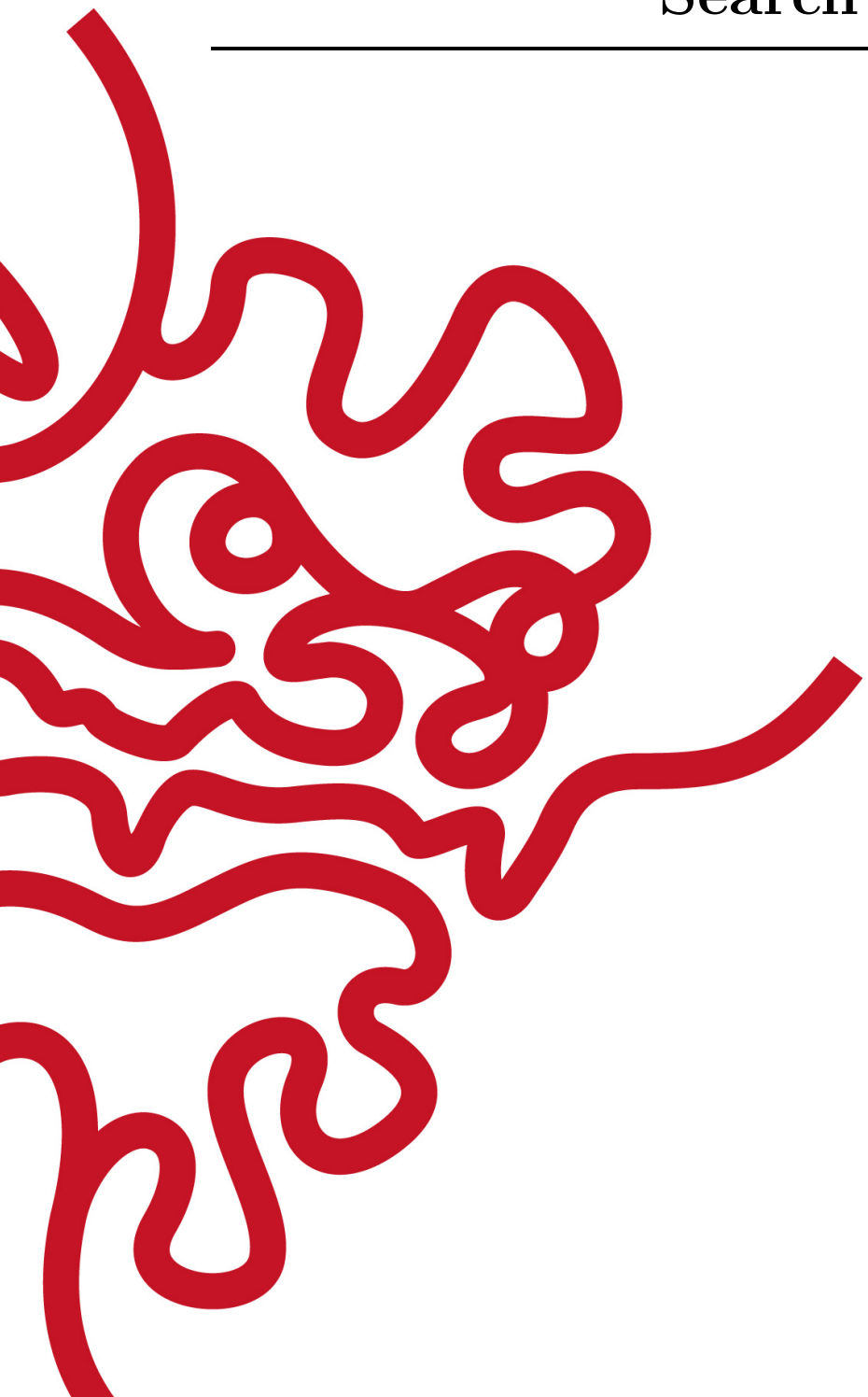
Okinawa Institute of Science and Technology
Graduate University

Thesis submitted for the degree

Doctor of Philosophy

Hyperadaptability by Behavioral Search



by

Alex Baranski

Supervisor: **Jun Tani**

December 2025

Declaration of Original and Sole Authorship

I, Alex Baranski, declare that this thesis entitled *Hyperadaptability by Behavioral Search* and the data presented in it are original and my own work.

I confirm that:

- No part of this work has previously been submitted for a degree at this or any other university.
- References to the work of others have been clearly acknowledged. Quotations from the work of others have been clearly indicated, and attributed to them.
- In cases where others have contributed to part of this work, such contribution has been clearly acknowledged and distinguished from my own work.
- None of this work has been previously published or pre-printed elsewhere, with the exception of the following:

  A. Baranski and J. Tani, "Life, uh, Finds a Way: Hyperadaptability by Behavioral Search," *arXiv preprint arXiv:2410.01349*, 2025.

