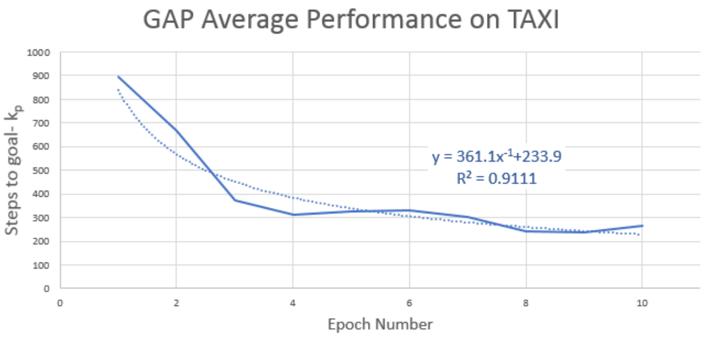


Figure 13: Average performance of the GAP algorithm learning the simple implementation of the Maze/TAXI joint domain problem on a 15x15 'maze' with three randomly placed target pickups and drop offs

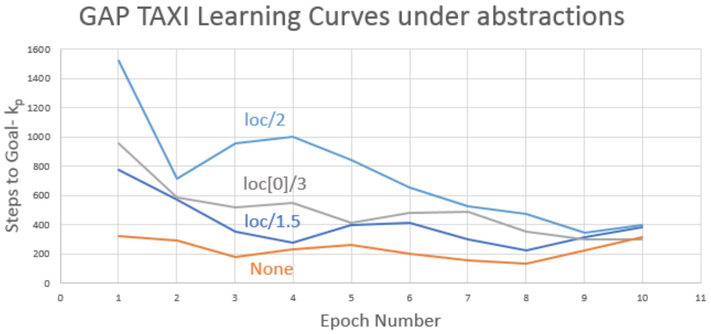


Figure 14: GAP learning curves using location–based state space compressing abstractions

We first plot the learning curve over the aggregate of all experiments (numbering 50 per abstraction, for 200 total trials) in Figure 13, to reinforce that the aggregate performance of the agent in this world matches the predicted learning behavior. As before, we see a reciprocal fit with a high correlation of $R^2 = 0.91$, better matched than that of the STRIPS case. The asymptotic ideal performance case is 234 steps across all trials, with the measured peak no–abstraction performance being 222 steps. This observation alone is interesting, as it suggests that the GAP algorithm is, among these abstractions, able to bring the performance curves for the abstracted state spaces to within about 5% of the asymptotic non-abstracted performance level. This implies that the $Q(\alpha_T)$ factor for these abstractions is substantially better than for the pure random errors; a qualitative argument for the efficacy of the abstractions themselves, and thus their utility as tools for examining system behavior distinct from induced error.

Figure 14 presents the learning curves acting on each of the abstractions independently. Though the asymptotic performance limits are comparable, the long term behavior of the curves themselves are substantially varied, with a direct relationship between the effectiveness of the abstractions in learning performance; the 'loc/2' abstraction having the most negative impact, followed in order by 'loc[0]/3', and then 'loc/1.5', with the effective number of initial steps to the goal at the first epoch being directly correlated to learning performance throughout the trials, a trend observable in the STRIPS-style problem, and one which we will continue to see in latter experiments.

Abst.	k_p Meas:	k_p Pred:	%E
loc/1.5	307.1	245.1	20.2%
loc/2	405.5	421.5	3.9%
loc[0]/3	317.1	304.5	3.9%
None	222.7	192.9	13.4%

Table 3: Measured and model–predicted parameter comparison for the GAP algorithm learning the simple Maze/TAXI Domain problem

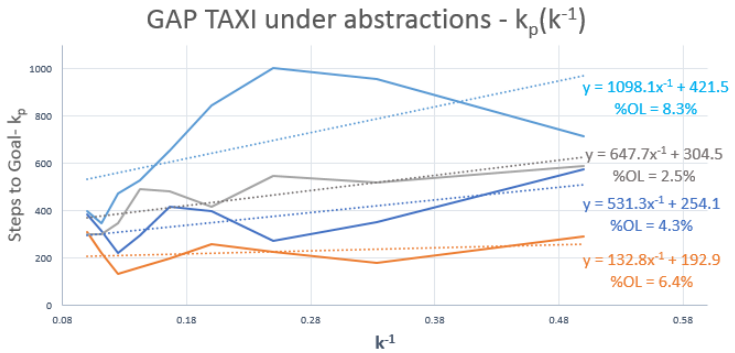


Figure 15: Plot of k_p as a function of $\frac{1}{k}$, along with corresponding linear fits and percentage of off–linear components

Table 3 presents the measured and predicted k_p for this problem with each of the state abstraction cases, along with the corresponding errors. In this data, we find a curious relationship between the scale of the error and the abstractions: the no abstraction case and 'loc/1.5' case, which have the best pair of asymptotic learning performance levels, also have the highest proportional discrepancy. On the one hand, the superior performance means that the impact of differences is magnified, however the scale of the errors, 13% and 20%, is unlikely to be accounted for by this alone.

We may, however, gain some additional insight by exploring the fit curves themselves, plotted on Figure 15. Here, we can see that the off–linear errors for each curve are relatively low, all less than 10%, indicating solid close–to–form adherence to the model of Equation 18. However, the errors associated with the no abstraction and 'loc/1.5' abstractions are approximately twice that of the other pair. Given that these two are the higher performing instances, we may thus hypothesize that the error discrepancy is likely due to fit errors associated with the agent reaching asymptotic performance levels earlier than the slower learning cases. When we make this assumption, and truncate the cure fit at 5 epochs, rather than 10, we find k_p of 288.7 and 205.5 for the 'loc/1.5' and no–abstraction cases, respectively, with corresponding errors of 5.9% and 7.7%, comparable to the performance of the other pair of trials. These results show that, as with error induction, it is possible for the algorithm to learn effectively in states spaces which are mixed and of reduced size under abstraction perturbation.

5.3.2 Complex Maze/TAXI Domain

Building on the validation of the capability of learning under abstracted state spaces in the prior subsection, we now turn to investigating the substantially more complex working space of a full maze. Such a maze is illustrated in Figure 16, which highlights a few particular complications we have produced: rather than restricting ourselves to simple mazes without interior spaces, we have allowed for the inclusion of open space regions in the maze. Those mazes with uniform width traversals present a set of inherent relationships which make them amenable to solution by simple form maze navigation algorithms. This indicates

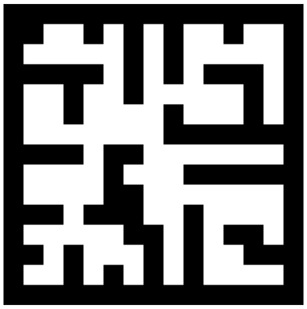


Figure 16: An example of a randomly generated ill–conditioned maze used in these Maze/TAXI problems

the existence of an inherent state space simplification embedded in the structure of the problem. We remove this constraint from the maze generation algorithm to deliberately increase problem complexity, and therefor the sensitivity of our experiments to impacts of varying error and abstraction.

We also elect to represent the state vector as a relative measure. In the most fundamental case, we implement the state model based on the local environment phrased as available movements, a relative vector towards the next objective, and whether or not the agent currently has a 'passenger'. In implementing relativistic states such as this, it becomes possible to learn the problem in a more general sense than a specific sequence of fixed tasks. This generalized formulation then allows us to examine properties of learning transference and generalization, especially valuable because as in the prior case, each *epoch* is trained in a different maze, with new, random objective locations. As a result, the maximal state space size is variable, however for the maze generation parameters used, averages to $|\mathcal{S}|_{max} = 18432$.