  Contributions:

  - A. Baranski conceived of the research topic under the supervision of J. Tani.
  - A. Baranski developed the theoretical results.
  - A. Baranski developed and wrote the algorithm.
  - A. Baranski developed and wrote the neural implementation.
  - A. Baranski conceived of and ran all experiments.
  - A. Baranski analyzed and interpreted all results.
  - A. Baranski conceived of, wrote, and edited the manuscript.

Date: December 2025
Signature:

Abstract

Organisms possess extraordinary adaptive potential, far surpassing learning machines in their robust ability to solve problems. Even in totally novel situations where they lack long-term experience, organic intelligence can improvise new useful behaviors. To investigate the core mechanisms enabling behavioral adaptation, I study its theoretical limit: the *guaranteed* ability to solve *any* solvable continuous-domain problem, in unbounded but *finite time*, *regardless* of the quantity or quality of any *prior knowledge*. I call this limit *hyperadaptability*. With *no* prior knowledge, the only way to provide this guarantee is to perform a complete *search* over the set of *all possible behaviors*.

As this set is *uncountably infinitely* large, it is mathematically impossible to perform a complete search of it. However, I show how to construct a cognitively meaningful subset of all behaviors which is *countably infinite*, so that any behavior in the set can be reached by just trying one behavior after another, and is *dense* in the set of *all behaviors*, so that any possible behavior can be arbitrarily well-approximated. Searching this set one behavior at a time guarantees that any behavior will eventually be tried to arbitrary precision, ensuring the solution of arbitrary problems. The construction of this set uses a mutable cognitive graph, with behaviors corresponding to paths over the graph. The graph can be extended to add qualitatively new behaviors, refined to increase behavioral precision, and partially inhibited to temporarily ignore ineffective behaviors. Careful control over the feedback loop between path selection and graph mutation ensures that the search through behavior-space is complete without being brute-force.

I validate hyperadaptability by behavioral search using simulation experiments to test the effectiveness of a proof-of-concept algorithm based on the theory, which is able to rapidly master navigation of random mazes, and outperforms a strong baseline on a challenging reinforcement learning task. To demonstrate the biological possibility of behavioral search, the cognitive graph is implemented in a neural network using Hebbian learning and a novel harmonic state-space encoding that supports dynamic representation resolution, which together bear a strong resemblance to the hippocampal complex.

The theory detailed in this thesis provides a novel way of understanding behavioral adaptation, enabling fundamentally reliable problem-solving in continuous domains. The specific construction used for the theory is functional, but also illustrative of a staggering variety of alternative schemes that could be dramatically more efficient, flexible, and powerful. By circumventing core conceptual issues in current approaches, this framework offers a new path toward completely autonomous systems that can quickly and robustly self-organize their behavior to acquire new skills in an open-ended way.

Acknowledgments

I would like to acknowledge my supervisor, Jun, for early conversations while I explored what topic to study, and for providing me with lab computing resources. I would also like to acknowledge OIST's Scientific Computing & Data Analysis Section for use of the Saion cluster of computers so I could run many parallel simulation experiments. I would like to thank Henrique Oyama, Jeffrey Queißer, and Lakshmipriya Swaminathan for reading and providing commentary on this thesis or [1], on which it is based.

List of Abbreviations

AI	Artificial Intelligence
AGI	Artificial General Intelligence
DL	Deep Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
GWM	Graph Working Memory
HaRK	Harmonic Relational Key
ARMS	Adaptive Real-time Metasearch over Segraphs
BIX	Behavioral Index Structure

Glossary

Hyperadaptability	The ability of a system (organism or agent) to guarantee finding a solution to a solvable problem in finite time.
Ethogenesis	The process of the generation of new behaviors.
Segraph	A graph that segments a state-space.
Adaptive Real-time Metasearch over Segraphs	Algorithm for performing a pragmatic search over a set of rational behaviors.
Harmonic Relational Key Memory	A memory system using random Fourier features and Hebbian learning.

Nomenclature

$\mathcal{A}$	An agent
S	A Euclidean **statespace**
S_0	The subset of S physically accessible from the agent's starting state
$\boldsymbol{s}, \boldsymbol{s}^\star$	A state in S, and a goal-state in S (together, a **problem**)
K	A stateful, goal-oriented **controller** that tries to reach a goal-state
${}^+\mathsf{x}(\boldsymbol{h}_t)$	Controller **success** (0 for not yet, 1 for success)
${}^-\mathsf{x}(\boldsymbol{h}_t)$	Controller **failure** (0 for not yet, 1 for failure)
$T^K_{\boldsymbol{s},\boldsymbol{s}^\star}$	Termination time (either ${}^+\mathsf{x}(\boldsymbol{h}_t) = 1$ or ${}^-\mathsf{x}(\boldsymbol{h}_t) = 1$) for controller K trying to reach $\boldsymbol{s}^\star$ from $\boldsymbol{s}$
$\mathfrak{t}_K(\boldsymbol{s}, \boldsymbol{s}^\star)$	Finite physical **trajectory** generated by controller K going from $\boldsymbol{s}$ to $\boldsymbol{s}^\star$
$r_K(\boldsymbol{s}, \boldsymbol{s}^\star)$	**Reachability** for controller K from $\boldsymbol{s}$ to $\boldsymbol{s}^\star$ (failure=0, success=1)
$\mathrm{con}_K(\boldsymbol{s})$	The points reachable by controller K from state $\boldsymbol{s}$
$\mathrm{pre}_K(\boldsymbol{s})$	The points that can be reached from state $\boldsymbol{s}$ using controller K
$\mathfrak{s}$	A **plan**, or sequence of states (waypoints)
$\mathfrak{t}_K(\mathfrak{s})$	Finite physical **trajectory** generated by controller K following plan $\mathfrak{s}$
$r_K(\mathfrak{s})$	**Reachability** for controller K following plan $\mathfrak{s}$ (failure=0, success=1)
$B_\epsilon(\boldsymbol{s})$	A hyperball around $\boldsymbol{s}$ with radius ϵ, $B_\epsilon(\boldsymbol{s}) = \{\boldsymbol{s}' \in S : \|s' - s\| \leq \epsilon\}$
$\mathfrak{S}$	The **set of all plans**. This symbol admits many variations. The set of plans of length n is $\mathfrak{S}^n$, of any length, $\mathfrak{S}^*$. The set of plans with waypoints in the set A is $\mathfrak{S}_A$. The set of robust plans is ${}^+\mathfrak{S}$. The set of plans that start at $\boldsymbol{s}$ and end at $\boldsymbol{s}'$ is $\mathfrak{S}(\boldsymbol{s}, \boldsymbol{s}')$
$\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$	The radius-$\boldsymbol{\epsilon}$ tube around plan $\mathfrak{s}$ of alternative plans
${}^+\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$	The subset of $\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$ that is successful
$r^{\boldsymbol{\epsilon}}_K(\mathfrak{s})$	The fraction of $\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$ that succeeds, if $r^{\boldsymbol{\epsilon}}_K(\mathfrak{s}) = 1$ for $\boldsymbol{\epsilon} > 0$ then $\mathfrak{s}$ is **robust**
S_K	The subset of state-space reachable using the controller K
$\mathfrak{s}_1 \sim \mathfrak{s}_2$	Plans are equivalent if they are both solutions to the same problem, starting at the same point $\boldsymbol{s}^\circ$, ending at the same point $\boldsymbol{s}^\star$, and succeeding, meaning $r_K(\mathfrak{s}_1) = 1$ and $r_K(\mathfrak{s}_2) = 1$. Equivalently, $\mathfrak{s}_1, \mathfrak{s}_2 \in {}^+\mathfrak{S}^*(\boldsymbol{s}^\circ, \boldsymbol{s}^\star)$
$\xi^{S_K}_{\boldsymbol{s},\boldsymbol{s}'}$	Equivalence class of plans from $\boldsymbol{s}$ to $\boldsymbol{s}'$ with waypoints in S_K, equal to ${}^+\mathfrak{S}^*_{S_K}(\boldsymbol{s}, \boldsymbol{s}')$
Ξ_{S_K}	The set ${}^+\mathfrak{S}^*_{S_K}$ partitioned by $\sim$, i.e. the **set of solveable problems** in S_K, or the set of all equivalence classes of plans with waypoints in S_K