Figure 17 shows the average learning curves for the complex Maze/TAXI problem across trials with error ranging from 0% to 30%. With an R^2 of 0.84, less than the previous

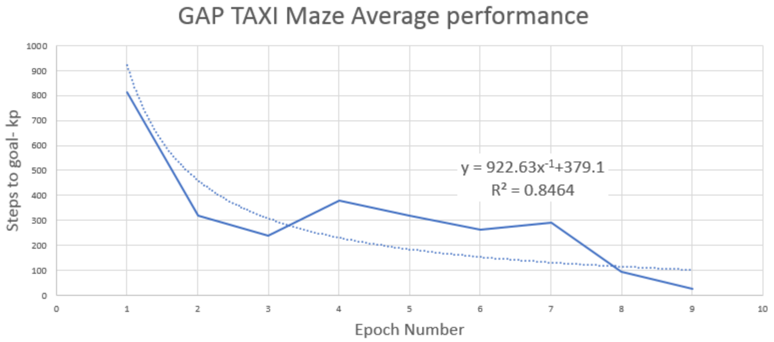


Figure 17: Average learning curve for the GAP algorithm operating on the complex Maze/TAXI problem

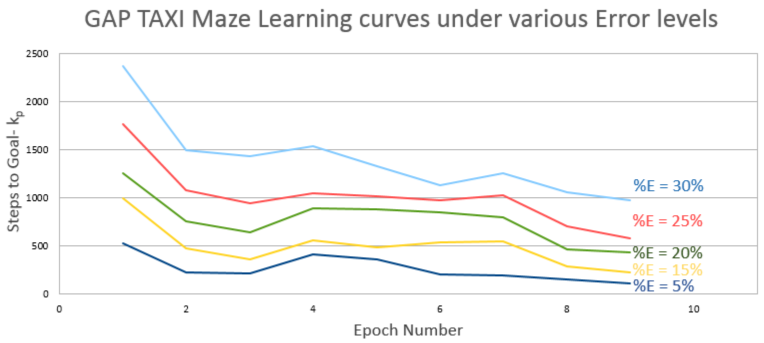


Figure 18: Performance of the GAP algorithm across multiple levels of induced error on the Maze/TAXI problem space

Maze/TAXI case, but still greater than the STRIPS trials. A primary contributor to the error in fit quality is the presence of outlier learning cases within the data, with the scale of these rare disturbances visible on Figure 17 as the sharp jump from epochs 4 to 7. This can be readily observed to be due to the randomization of the maze occasionally presenting radically different structures from those previously learned. The agent must then adapt to an expanded problem space. The overall trend towards the asymptote, however, is clear evidence that in the face of this adaptation challenge the GAP algorithm is able to learn the more expansive problem after a few epochs. Further, of particular note is that the amortized scale of the disturbance is much lower than that of the initial performance prior to any learning. This indicates that a measure of learning transference is necessarily occurring- a wholly novel circumstance would demonstrate performance at a level comparable to the first epoch.

In Figure 18, we plot the curves for the GAP algorithm learning the Maze/TAXI problem across levels of induced error ranging from 5% to 30%. Present here are two previously noted trends: the relationship between the asymptotic k_p's proportionality to the error rate, and the correlation between initial performance and long term performance across errors, as well as the presence of 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.

We have, however, observed this range varying for individual cases 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. In trials investigating long run trends (extending to 100 epochs), we observed that the average case over each error level achieved asymptotic performance by 9 epochs, with no statistical outlier cases of sudden change in performance level ever occurring after 17 epochs across 1000 instances of training.

Additionally, establishing continued adherence to our expected learning curve is necessary in preparation for more sophisticated analysis. To demonstrate this, we have again plotted the relationship between $k_{p\alpha}$ and $\frac{1}{k}$, in Figure 19. We see low levels of off–linear error for each curve, with the scale of these errors being roughly inversely proportional to the

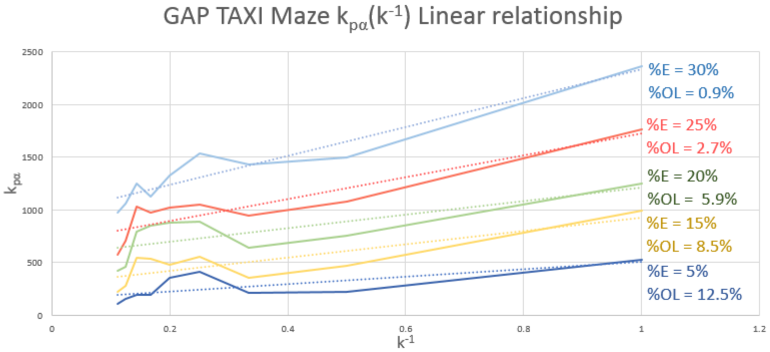


Figure 19: Plot of k_p as a function of $\frac{1}{k}$, and the measures of non-linear divergence for the Maze/TAXI trials at each induced error level.

induced error. This is a manifestation of the same phenomena described in the prior section whereby early convergence skews the fit equation, in this particular case the variation being due to error rather than abstraction.

We have here explored the performance over error regimes, with some interesting hints as to the capacity for learning transference, but our primary goal is further investigation of the impact of abstractions on the GAP algorithm's learning. We will be using the method described in Section 5.1.4, based on error thresholds, to identify L_{max} for a group of abstractions used *in combination.*

For this additional battery of experiments, we apply three different kinds of abstractions to the state. These naturally represent reductions in the total expressiveness for the agent's learning system, but when adequate to enable valid solutions, also offer the chance for more rapid convergence of learning. We refer to the three as AI, AII, and wA:

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.

We produce four different state generation methods with these: '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 joining the different models in this way, we can compare the relative impact of each different abstraction, in accordance with the form of Equation 14.

Figure 20 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 directly that the learning curves of 'AII wA' and 'AII wA' are very similar, with 'AII w/oA' being less effective, but in the same general range, and finally that 'AI w/oA' performing substantially worse, with convergent behavior an order of magnitude worse than the other three. Taken together, this highlights the critical sensitivity of performance to $||T_{\alpha k}||_1$ demonstrated in Equation 16 and discussed in Section 5.2, with the inclusion or non-inclusion of the 'wA' abstraction alone being responsible for a 13-fold reduction in asymptotic performance.

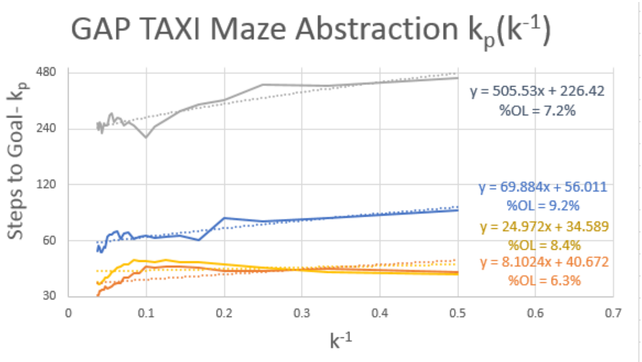


Figure 20: Linearized plot of the learning curves for the Maze/TAXI learning under abstraction with corresponding measures of off–linear performance for each of the four combinations of abstractions implemented

We also use the profile of each joint abstraction across error levels to measure the same indicators of correlation to the model as before, the first set being the measured and predicted k_p laid out on Table 4. On this table, we have each abstraction at each of the error levels sampled, and the average percent error across abstractions. For 'AI wA', 'AII wA', and 'AII w/oA' we have relatively low errors at 4-8%, with 'AI w/oA' being an outlier at 14.5% error. In this case, we discover that this is the reverse of the 'early convergence' phenomena, whereby extending the runs past the established 10 epoch threshold, we find that the 'AI w/oA' trials were not fully converged until approximately 15 epochs. Though they are then within about 5% of the value at 10 epochs, this is enough to skew the fit.

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

	L_{max} Pred	L_{max} Meas	%E
AI wA	61.7	69.1	10.6%
AII wA	32.2	36.8	12.5%
AI w/oA	527.1	573.1	8.0%
AII w/oA	40.4	43.8	8.3%

Table 5: Comparison of measured and predicted L_{max} across abstractions for the complex Maze/TAXI domain with joint abstractions

Table 5 additionally presents the calculated values for L_{max}. 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 4.

In addition to these observations, we can also make some substantial insights through the use of induced error in addition to the abstractions, allowing us to calculate estimates of several system parameters by measurement of the proportional effects achieved by the various combinations of abstractions. Because a joint P_α can be constructed by multiplying together the constituent abstractions comprising it, and these can be observed in their effect via Equation 14, we can use the discovery of k_p and L_{max}, along with the fit functions for $k_{p\alpha}$ as a function of $log(1 - P_{thresh})$, to estimate $||\alpha^+_{Ts}||_1 \cdot ||T_{\alpha k}||_1 \cdot ||\alpha_{Ts}||_1$.

On Figure 21, we have plotted $k_{p\alpha}$ as a function of $log(1 - P_{thresh})$, in order to establish that the correct behavior of Equation 14 exists for the succeeding analysis. It is possible to see here that while the error rates for these plots are higher than we have been seeing for the actual learning curves, especially for 'AI w/oA', there is still a qualitatively visible linear pattern, from which reasonable approximations may be made.

Given this form, we can then calculate the remaining values in Equation 14 from the curves in Figure 21, which yields Table 6, containing the estimated values of the L1 norm of

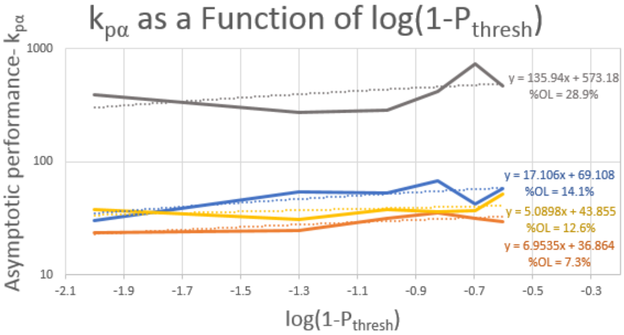


Figure 21: Convergent behavior of the four abstractions as a function of $log(1 - P_{thresh})$, as in Eq. 14, along with the percent off–linear deviations for each curve, corroborating the use of Eq. 14 as a proxy for calculating the remaining components of $|\alpha^+ T\alpha|$

	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%)	

Table 6: Measured $k_{p\alpha}$ and corresponding $|\alpha^{+}T\alpha|$ estimates

the abstracted transition array (denoted thereupon as $|\alpha^{+}T\alpha|$ for compactness). Because we can use the variant $k_{p\alpha}$ and P_{thresh} at each induced error level alongside the generally derived L_{max}, we are able to produce an estimate for $|\alpha^{+}T\alpha|$ at each error level. These show a relatively small variance across error level, as expected given that the abstraction matrices are constant, though there is some statistical variance due in part to the stochastic nature of the experiment trials, and likely also approximation errors evolving from the fit functions of Figure 21.

With the values calculated across error levels for $|\alpha^{+}T\alpha|$ being in close correspondence, we can conclude that we have successfully estimated this parameter. However, it is by nature the estimate for the total abstraction across the mapping from the 'measured' state to the 'true' state, and may contain other factors not due to the specifically controlled mappings, such as other components of the state vector (like the effect of the 'carries passenger' component) or implicit features of the agent's workspace (of the same type as the implicit abstraction in assuming a well conditioned maze, discussed earlier).

Because we elected to combine the abstractions as groupings of subsets of two, though, we can make an estimate of the impact in transitioning from one of the paired subsets to the other, relying on the submultiplicity of the L1 norm. By taking the ratios of the pairs' measures, we can approximate the impact of each transition from 'AI' to 'AII', both in the 'wA' and the 'w/oA' cases, and compare these, and vice versa for 'wA' and 'w/oA' across 'AI' and 'AII' both.

On Table 7, we have these values, and of note is the level of correspondence between the independent transition ratios. When presuming 'AI', changing from 'wA' to 'w/oA' effects a 0.877 factor shift in the L1 norm of the abstracted transition array, and in the 'AII' case, a 0.798 scale change. The transition from 'AI' to 'AII' measures as a 1.091 factor for 'wA', and 0.992 for 'w/oA'. Both sets present extremely similar scale changes,

Q(α)	AI	AII	$I \rightarrow II$		AIwA$\rightarrow$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 $\rightarrow$ w/oA	0.877	0.798		Pred 2:	0.791 (-9%)

Table 7: Calculated $|\alpha^{+}\alpha|$ ratios across abstractions and predicted transform measure, derived from the entries in Table 6 and Eq. 14

suggesting strongly that they are very close to the actual impacts predicted by Equation 14. It is difficult, however, to judge the scale of these deviations, as a ground truth measure is not directly available. Instead, we can get a sense of the net comparison by independently combining the two transitions in the two combinations 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 first estimate to be 0.958, and the second 0.791; respectively 10% and 9% off of the actual ratio of 0.871, which puts the relative accuracy into proportion.

5.4 Tower of Hanoi Domain

The Tower of Hanoi puzzle is a perennial favorite problem class for mathematical analysis. Conceptually simple, the puzzle consists in the most basic form of a number of disks and three or more pegs on which these disks may be stacked, with the disks labeled by an ordinal index which must be preserved when disks are transferred between pegs. The canonical implementation has 3 pegs, and typically from 3 to 7 disks, but both categories can be expanded to change aspects of the problem case. Expanded problems are usually represented as $ToH_{p,d}$, where p is the number of pegs, and d is the number of disks.

This problem provides a wide range of benefits for experimentation and demonstration with regards to exploring properties of a problem solving agent. It is a well defined problem which is straightforward to simulate and solve, with the mathematics of the 3-peg case being particularly well studied. This allows for direct performance bounding, with greater numbers of pegs presenting wider state spaces with lower net complexity, and increased numbers of disks representing increases in the depth of complexity.

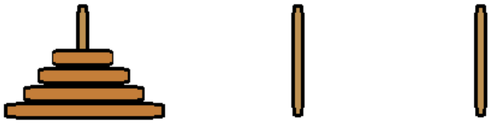


Figure 22: Illustration of a traditional Tower of Hanoi (ToH) problem: the objective is to move all disks from the first peg to the third, by only moving disks between pegs, and under the constraint that a disk may only be moved on top of a larger disk or to an empty peg. This graphic shows the 3-peg, 4-disk, variant of the problem, $ToH_{3,4}$

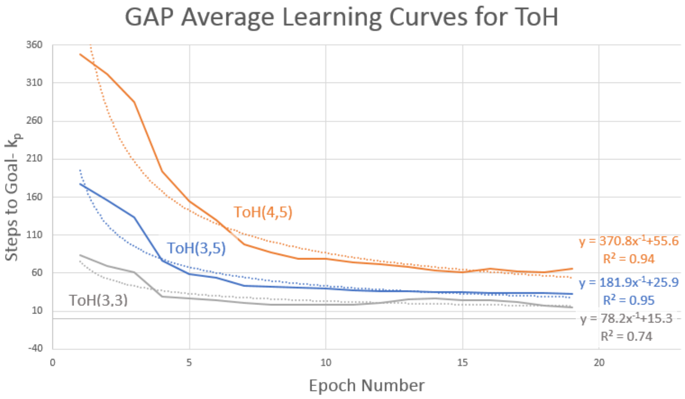


Figure 23: Average learning curves for the GAP algorithm over the three investigated ToH domains, $ToH_{3,3}$, $ToH_{3,5}$, and $ToH_{4,5}$ at varying error levels, along with reciprocal fit curves

For our purposes, there is also the added benefit of the state space and action space complexity. The scope of the state space is around $|\mathcal{S}|_{max} = p^d$, depending on whether pegs are considered interchangeable, and the action space at $|\mathcal{A}|_{max} = p^2$. Naturally, in this particular problem only a small subset of the space forms a reachable neighbourhood from any other state, so the graph complexity is lower than the upper bound, but this still represents a substantial space to explore. Further, by contrast to the prior cases which had fairly small action sets regardless of state space size, the Tower of Hanoi problem allows for more expansive action spaces by increasing the number of pegs.