$\Xi^{\mathcal{A}}_{S_K}$	The set of equivalence classes of solutions with waypoints in S_K that the agent $\mathcal{A}$ can generate
$\mathcal{S}$	A discrete set of waypoints in the state-space S
E	A function assigning an ϵ to each waypoint in $\mathcal{S}$
B_E	A function assigning a hyperball to each waypoint in $\mathcal{S}$ with radius given by E
$\mathbf{b}_{\boldsymbol{s}}$	The hyperball assigned to the waypoint $\boldsymbol{s}$
$\mathcal{B}$	A set of hyperballs. $\mathcal{B}(\mathcal{S})$ are the hyperballs corresponding to points in $\mathcal{S}$
$\mathfrak{b}$	A sequence of hyperballs, or **path**. If $\mathfrak{s}$ is a plan drawn from $\mathfrak{S}^*_{\mathcal{S}}$, then $\mathfrak{b}_{\mathfrak{s}}$ is the corresponding sequence of hyperballs
$[\![\mathfrak{b}]\!]$	The bundle of plans ($\boldsymbol{\epsilon}$-tube) represented by the hyperball sequence $\mathfrak{b}$
$r_K(\mathfrak{b})$	The $\boldsymbol{\epsilon}_{\mathfrak{b}}$-reliability of $\mathfrak{b}$, where $\boldsymbol{\epsilon}_{\mathfrak{b}}$ is the vector of corresponding hyperball radii. $\mathfrak{b}$ is **robust** if $r_K(\mathfrak{b}) = 1$
`refine`	The **refinement mutation**, replaces a hyperball with several smaller hyperballs in the same area, increasing the local resolution of $\mathcal{B}(\mathcal{S})$
$\mathfrak{B}$	The **set of all paths**. This symbol admits many variations. The set of paths of length n is $\mathfrak{B}^n$, of any length, $\mathfrak{B}^*$. The set of paths with hyperballs in the set A is $\mathfrak{B}_A$, all paths that are robust is ${}^+\mathfrak{B}$. All paths from $\mathbf{b}_1$ to $\mathbf{b}_2$ is $\mathfrak{B}(\mathbf{b}_1, \mathbf{b}_2)$
$\mathfrak{b}_1 \sim \mathfrak{b}_2$	Paths are equivalent if they are both robust, start at the same hyperball, and end at the same hyperball. Equivalently, $\mathfrak{b}_1, \mathfrak{b}_2 \in {}^+\mathfrak{B}^*(\mathbf{b}, \mathbf{b}')$
$\xi^{\mathcal{B}}_{\mathbf{b},\mathbf{b}'}$	Equivalence class of paths from $\mathbf{b}$ to $\mathbf{b}'$ with hyperballs in $\mathcal{B}$, equal to ${}^+\mathfrak{B}^*_{\mathcal{B}}(\mathbf{b}, \mathbf{b}')$
$\Xi^{\mathcal{B}}_{\mathcal{B}}$	The set ${}^+\mathfrak{B}^*_{\mathcal{B}}$ partitioned by $\sim$, i.e. the set of all equivalence classes of *paths* with hyperballs in $\mathcal{B}$
$\Xi^{\mathcal{B}}_{S_K}$	The set of equivalence classes of *plans* with members inside a path in ${}^+\mathfrak{B}^*_{\mathcal{B}}$
`extend`	The **extension mutation**, adds new larger hyperballs around an existing hyperball, increasing the area of statespace covered by $\mathcal{B}(\mathcal{S})$
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A **segraph**, a graph with directed edges $\mathcal{E}$ that connect vertices $\mathcal{V}$ that segment statespace, which is used to organize the enumeration of plans
$\mathbf{v}_i$	A segraph **vertex** $\mathbf{v}_i$ representing a "chunk" of state-space, a corresponding hyperball $\mathbf{b}_i \subset S$
$\mathbf{e}_{i,j}$	The directed segraph **edge** between vertices $\mathbf{v}_i$ and $\mathbf{v}_j$, representing a chunk of connection space $\mathbf{c}_{i,j} = \mathbf{b}_i \times \mathbf{b}_j \subset S^2$
M	The **measure** of a vertex or edge, $M(\mathbf{v}_i) = \lvert\mathbf{b}_i\rvert$, $M(\mathbf{e}_{i,j}) = \lvert\mathbf{c}_{i,j}\rvert$
$r_K(\mathbf{e}_{i,j})$	The theoretical **reliability** of the edge $\mathbf{e}_{i,j}$, equal to $r_K((\mathbf{b}_i, \mathbf{b}_j))$
$n_s(\mathbf{e})$	The number of **successful traversals** by $\mathcal{A}$ over $\mathbf{e}$
$n_f(\mathbf{e})$	The number of **failed traversals** by $\mathcal{A}$ over $\mathbf{e}$
$n(\mathbf{e})$	The total number of **traversals** by $\mathcal{A}$ over $\mathbf{e}$, equal to $n_s(\mathbf{e}) + n_f(\mathbf{e})$