Here, we investigate three instances of the problem, in order to explore the effectiveness of learning across multiple complexity classes. Figure 23 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. We also investigate the performance of the GAP algorithm under changing error levels. Table 8 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.

Perhaps more interesting, we can see that the $ToH_{4,5}$ levels deviate substantially with error, suggesting that the expanded state space is more vulnerable to impacts of error (a property we will interrogate further shortly). In addition to the convergent performance values, 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 as with several of the previous low error cases, and indeed the $ToH_{3,3}$ case converges at approximately 5 epochs rather than the 20 sampled.

Because the Tower of Hanoi is such a well studied problem, optimal algorithmic solutions are available from (Van Zanten, 1990), and we can therefor compare the asymptotic performance of the agent to the theoretical optima. For the 3 peg cases, the optimal number of

	P_{thresh}	k_p	$k_{p\alpha}$	%E	%OL
ToH(3,3)	5%	16.5	15.6	5.5%	19.3%
	15%	47.3	47.8	0.9%	7.8%
	20%	61.9	63.4	2.4%	1.8%
ToH(3,5)	5%	32.7	30.1	11.4%	5.4%
	15%	34.8	37.1	6.4%	5.8%
	20%	44.9	48.7	8.2%	16.1%
ToH(4,5)	5%	206.4	201.5	2.4%	12.3%
	15%	696.7	707.5	1.6%	2.3%
	20%	2052.2	2006.9	3.0%	1.2%

Table 8: Chart of the correlation measures for the GAP Algorithm learning the Tower of Hanoi problem, across error level and problem complexity class

moves is $2^d - 1$, giving 7 moves for 3 disks and 31 moves for 5 disks. The convergent behavior for the agents on these cases with random error are 15 and 33 steps respectively across the full error range, with the 0% error cases naturally achieving the optimal performance level after one epoch. One observation we can make across the test batteries is that both the scale factor A and the asymptotes k_p of the learning curves increase by approximately a factor of 2, meaning that the learning behavior is essentially identical, but delayed and elevated for larger state/action spaces, matching the optimal solution as a power function of 2.

5.4.1 Tower of Hanoi under Abstractions

We can also use the substantial state space, evolving out of a restricted set of elements (the disks, pegs, and rules for movements) for further analysis. The Tower of Hanoi state space is readily amenable to state abstraction, of which we develop and implement four, referred to as 'AI', 'AII', 'AIII', and 'AIV'.

These abstractions are:

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;

For example, for the 3-peg, 5-disk problem, if the current state were represented in full as $\{[1,3],[2],[4,5]\}$, AI would produce '25'; AII would give '[4,2,9]'; AIII yields '[(2,1),(1,2),(4,2)]'; and finally AIV has '[2,1,2]'. These abstractions produce incrementally compressed state spaces as measured by the number of total aliased states, with AI being nearly equivalent to the full state space, and AIV being massively reduced.

Of note, in addition to these four state compression models, we also attempted trials with one additional abstraction: a reduced form of AIII including only the size of the topmost disk. However, in this case learning convergence failed wholesale, with the 'asymptotic'

	Abst.	k_p	$k_{p\alpha}$	%E	L_{max}	L'_{max}	%E
$ToH_{3,3}$	AI	16.5	15.6	5.6%	15.6	17.5	11.4%
	AII	27.8	31.5	13.5%	8.1	8.8	7.3%
	AIII	21.6	17.3	20.2%	17.22	14.8	16.3%
	AIV	17.0	15.3	9.7%	31.5	35.1	10.3%
$ToH_{3,5}$	AI	35.3	34.2	3.1%	62.1	59.4	4.7%
	AII	31.4	35.1	11.8%	64.9	69.0	5.9%
	AIII	31.0	31.0	0%	N/A	N/A	N/A
	AIV	31.0	31.0	0%	N/A	N/A	N/A
$ToH_{4,5}$	AI	254.5	256.9	0.9%	112.06	104.5	6.8%
	AII	278.1	241.3	13.2%	101.9	112.6	10.5%
	AIII	1267.2	1354.6	6.9%	391.9	371.5	5.2%
	AIV	2017.7	1843.2	8.6%	719.9	749.8	4.1%

Table 9: k_p and L_{max} comparisons for the GAP algorithm learning the ToH problem with various abstractions and across complexity classes, note that for ToH(3,5), AIII and AIV consistently converged to the optimum after one epoch, preventing fit curves being derived to calculate L_{max}

performance invariably being within a single standard deviation of the initial performance, indicating essentially random action similar to the observed situation when increasing the induced error rate to extreme levels in the Maze/TAXI domain.

Table 9 presents the essential measurements for the entire battery of experiments, spanning $ToH_{3,3}$, $ToH_{3,5}$, and $ToH_{4,5}$; all four abstractions across error rates from 0 to 20%. A few features of this aggregate data are immediately notable: first, we observe that 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 $ToH_{4,5}$.

Additionally, on Table 10 are presented the coefficients for the reciprocal fit $(A\frac{1}{k} + k_p)$ of each trial set for each of the ToH cases, and the corresponding off–linear error associated with each such curve. We can see that the curves match the predicted form to comparable levels as in the previous problems, with errors on the order of 2-8%.

	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%

Table 10: 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.

One particular comparison to be made between between this pair of tables is in the $ToH_{4,5}$ cases for AIII and AIV. For AIII and AIV, the convergent k_p are substantially larger than those for the AI and AII cases on this same problem. We may hypothesize that this could be due to insufficient numbers of epochs explored to effect comprehensive learning, however the corresponding reciprocal fit curves have errors of only 7.5% and 1.6%, well on par with other experiments not presenting immature learning. Further, we can note that the average, no–abstraction $ToH_{4,5}$ curve has convergent behavior at about 56 steps to goal, as evidence that the $ToH_{4,5}$ problem is not inherently resilient to the GAP algorithm's learning, and thus we conclude that these abstractions are therefore ill–conditioned to the problem case.

We can apply similar rationale to the AI and AII cases, which have terminal behavior substantially worse than the non-abstracted case, unlike the counterparts in the $ToH_{3,3}$ and $ToH_{3,5}$ cases, both of which are well within range of the average case performance. Because all of the abstraction trials using $ToH_{4,5}$ perform substantially worse than the non-abstracted case, yet the other problem classes perform similarly, we my hypothesize that the cause of the discrepancy is the unsuitability of the AI-AIV models to represent the $ToH_{4,5}$ problem space, specifically.

Indeed, if we examine AI in particular, we may note that by increasing the number of pegs, we have created a case in which the number of abstracted states remains nearly constant (as a function of the disk placements) yet the number of real states has increased exponentially. In a general sense, we can consider this ratio of abstracted to real states as a metric component which puts an upper bound on $||T_{\alpha k}||_1$, and by extension $Q(\alpha_T)$. 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.

One way we can see this effect in action is by using error rate as a proxy for effectiveness on performance. In Figure 24 (left) are plotted the linearized curves for $ToH_{3,3}$ across induced error rates 5, 15, and 20%. Here, we see that the k_p associated with the abstractions congregate around the 5% and 15% error levels, and with a ratio of 3.8x between the 20%

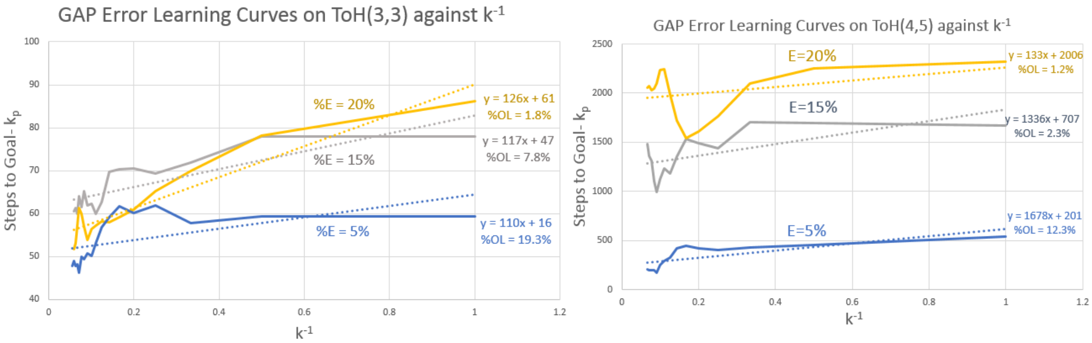


Figure 24: Plots of the linearized GAP learning curves for the $ToH_{3,3}$ (left) and $ToH_{4,5}$ (right) problems across multiple levels of induced error and associated off–linear error measures

and 5% k_p. In Figure 24 (right), by contrast, are the error impacts on the $ToH_{4,5}$ case. Here, we can first see that there is a substantially higher susceptibility to induced error for this problem case, with the proportional change between 5 and 20% being a factor of 10x. In this case, we can see that 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.

By these comparisons, we can see the ways in which the relative size of an abstracted state space impacts overall performance, producing similar results to that of changing error level, as predicted by the existence of the $Q(\alpha_T)$ metric. The implication is that, as we can measure overall performance in terms of k_p, and there is a direct relationship between $||T_\alpha||_1$ and $k_{p\alpha}$, the ratio of the sizes of the native and abstracted state spaces has a substantial limiting impact on the overall performance of the GAP algorithm.

6. Conclusions and Future Work

In this paper, we have presented a hypergraph–based learning and planning algorithm, the Goal Agnostic Planner, GAP, designed to learn 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 are used to retain information pertaining to occasions, or state-to-state transitions precipitated by actions. 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.

The additional array/linked lists augmenting the primary 3–dimensional array are used to model 2-dimensional slices of the hypergraph, a space complexity bounded by $O(n^3)$. In Section 4.1, we proved that these slices contained the path through the state space 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.

We then used this information to construct a model of dynamic agent performance by converting maximal probability trees associated with the hypergraph slices into a transition matrix for analysis by Markov chain methods. In Section 4.2.1 we used this model to predict the time evolution of the GAP agent, and showed that the probability of goal achievement was monotone in time. In Section 4.2.2 we derived a metric describing the probability that the agent becomes unable to transition to the goal, fully describing the agents' convergence properties. In Section 4.2.3, we demonstrated that the convergence rate is bounded above by an exponential function, illustrating that the transition to the terminal states is efficient.

Based on these relationships, in Section 4.3 we were able to introduce a model for disturbed models as a transform between two transition arrays. Using this transform, we are able to derive the conditions under which a path planned in an abstracted or uncertain space will also be a valid path in the true state space. We also, in Section 4.4, analyzed the impact of the abstraction on system performance, and derived a metric for describing the 'quality' of an altered model in terms of this performance change, parametrizing the change in speed of convergence due to a perturbation on the state space.

We then used a specialized one-to-one transform to model incremental learning as a state space perturbation in Section 4.5, showing that this transform approaches the identity as learning progresses, proving that GAP agents will demonstrate progressive learning. Additionally, from the successive analysis of the transition array's change over time in Section 4.5.1, we were able to determine that learning curves for GAP agents will, on average, follow a reciprocal trend, showing that learning rates will be tractable.

To investigate the performance of the GAP algorithm on actual problem cases, we performed trials on three problem cases in Section 5: a fixed environment procedural task based on traditional STRIPS problems in 5.2; variable environment hierarchical problems constructed by combining maze navigation and the TAXI domain in 5.3; and the Tower of Hanoi puzzle under multiple configurations in 5.4. 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.

To validate the results of our analysis, we made proxy measurements derived from the best fit learning curves to compare to values derived from distinct results of our analysis. We further applied varying levels of artificial error in each experiment case to study the impact of disturbances on learning performance, showcasing convergent behavior in the face of these disturbances. 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.

Aside from these broad investigations, we also used individual experiments to explore specific properties related to the GAP algorithm. We use the STRIPS problem case to study the power law relationships predicted for convergent performance under varying uncertainty. In the Maze/TAXI experiments, we implemented a set of abstractions which could be applied in tandem, using the derived performance relationships to confirm theoretical predictions about performance under composed transforms. In the Tower of Hanoi experiments, we used abstractions with a range of levels of state space size reduction to investigate the properties of the abstraction quality metric, comparing it to performance changes induced by artificial error.

6.1 Limitations

In this section, we discuss some of the outstanding limitations associated with implementation of the GAP algorithm.

6.1.1 Memory Use of INC Array

Perhaps the most substantial hurdle is the size of the INC array, which scales as $O(|\mathcal{S}| \times |\mathcal{S}| \times |\mathcal{A}|)$. While similar to problems of combinatorial explosion present in many planning systems, the growth of the array itself is confined to polynomial order, with the addition of hierarchical domains (rather than additional states) representing the primary manifestation of the issue.

For instance, suppose, for 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 (Jiménez et al., 2012), 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 a potential approach for implementing traditional methods of compression in a stable, well-defined manner. Further, many hierarchical problems are likely to have highly sparse probability spaces. Unlike the traditional 'curse of dimensionality', for GAP agents, graph–based planning means that sparsity can be leveraged for improved performance. One might then represent INC efficiently using an alternative graph–like structure which does not retain empty regions of the array. For instance, constructing a hypergraph object from $|\mathcal{S}|$–dimensional link objects, labeled with a hash function of the input (s_i, s_f, a_l) coordinates.

Additionally, alternative methods of representing INC which can reduce the size of the array are discussed in Appendices A.3 and A.4.

6.1.2 Speed of Dijkstra's Algorithm

In our application cases, we have used a naive implementation of Dijkstra's algorithm which operates on the worst–case time order of $O(|\mathcal{S}|^2)$. While this is not substantially weak computational performance, it is in current form designed mainly to interact in a natural, easily understood way with the INC and AFI datastructures. However, more advanced, faster path finding algorithms can be applied to improve overall performance.

Given that the maintenance of the maximal probability subgraphs is independent of the path–planning algorithms operating on it, any algorithm compatible with optimization of path costs on a graph would be viable for planning, the most immediate example of such being the A* algorithm. 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.

6.1.3 Familiarization Phase

For any problem, the algorithm proceeds through a phase associated with primarily random actions for exploration of the state space, using the initialized random actions in the untrained AFI array. One may even deliberately program a phase of execution with entirely random actions, rather than actions planned in the randomly populated state space for the purpose of saving computation time until a suitably large number of states are known reachable. However, even if this is not the case, systems will have a period of mostly random exploration until some paths to the goal have been identified. While this is a feature of most learning systems, it is not an ideal situation, and patterns randomly expressed during this phase may affect later learning.

Approaches to examining the impact of, and amelioration for, this phenomena are discussed in Appendices A.1, A.2, and A.3. Additionally, a method which we explored in pilot experiments for the Maze/TAXI domain and the Tower of Hanoi domain is the use of Tabu search in the exploration phase. In these initial experiments, we implemented a policy by which the selection of exploration actions was guided by balancing previously explored actions. That is, rather than taking an entirely random action when the goal state had not yet been discovered, each time a state was visited the selection of action was made among the heretofore least–chosen actions for that state, ensuring that each state was uniformly explored during the familiarization phase.

6.2 Further Work

Though the problem cases explored herein present ample evidence pertaining to the proposed properties of the GAP algorithm, there are a selection of topics which are not directly addressed by the current experiments that bear substantive further examination.

Firstly, 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. Though we can confirm that trap nets do not present a barrier to operation in these problems, we are unable to directly confirm our predictions pertaining to them with the results of experiments that lack them.