$\tilde{r}_{K}(\mathbf{e})$	The empirically **estimated reliability** of $\mathbf{e}$
$\mathfrak{p}$	A **path**, a sequence of unique vertices
$\mathfrak{P}$	The **set of all paths**. This symbol admits many variations. The set of paths of length n is $\mathfrak{P}^n$, of any length, $\mathfrak{P}^*$. The set of paths with vertices in the set A is $\mathfrak{P}_A$. The set of robust paths is ${}^{+}\mathfrak{P}$. The set of paths that start at $\mathbf{v}$ and end at $\mathbf{v}'$ is $\mathfrak{P}(\mathbf{v}, \mathbf{v}')$
$\mathcal{G}$	The probability distribution over goals used to represent the way an agent selects problems to try to solve
$\mathcal{P}$	The probability distribution over paths used to represent the way an agent selects plans to try to solve a problem
$r_{K}(\mathfrak{p})$	The reliability of a path, equal to the reliability of the corresponding hyperball sequence. When $r_{K}(\mathfrak{p}) = 1$, the path is **robust**
$\xi^{\mathcal{G}}_{\mathbf{v},\mathbf{v}'}$	The equivalence class of robust paths over $\mathcal{G}$ between $\mathbf{v}$ and $\mathbf{v}'$
$\Xi^{\mathcal{G}}_{\mathcal{V}}$	The set of all equivalence classes of robust paths over $\mathcal{G}$
$\texttt{try}(\mathfrak{p})$	A function modeling the action of an agent **trying to follow a path**
$\mathcal{E}^{\times}$	The set of edges that fail when an agent tries to follow a path. If $\mathcal{E}^{\times} = \emptyset$, then the path succeeds
$\mathcal{E}^{\downarrow}$	The set of edges that are inhibited (excluded from pathfinding by the agent) due to having failed while trying paths
$\mathcal{E}^{+}$	The set of edges that are not inhibited and can be used in pathfinding
$\alpha(\mathfrak{p})$	The total affordance provided by a path $\mathfrak{p}$
$q(\mathbf{e})$	The utility of an edge $\mathbf{e}$ in terms of the incremental affordance it provides
$q_{\text{ref}}(\mathbf{e})$	The total utility of the edges produced by refining $\mathbf{e}$
$\Delta q(\mathbf{e})$	The theoretical change in utility resulting from refining $\mathbf{e}$
$\Delta\tilde{q}(\mathbf{e})$	An empirical estimate of $\Delta q(\mathbf{e})$
α'	An affordance-threshold used to regulate the order in which paths are tried
$\alpha(\mathbf{e})$	The affordance of the edge $\mathbf{e}$, $\alpha(\mathbf{e}) = r_{K}(\mathbf{e}) \cdot M(\mathbf{e})$
$\tilde{\alpha}(\mathbf{e})$	The estimated affordance of the edge $\mathbf{e}$, $\tilde{\alpha}(\mathbf{e}) = \tilde{r}_{K}(\mathbf{e}) \cdot M(\mathbf{e})$
$\mathcal{E}^{<}(\alpha')$	The set of edges that are below the affordance threshold α'
$\mathcal{E}^{\geq}(\alpha')$	The set of edges that are above the affordance threshold α'
$\mathfrak{P}_{\mathcal{G}}(\mathbf{v}, \mathbf{e})$	The set of all paths over $\mathcal{G}$ that start at $\mathbf{v}$ and end by crossing $\mathbf{e}$
$\mathcal{P}^{\mathbf{v}\to\mathbf{e}}_{\mathcal{G}}$	The probability distribution of paths from $\mathbf{v}$ to $\mathbf{e}$ over $\mathcal{G}$
$\Upsilon(\mathbf{e})$	The epistemic value of an edge $\mathbf{e}$, used to weigh the probability that $\mathbf{e}$ is chosen as a goal, modeled by the distribution $\mathcal{G}_{\Upsilon}$ over edges
$\mathbb{O}_{S}$	The set of all hyperballs in S.
$s(\cdot)$	A *selector* function.
$\check{\delta}(\cdot)$	A *generator* function.
$\mathfrak{m}_{s,\check{\delta}}(\cdot)$	A *mutator* function.
M	A set $\{\mathfrak{m}_1, \mathfrak{m}_2, ..., \mathfrak{m}_n\}$ of mutator functions.
L	A *mutation schedule* for a set of mutations.

$k_{m_j}(i)$, $k_j(i)$	The schedule-index in a mutation schedule after which the mutator m_j has been called i-times, called the m_j-index.
$m_{M,L}(\cdot)$, $m^i(\cdot)$	A *polymutator* function applying a set M of mutators.
$I_j(\mathbf{b})$	The *index function* the indicates the m_j-index at which a specific hyperball $\mathbf{b}$ is selected for mutation.
$\psi(\mathcal{B}_0, \mathfrak{s}, r)$	The schedule-index after which the plan $\mathfrak{s}$ is guaranteed to be been generated by a polymutator, starting from $\mathcal{B}_0$, which is within $\mathcal{N}_r(\mathfrak{s})$.

To my wife Sarah, for laughing with me.

To my parents, for raising me.

To my grandparents, Uncle Dave, and Rich:
I wish you could read this.

Table of Contents

List of Figures

List of Tables

Chapter 1

Hyperadaptability

The world is always changing, and to survive, one must change with it. In particular, *adaptability* is the capacity to change *usefully* in response to new goals and circumstances. Life has endured for billions of years on Earth by adapting through the change of heritable traits such as metabolism, morphology, and instinctual behavior over successive generations [2]. Living things (organisms) also possess the ability to adapt within their lifetimes. Cellular pathways can reorganize themselves in new chemical environments [3], as can gene-regulatory networks [4, 5]. Bodies can heal injuries [6], strengthen tissues like skin [7], bone [8], and muscle [9] through use, and even reorganize their morphology for phylogenetically new locomotion strategies [10]. Given the plasticity of cells, tissues, and organs, it is perhaps not surprising that a wide range of organisms demonstrate plasticity of *behavior*.

The advantage enjoyed by an organism with some degree of behavioral adaptability is clear: by being able to generate novel behaviors to achieve a goal, the organism's ability to satisfy its wants and needs becomes partially invariant to the nature of those wants and needs, the environment it finds itself in, and even its own prior experience and insight. In a sense, the potential *degree* of behavioral novelty is a measure of the adaptability of an organism. At the lowest end of the spectrum, a "new" behavior could be practically identical to an existing one, perhaps acquired through observation or tutelage. Moving up, a behavior could be modified in some small way, such as reaching a bit farther or walking a bit slower. Moving up again, a behavior could be *constructed* through recombination of other behaviors or their parts, as if rearranging blocks. In the extreme, we can imagine a behavior that emerges seemingly *ex nihilo*, through either grueling effort or a stroke of genius. As a matter of logical necessity, in all these cases *something new* has to enter the world. This raises an interesting question: *where* does the *newness* come from, and *how* does it get *here*? This is the *problem of ethogenesis*[1], or the origin of behavior. In order to understand behavioral adaptation, or "useful" ethogenesis, I propose the theoretical limit of behavioral adaptability: *hyperadaptability*, the guaranteed knowledge-invariant ability to generate a behavior to achieve any achievable goal, in any context, in finite time.

Of course, practical limitations on cognitive resources ensure nothing is *truly* hyperadaptable; nevertheless, it provides a useful theoretical lens through which to view behavior and adaptation, in much the same way that we say that physical computers

[1] From Greek *êthos*: 'habit', 'character', or 'custom'; and *genesis*: 'origin'.

are Turing-complete [11], even though Turing-completeness requires infinite memory, which no real computer has. In this thesis, I study the problem of how ethogenesis must occur in order to achieve hyperadaptability, ultimately offering the following solution: the solution to every solvable problem exists in an *infinite space of all possible behaviors*. By *systematically*, but not necessarily *unintelligently*, searching this space, an organism can *guarantee* that it finds *a* solution. However, this space, which I call the *real behaviors* in analogy to $\mathbb{R}$ (for the uninitiated, see the footnote[2]), is *uncountably infinite*[2] and *unbounded*, meaning it *cannot be searched*. This apparent impasse is resolved by constructing *dense*[3] *countably infinite*[2] *subsets* of the real behaviors. By virtue of their countability, any of these sets can be systematically *searched* by simply *enumerating*[2] them, and since they are *dense* in the set of *all real behaviors*, an agent searching one of these sets will eventually get arbitrarily close to any desired "real" behavior. Because of this, in analogy to $\mathbb{Q}$, I call them *rational*[2].

Enumerating a rational set of behaviors by trying each of them in turn until one solves an arbitrary problem is equivalent to *searching it for a solution*, which due to the countability of the set, is guaranteed to occur in finite time, thus enabling *hyperadaptability*. As we will see, the enumeration suggested by the structure of the behavior-space lends itself to a very natural interpretation in terms of cognitive heuristics for trial-and-error exploration. I formalize this *adaptation-as-search* perspective with a theory of *behavioral search* in which the *state* of a *cognitive graph* [12] serves as an *index structure* for a set of rational behaviors. I use this theory to design an algorithm that operates over a neural instantiation of the cognitive graph, and test it in difficult simulated environments.

1.1 Search vs. Inference

Adaptation-as-search is *not* the predominant view of behavioral adaptation within the fields that study it: psychology, neuroscience, and artificial intelligence. The more prevalent view [13–21] in these fields is what I will here call the *adaptation-as-inference* perspective, which in brief, takes adaptability to be a consequence of the capacity to extract statistical regularities from observations which generalize to new situations by virtue of their (hoped-for) correspondence to latent structure in reality. By combining methods from optimization, statistical inference, and function approximation, this view has achieved trans-disciplinary status as a unifying theoretical framework, with both scientific explanatory power [22] and impressive technological results, specifically as manifested in deep reinforcement learning (DRL).

From this point of view, my proposal may seem quixotic, so I will spill some ink defending the broader point. DRL, which uses deep learning (DL) to implement rein-

[2]Some mathematics knowledge is assumed of the reader. To begin $\mathbb{N} = \{0, 1, 2, ...\}$ and so on is the set of "counting" or "natural" numbers. The set is infinite, but any element can be reached in a definite number of finite steps, i.e., "counted to", so $\mathbb{N}$ is said to be *countably-infinite*, or *enumerable*. Any set that can be arranged by putting all of its elements "side-by-side" with $\mathbb{N}$ is also countably-infinite. The set of all integers $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$ is such, as is the set of all fractions $\mathbb{Q}$. However, the set $\mathbb{R}$ of all numbers on the number line, including $\sqrt{2}, \pi$, etc. can*not* be so-ordered, so it is un*countably infinite*.

[3]One set is dense in another if every element of the latter is arbitrarily close to one of the former.

forcement learning (RL) algorithms, has *significantly* advanced the ability of machines to perform tasks previously restricted to biological intelligence, which has lent credence to the idea that DRL may serve itself as a model *for* biological intelligence. However, at least two glaring gaps remain: DRL agents require *vastly* more "experience" than biological agents to master a task [23], and are dramatically *worse* at *generalizing* from those experiences [24–28]. Despite the development of many methods [29–37] to address these problems, success has been limited [38, 39]. While "life finds a way" [40], it seems our machines often do not.

Clearly, generalization and statistical inference *are* important: however, I believe it is misguided to think that taking this approach to its limit will yield artificial general intelligence, because I am convinced that they are, by necessity, only *part* of what intelligence is. In short, the adaptation-as-inference perspective has an untenably *empirical* epistemology. The ultimate hope, that we can build a machine that will extract veridical representations of latent structure that perfectly generalize to new situations, relies on the *logically invalid* inference of a universal rule about one's observations *from those observations* [41]. The adaptation-as-*search* perspective adopts a different epistemology: *critical rationalism* [42], which rather than *inferring* rules *from* observations, *conjectures hypotheses* that can be *falsified by* observations.