Similarly, though the concept of multiple attractor states (including multiple concurrent, equally valid, goal states) is discussed in Section 4.2.1, all systems demonstrated thus far possess singular goal states. As such, the impact on performance curves due to multiple target states is not addressed. Two minimally studied cases in Appendix B.1 and B.2 do illustrate effective learning of problems with multiple goals, though.

Additionally, in the presented experiments, we have explored non-deterministic action of the GAP algorithm vis-a-vis error induction, as well as by the exploration of abstractions which reduce the size of the state space (causing certain states to be mixed in the learning structure). Such non-determinism is not strictly uniform across the state/action space as it is random across actions, and therefor creates asymmetric distributions depending on the possible outcomes of those actions. However, none of the problem cases presented in the set of validation experiments above is *inherently* non-deterministic. Though the combination of abstractions with error is likely to produce systems which appear to the agent to be highly stochastic, direct investigation of non–deterministic systems is still needed to establish performance of the GAP algorithm in the context of inherent uncertainty, rather than uncertainty modeled as a perturbation or a transform of a deterministic state space.

A further complex issue is learning transference. In the randomized worlds used in the Maze/TAXI experiments using relativistic states, we have provided some evidence of transferred learning. The changing maze structure presents a variant problem of the same type and complexity, but with diverging state spaces. However, we have not constructed an explicit model describing performance under these conditions beyond considering the adapted learning as a special case of the general form for learning. This condition was alluded to in discussion (within Section 5.3.2) pertaining to the 'adaptation bumps' observed during Maze/TAXI learning. However, we address it only in the context of the fundamental reciprocal model, not an explicit adaptation model. 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.

One final concept which we wish to investigate further is unsupervised operation in the context of the GAP algorithm. Though the actual learning process for the GAP method can be considered inherently unsupervised, the planning phase itself is not. 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 the path planning algorithm to terminate when identifying any destination state meeting some conditions is something of a hybrid model. The potential for learning cost improvements opens the possibility of both unsupervised planning and learning as well as process optimization.

Appendix A. GAP Algorithm Modifications

Aside from the direct investigations in the paper proper, the structure of the GAP algorithm presents itself naturally to a set of modifications, all of which may present various advantages. We have experimented with each of these in our test cases, but none in sufficient detail as to warrant such expansion of the scope of this paper without more thorough analytic investigation.

A.1 Implicit learning rate

Though the GAP algorithm does not incorporate an explicit learning rate parameter, we have shown that there is learning convergence which will follow a reciprocal function. This function was intimately tied to the number of recorded observations of each occasion. In the prior sections, we examined this process as an incremental change to a transform on the transition array during learning, but it could also be viewed as an averaging function over the number of samples.

For example, consider the impact of one fluke observation at different times. Presume that we have observed a state-to-state 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 thus far, but the tenth observation is precipitated by a_2; the associated a posteriori probabilties, then were previously:

$$P(a_1(s_i) \rightarrow s_j) = 1.0; P(a_2(s_i) \rightarrow s_j) = 0.0$$

whereas after, they are:

$$P(a_1(s_i) \rightarrow s_j) = 0.9; P(a_2(s_i) \rightarrow s_j) = 0.1$$

a relative shift of 10%. If the total observations, however, were 99 prior to the anomalous result, then the probability shift would be only 1%. This tidily illustrates the way in which learning is tied to the reciprocal function, and how this function includes learning inertia. If an anomalous result appears early in training, it will take longer for the agent to learn corrections than if it were to occur later. Further, if learning is performed online with planning, this may bias the agent from exploring certain paths, whereas an offline learning phase would ameliorate this.

An alternative method which can be employed towards online training is the use of an artificial learning rate, implemented as a fixed proportion moving average calculation for the probabilities. Using such a technique, the probabilities at each update are calculated as proportions of a fixed number of samples. This would be calculated by setting the count increment as the scaled proportion between the actual sum of observations to the desired fixed window, and effecting the change in INC proportionally to this scale.

A.2 Alternative choice planning

In certain situations we might consider an agent which, rather than uniformly selecting the most probable action choice at any given state, may instead select from all available actions based on a weighted expectation of each.

For instance, in state s_i, we have $AFI[s_i, :, :]$ describing the probabilities of outcomes for actions taken from that state. For any action/result pair, then, we can an assign a local probability: $P_{s_i,a_l}(s_f) = P(a_l|AFI)P_{a_l(s_i)\to s_f}$ Where $P(a_l|AFI)$ represents the probability that a_l is chosen: the output of the decision algorithm. Using this expression, we can write an alternative method of hypergraph compression based around the probability of each state transition:

$$P(s_i \to s_f) = \Sigma_{\forall a_l} P(a_l|AFI)P_{a_l(s_i)\to s_f}$$

This equation compresses the hypergraph along the action slice by coupling all action results together with the $P(a_l|AFI)$ function. Note that if we define:

$$P(a_l|AFI) = \begin{cases} 1 & a_l = \underset{l}{\text{argmax}} P(a_l(s_i) \to s_f) \\ 0 & otherwise \end{cases}$$

Then the compression resolves to the maximally probable subgraph (essentially, a binary definition of an objective function, for which the GAP algorithm as presented herein is a special case). As an alternative, however, consider that we let

$$P(a_l|AFI) = \frac{P(a_l(s_i) \to s_f)}{\Sigma_{\forall a_l} P(s_f|s_i, a_l)} \tag{19}$$

In this case, then, 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 function may still be used to construct the necessary Markov Decision Process for analysis, albeit substantially more complex in representation. Such a system will converge less aggressively, but perhaps also exhibit overall–improved asymptotic behavior due to more expansive state space exploration. An envelope function using Equation 19 as the input parameter may then enable selection of policy to favor exploitation or exploration as an explicit, bounded system property. This adjustment is similar to the Tabu exploration described in 6.1.3.

A.3 In situ transfer functions

One alternative to learning a full AFI table for a problem would be to represent the output of the table, namely the conditional occasion probability, with a modeled function. Such functions are in fact fairly commonplace, as every state machine or combinatorial planner is, in essence, a distilled transfer function for a given problem operating on a fully binary transition space.

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. As in practice full states are often represented with vectors of individual substates, this would amount to storing some portion of the state vector in INC, and then expanding it into multiple vectors with latter portions generated by I at each step of the planning algorithm.

In application, this function would then be called during the planning stage, during which only allowable transitions would be evaluated at each step in building the most probable path subtree within AFI, and possibly even further specifying the associated probabilities with all or a subset of these paths vis-a-vis generating entries in INC, allowing for the array to be partly learned and partly generated by $I(\cdot)$. In this way, the size of the INC space may be reduced, supplanting learned relations for occasions in part with known rules.

If such a system is available, then through each step in which the GAP algorithm evaluates the transition probabilities, rather than fetching from an array a span of outputs from the transfer function may be collected instead. Naturally, in this case the consistent maintenance of the maximum probability subgraph is likely to be inefficient without the use of problem specific features of the state space. We can, however, note that because each node on the maximal probability tree, as it is built, contains the prior state information leading from the source node, we may instead maintain a single array linked list containing the sorted boundary nodes- for a complexity increase to $O(n)$.

Under this conception, one would essentially be substituting portions of the INC array with pre-existing model knowledge by calculating the transition probabilities from the model where applicable. 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 this would re-introduce the undesirable effects of designer choice into the system if achieved manually, automated analysis of INC could instead be used to derive $I(\cdot)$. The action of this evaluation would be tantamount to autonomous hierarchical decomposition of the state space via identification of an abstraction transform.

A.4 Generalized Heuristics for A*

Previously, mention was made of A* for implementing more computationally efficient versions of Algorithm 2. While the definition of heuristics which would be applicable to any possible problem space would be difficult at best, it is possible identify such an approach based in the algorithm's construction rather than the problem space.

We can illustrate this with a learned–structure based example. Recall that we implemented state encoding by use of a hash function, assigning states numerical labels in the order of discovery of the state. Given that under random exploration we expect states to be assigned label values roughly in proportion to the number of actions needed to transition between them, we can expect a structural relationship to be present in the AFI graph whereby transition probabilities are clustered around the primary diagonal.

This behavior can be observed in practice by examining visual plots of the AFI arrays. Figure 25 demonstrates this exact phenomenon for one of the $ToH_{3,5}$ agents and one of the Maze/TAXI agents after training. Because adjacent states are most likely to be determined

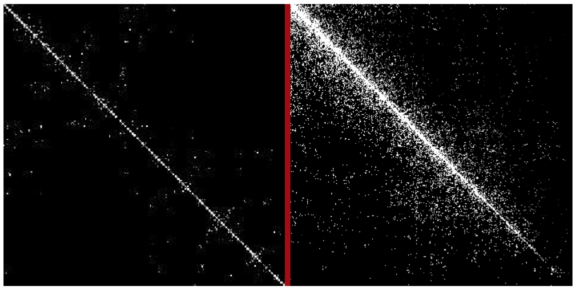


Figure 25: Visual representation of example AFI arrays for learning on the $ToH_{3,5}$ problem (left), and the Maze/TAXI problem (right) showing the relationship between structure in the AFI array and state discovery order

within relatively short time periods, there is a tendency towards a statistical distribution around the diagonal, with a measure of state density variance off the diagonal skewing the span of items away from this line, inversely proportional to the state index parameter.

Drawing from this observation, and presuming we have a roughly Gaussian distribution of probabilities around the line $s_i = s_f$ with a variance which decreases approximately linearly from some initial value v_1 (measured empirically) as $s_i \to |\mathcal{S}|$, we can use a bit of geometric inference to model the probability estimate $P_e(s_i, s_f)$:

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

Which approximates the joint probability of transition between two states, extrapolating from distribution of the observed direct transitions. We can then write a heuristic function $h(s_i)$ as $h(s_i) = P_e(s_i, s_g)$, with the starting state being s_0, such that A* maximizes:

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

Which function represents the expected total probability from the initial state, s_0, to the goal s_g, in terms of the actual joint probability from s_0 to s_i and the predicted remaining probability of transitioning from s_i to s_g extrapolated from the structure of AFI vis-a-vis the state discovery mechanism.

It is of course important to note that not all problems will allow this sort of adjacent state representation, and thus the problem topology will impact the exact probability distribution used to determine $P_e(s_i, s_f)$. For instance, in the left of Figure 25 we can observe two 'blocks' of available transitions in the upper left and lower right of the array, roughly corresponding to the two–phase nature of the workspace where the bulk of the disks are on the first or third peg. The original model is much better fitted for the Maze/TAXI case, and in general we would expect the state discovery oriented shape to be more preserved in highly localized problems. However, it is entirely possible to conceptualize a similar process of identifying a statistical distribution over AFI not based on the state discovery process which may guide the construction of $h(s_i)$ as illustrated above.

Indeed, looking at the block like structure in Figure 25 suggests that statistical analysis of AFI may allow for autonomously identifiable hierarchical decomposition in INC, as described in A.3, which may allow for the creation of beneficial heuristics or abstractions to represent the state space in more compact form. 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.

Appendix B. Additional Experimental Cases

In this section, we present some minor results from common test cases which are frequently used to illustrate the effectiveness of machine learning and automated planning systems, but which we do not include in the main body of the paper because: (a) they are insufficiently complex to present nuanced investigation of behavior of the GAP algorithm; and (b) the aggressive learning of the GAP algorithm eliminates fine differences between uncertain cases and thus precludes detailed analysis, similar to that seen with the learning of the $ToH(3, 5)$ case in Section 5.4.

B.1 Blocksworld

The Blocksworld problem domain is a simple, illustrative example used to study planning algorithms. It consist of, in essence, a 'table' on which blocks can be placed, numbered blocks which can be moved one at a time and stacked, and a final goal state in which all of the blocks are stacked on top of one another in order. It is particularly notable for the presence of the Sussman Anomaly, first presented by (Sussman, 1973), a fault in some kinds of planning algorithms in which an agent incorrectly resolves interleaved subgoals without generating a valid solution.

We test the GAP algorithm on the Blocksworld problem with numbers of blocks, N_b, ranging from 3 to 6 and at induced error rates of 0, 10, and 20 %. Each combination of N_b and error rate is sampled across 100 trials with randomly selected initial placement,

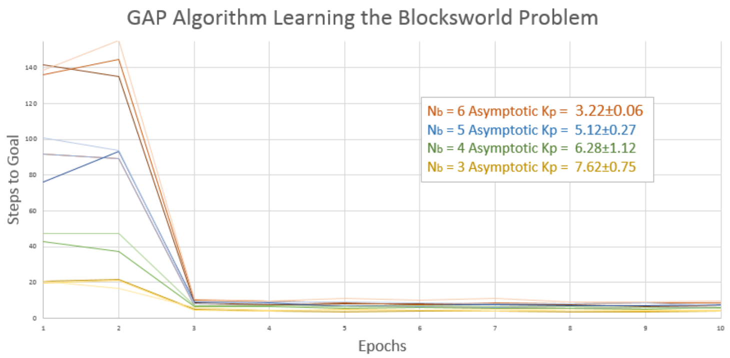


Figure 26: Learning curves for the Blocksworld problem, across varying numbers of blocks and error rates. Note that the rapid convergence precludes fitting of a proper reciprocal curve.

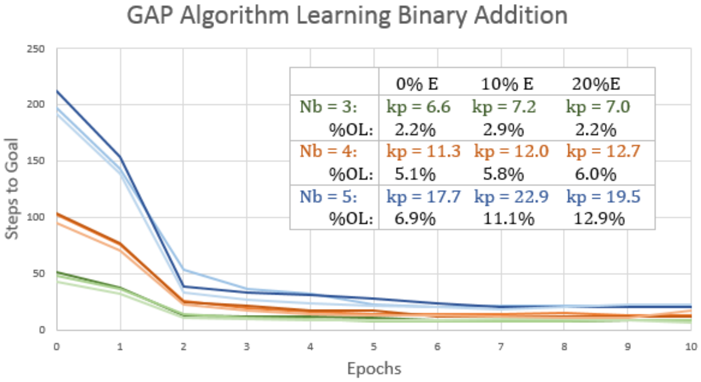
running to 10 epochs. For this problem, the proper ordering of blocks from highest index to lowest in any position is considered a goal, and planning is performed by building the maximal probability subtree until any such state is reached.

We find that the GAP successfully learns the problem, and does not become trapped. Further, as we can observe in Figure 26, the learning process is both expedient and comprehensive, even with induced error rates of 20%, and convergence to k_p uniformly occurring within 3 epochs. By examination we find that among our random samples for $N_b = 3$, 17 of the tests possessed the conditions for the Sussman anomaly to be present, 10 of the $N_b = 4$ trials, 12 of the $N_b = 5$, and 24 for $N_b = 6$, indicating that the GAP algorithm successfully avoids falling prey to the Sussman anomaly in both learning and planning phases.

B.2 Binary Addition

Binary addition is the process of taking the sequential digits of a pair of binary numbers and attempting to generate the corresponding next digit in their sum. It presents a useful demonstration case for learning algorithms as it is simple to implement, check, and design reward functions for. In the process of developing the GAP software, we used the simplicity of the binary addition problem as a trial case for validation and debugging.