What difference does this make? Within empirical induction, there is a mystery about what is the "right" rule to infer from a set of observations. Within critical rationalism, this mystery is dissolved: one simply guesses the *maximally general rule* one can imagine, then tries to prove it wrong. Without the assumption of a "right" rule that is somehow "contained" in the observations, one is freed to go *well beyond* the data, conjecturing wildly such that making extremely far-reaching generalizations is trivial [43]. The history of science shows this pattern: many wildly wrong conjectures, and a sprinkling of gems that have survived all tests (so far). The takeaway for behavioral adaptation is that at no point, when we are about to *do* something in order to achieve a goal, are we justified in being sure that it will work: there is always a chance that we will be obliged to do something *completely different*, necessitating *search*.

To be more specific, and by way of explanation of DRL's tremendous success, the methods of DRL and its antecedents were developed in "easy-mode", within domains where empirical induction could reasonably be expected to work, e.g. in *games*. The theoretical assumptions that the adaptation-as-inference perspective was built on are the existence of a stable distribution of events, limitless and independent interactions with that distribution, and the distribution being discrete and (usually) finite. Finding ways to make DRL and related methods work when combinations of these assumptions are relaxed has made up the bulk of research in the field. While I have of course had to make my own simplifying assumptions, I have to the best of my abilities and in the time allotted to me developed a theory of behavioral adaptation in "hard-mode": limited and non-independent interactions with a continuous and unbounded distribution that is only stable on short time-frames. The aim of future work, of course, will be to seamlessly combine the two perspectives and make *no* simplifying assumptions.

In attempting to explain all of intelligence and behavioral adaptation in terms of inference, my feeling is that we have, in seeing a tree from high up and far away, concluded that it is made mostly of leaves. However, a tree without leaves is still recognizably a tree, whereas one without a trunk or branches is just autumn amuse-

ment. Likewise, I blame the impressiveness and obvious importance of the neocortex for the pervasive focus in our field on inference. Nevertheless, while an animal without its neocortex is rather simple and foolish, it still recognizably acts like an animal and can adapt its behavior, whereas the same animal without its subcortical structures is dead. To my mind, the primary *engine* of behavioral adaptation is the ability to *search* through the space of all behaviors. In this context, the role of *inference* is to *bias* the search by *prioritizing* behaviors that, based on past experience, are thought likely to work again. If they do, they do, if they don't, the behavioral search merely continues.

1.2 Behavioral Search

Pondering the requirements of behavioral adaptation in general and some specific features of organic ethogenesis in particular have, in the context of the "hard-mode" version of reality, driven me to adopt what at first glance may seem to some a peculiar formulation of behavior, and behavioral search. I will leave the *motivation* for these choices to the next Chapter, and for now simply provide an overview. What I introduce in this thesis should not be considered *the* theory of behavioral search, but rather, *one principled way* of constructing a behavioral search theory.

First, rather than consider the point of behavior to be the maximization of a *reward function* that implicitly defines a task, we instead view it as being to solve a *problem*, where a problem is merely the situation of being in one part of state-space and wanting to be in *another* part. A state-space is construed very broadly: The state-space could be the arrangements of a chess-board, the configurations of a robot's servos and sensors, or physical space. In those cases, goals could be any position with your opponent in check-mate, a specific move of a dance, or a used bookstore, respectively.

In my formulation, a behavior has two aspects, one *aspirational* (called a plan) and the other *actual*, a trajectory generated by a *low-level goal-directed controller* that tries to execute the plan. Plans are represented by sequences of waypoints or *sub-goals*, with the simplest plans corresponding to just a start-point and target-point being called *1-plans*. Arbitrarily complex plans are constructed by sequential composition of 1-plans. A 1-plan succeeds if the controller reaches the target, and a complex-plan succeeds if all of its constituent 1-plans succeed: otherwise it fails. The agent searches *plan-space* by trying one plan after another, until one succeeds. We do *not* let the agent "reset" to an initial state, as is usually done in RL.

To actually enumerate a set of rational behaviors, we need a way to *index* the set and a way to "increment" the index using a *successor function*[4]. I call my choice of *index structure* a *segraph*, a graph whose vertices are associated with chunks of state-space called *fields*, together segmenting state-space. A directed edge between two vertices represents the set of all possible 1-plans between the fields of the edge's start and target vertices. To generate a specific plan, the agent first finds a path from its current vertex to a vertex containing its goal. As a single edge represents infinitely many 1-plans, so a path represents infinitely many plans. The agent samples one plan and follows it.