For this problem, the world consists of two randomly chosen binary numbers of N_b many digits to be added together, a ’carry bit’ state, the resultant number, and the actual sum of the former pair, as well as an index to digits of the resultant. The agent’s possible actions are toggling the current resultant bit, toggling the carry bit, and incrementing or decrementing the index. States are constructed from the digits at the current index, the carry bit, and the current number of correct resultant digits in a relativistic construction similar to that used for the complex Maze/TAXI problem. The goal states, then, are any state in which all digits are correctly selected. As with the Blocksworld case, we perform experiments in batteries of 100 trials of 10 epochs each, with 0, 10, and 20% induced error and N_b of 3, 4, and 5 digit numbers.



	0% E	10% E	20%E
Nb = 3:	kp = 6.6	kp = 7.2	kp = 7.0
%OL:	2.2%	2.9%	2.2%
Nb = 4:	kp = 11.3	kp = 12.0	kp = 12.7
%OL:	5.1%	5.8%	6.0%
Nb = 5:	kp = 17.7	kp = 22.9	kp = 19.5
%OL:	6.9%	11.1%	12.9%

Figure 27: Learning curves for the binary addition problem, across 3, 4, and 5 digit numbers and error rates of 0, 10, and 20%, along with convergent limits for each battery and corresponding off–linear percents against reciprocal fit.

Results of these experiments are presented in Figure 27. On this graph, we can see the expected reciprocal learning curves, with percentage off–linear values and convergent performance limits listed on the inset chart. Notably, across each level of induced error, we can see that the asymptotic performance approximately doubles as N_b increases by one, corresponding to the doubling of the workspace size each time a digit is added, similar to the growth observed in the ToH problem with added disks.

References

Baird, L., & Moore, A. W. (1999). Gradient descent for general reinforcement learning. *Advances in neural information processing systems*, 968–974.

Bertsekas, D. P., & Tsitsiklis, J. N. (1991). An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, *16*(3), 580–595.

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial intelligence*, *90*(1-2), 281–300.

Blum, A. L., & Langford, J. C. (1999). Probabilistic planning in the graphplan framework. In *European Conference on Planning*, pp. 319–332. Springer.

Bylander, T. (1996). A probabilistic analysis of prepositional strips planning. *Artificial Intelligence*, *81*(1-2), 241–271.

Dicken, L., & Levine, J. (2010). Applying clustering techniques to reduce complexity in automated planning domains. In *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 186–193. Springer.

Dietterich, T. G. (1999). State abstraction in maxq hierarchical reinforcement learning. *arXiv preprint cs/9905015*.

Dijkstra, E. W., et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271.

Dimitrov, N. B., & Morton, D. P. (2009). Combinatorial design of a stochastic markov decision process. In *Operations Research and Cyber-Infrastructure*, pp. 167–193. Springer.

Ding, X., Smith, S. L., Belta, C., & Rus, D. (2014). Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, *59*(5), 1244–1257.

Fikes, R. E., & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, *2*(3-4), 189–208.

Fox, M., & Long, D. (2003). Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, *20*, 61–124.

Fox, M. S., Barbuceanu, M., & Teigen, R. (2001). Agent-oriented supply-chain management. In *Information-based manufacturing*, pp. 81–104. Springer.

Geffner, H. (2000). Functional strips: a more flexible language for planning and problem solving. In *Logic-based artificial intelligence*, pp. 187–209. Springer.

Georgievski, I., & Aiello, M. (2014). An overview of hierarchical task network planning. *arXiv preprint arXiv:1403.7426*.

Grzes, M. (2017). Reward shaping in episodic reinforcement learning..

Guillot, M., & Stauffer, G. (2020). The stochastic shortest path problem: a polyhedral combinatorics perspective. *European Journal of Operational Research*, *285*(1), 148–158.

Gutiérrez-Basulto, V., Jung, J. C., Lutz, C., & Schröder, L. (2017). Probabilistic description logics for subjective uncertainty. *Journal of Artificial Intelligence Research*, *58*, 1–66.

Hostetler, J., Fern, A., & Dietterich, T. (2017). Sample-based tree search with fixed and adaptive state abstractions. *Journal of Artificial Intelligence Research*, *60*, 717–777.

Hunter, A., & Thimm, M. (2017). Probabilistic reasoning with abstract argumentation frameworks. *Journal of Artificial Intelligence Research*, *59*, 565–611.

Jiménez, S., De La Rosa, T., Fernández, S., Fernández, F., & Borrajo, D. (2012). A review of machine learning for automated planning. *The Knowledge Engineering Review*, *27*(4), 433–467.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, 237–285.

Karami, A.-B., Jeanpierre, L., & Mouaddib, A.-I. (2009). Partially observable markov decision process for managing robot collaboration with human. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pp. 518–521. IEEE.

Knoblock, C. A. (1990). Abstracting the tower of hanoi. In *Working Notes of AAAI-90 Workshop on Automatic Generation of Approximations and Abstractions*, pp. 13–23. Citeseer.

Koenig, S., & Simmons, R. G. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, *22*(1), 227–250.

Konidaris, G., Kaelbling, L. P., & Lozano-Perez, T. (2018). From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, *61*, 215–289.

Lekavỳ, M., & Návrat, P. (2007). Expressivity of strips-like and htn-like planning. In *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pp. 121–130. Springer.

Leonetti, M., Iocchi, L., & Stone, P. (2016). A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artificial Intelligence*, *241*, 103–130.

Lüdtke, S., Schröder, M., Krüger, F., Bader, S., & Kirste, T. (2018). State-space abstractions for probabilistic inference: a systematic review. *Journal of Artificial Intelligence Research*, *63*, 789–848.

Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2006). Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning. In *International Conference on Artificial Neural Networks*, pp. 840–849. Springer.

McCallum, R. A. (1995). Reinforcement learning. *Advances in Neural Information Processing Systems 7*, *7*, 377.

McDermott, D. M. (2000). The 1998 ai planning systems competition. *AI magazine*, *21*(2), 35–35.

Pineda, L., & Zilberstein, S. (2019). Probabilistic planning with reduced models. *Journal of Artificial Intelligence Research*, *65*, 271–306.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, Vol. 37. Citeseer.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, *5*(2), 115–135.

Sacerdoti, E. D. (1975). The nonlinear nature of plans. Tech. rep., STANFORD RESEARCH INST MENLO PARK CA.

Steinmetz, M., Hoffmann, J., & Buffet, O. (2016). Goal probability analysis in probabilistic planning: Exploring and enhancing the state of the art. *Journal of Artificial Intelligence Research*, *57*, 229–271.

Sussman, G. J. (1973). A computational model of skill acquisition..

Sutton, R. S., & Barto, A. G. (1987). A temporal-difference model of classical conditioning. In *Proceedings of the ninth annual conference of the cognitive science society*, pp. 355–378. Seattle, WA.

Szepesvári, C., & Littman, M. L. (1996). Generalized markov decision processes: Dynamic-programming and reinforcement-learning algorithms. In *Proceedings of International Conference of Machine Learning*, Vol. 96.

Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey.. *Journal of Machine Learning Research*, *10*(7).

Van Otterlo, M., & Wiering, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement learning*, pp. 3–42. Springer.

Van Zanten, A. (1990). The complexity of an optimal algorithm for the generalized tower of hanoi problem. *International journal of computer mathematics*, *36*(1-2), 1–8.

Vidyasagar, M. (2020). Recent advances in reinforcement learning. In *2020 American Control Conference (ACC)*, pp. 4751–4756. IEEE.

Vodopivec, T., Samothrakis, S., & Ster, B. (2017). On monte carlo tree search and reinforcement learning. *Journal of Artificial Intelligence Research*, *60*, 881–936.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

Watkins, C. J. C. H. (1989). Learning from delayed rewards..

White, D. J. (1985). Real applications of markov decision processes. *Interfaces*, *15*(6), 73–83.

Wu, G., Say, B., & Sanner, S. (2020). Scalable planning with deep neural network learned transition models. *Journal of Artificial Intelligence Research*, *68*, 571–606.

Younes, H. L., & Littman, M. L. (2004). Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, *2*, 99.

Zimmerman, T., & Kambhampati, S. (2003). Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, *24*(2), 73–73.