The edges of the graph record the number of failed and successful 1-plans sampled

[4]I am abusing terminology: *the* successor function is a specific endofunction on the natural numbers.

from them (called *traversals*): the agent *provisionally* generalizes about all 1-plans in an edge from a finite sample, and uses this information to *deprioritize* plans that are *empirically* less likely to succeed. The generalization is provisional because the agent can always go back and *refine* an edge that has failed, in order to more closely *probe* that region of plan-space, increasing the "resolution" of the agent's behavior. The *scope* of potential behaviors can be increased by *extending* the segraph, adding vertices to it to cover new, previously un-covered regions of state-space.

By balancing extension and refinement (that is, not "neglecting" to refine or extend any part of the graph that needs it), it is guaranteed that for any given "real" plan, the graph will eventually contain a path that is arbitrarily close to that plan, thus making the graph an index for a specific set of "rational" plans that is *dense* in the set of *all* plans. This means that an agent using this graph is guaranteed to find, in finite (but unbounded) time, a plan (and thus behavior) that solves it's problem, no matter the problem, and no matter its prior knowledge, making that agent *hyperadaptable* by allowing it to *try anything*.

To practically implement this theory I have developed a specific algorithm called Adaptive Real-Time Metasearch over Segraphs (ARMS). This algorithm is more ad-hoc than the general theory, and should be taken as a proof-of-concept. Because I am interested also in the biological realization of this theory, the algorithm operates over a *neural realization* of the segraph, which bears a cursory resemblance to both the structure and the function of the hippocampus and entorhinal cortex, neuroanatomical regions that are believed to be ancient and predate the much newer neocortex, making it possible that the mechanisms I describe in this thesis could be used in even the most primitive vertebrates.

1.3 What Is, and Is Not, Assumed

I have dropped several important assumptions that are common in DRL, but have been obliged to adopt some other assumptions which are *not* common in DRL, and for which DRL provides many suitable techniques for managing, which in the future could allow for a fruitful hybridization. The two major assumptions that I adopt are that the state-space of an agent is fully visible, and that observations, actuators, and dynamics are all noiseless. These seem like severe theoretical assumptions, but there are many powerful denoising methods [44–46], and Takens' embedding theorem [47] tells us that given enough time-delayed observations of even a single scalar-measurement of a dynamical system, it is possible to reconstruct its state-space up to diffeomorphism.

What is *not* assumed in this work is that there are rich external rewards that can guide ethogenesis, nor that there will be externally provided resets of the environment. To me, these two assumptions are the least realistic in common DRL practice, and represent the most serious block to the use of DRL "in the wild". As a corollary, I am studying agents that exist, in a sense, only for a single problem, with no transfer of knowledge between problems, in order to model a total and complete lack of prior knowledge. Obviously, transfer of knowledge will *accelerate* problem-solving, but I am interested for theoretical reasons in understanding the hardest possible scenario.

1.4 How to Read this Thesis

Chapter 2 gives some background information so that the reader has the necessary historical and technical context for this thesis. **Chapter 3** discusses some quirks of behavior, the attempted explanation of which have motivated this thesis. The bulk of the thesis introduces a theory of behavioral search, incrementally building mathematical constructs and concepts and offering intuition along the way, along with offering proofs of important properties of those constructs. In **Chapter 4** I give a general overview of my theory. In **Chapter 5** I define some foundational terminology that later sections build off of. In **Chapter 6** I formally define hyperadaptability, and provide a general recipe for enumerating the set of all behaviors. In **Chapter 7** I introduce segraphs, a data-structure corresponding to a *cognitive graph* that organizes the enumeration. In **Chapter 8** I describe how segraphs self-organize to actually perform the enumeration. These Chapters correspond to David Marr's *computational* level of analysis, the highest level. In **Chapter 9** I describe the concrete ARMS algorithm for performing behavioral search, corresponding to David Marr's *algorithmic* level of description. In **Chapter 10** I provide the details of the neural instantiation of segraphs, corresponding to David Marr's *implementation* level. In **Chapter 11** I present my experimental results of using the algorithm from Chapter 9. In **Chapter 12** I discuss my findings.

Chapter 2

Theories of Ethogenesis

Ethogenesis, or the ontogeny of behavior, was first proposed as a core problem *as such* by the ethologist Nikolaas Tinbergen [48]. As observed in real living creatures, there are a wide variety of ways that behavior in general, and specific behaviors themselves, can develop over time within an organism. The classic dichotomy is between those behaviors which are *innate* to the organism, endowed to it by its genes, and those behaviors which are *acquired*. Ignoring for the moment the fact that for *any* behavior to occur requires that the organism interacts with its environment through metabolism, there is usually no completely neat distinction between innate and acquired behaviors, with a range existing between behaviors that are more obviously innate and more obviously acquired.

On the innate end of the spectrum, animals such as the pied flycatcher bird show a species-typical response to owls, one of their predators, *regardless* of their upbringing, indicating that the behavior is innate [49]. Moving a little along the scale, weasels know when hunting mice to attack and bite them, but must learn through experience *where* and *how* to bite in order to kill the mouse [49]. For "fully" acquired behaviors, it is presumed that there is no genetic information which informs the structure of the behavior. Psychologists and ethologists debate *how much* and *which parts* of *which behaviors* in different animals are innate versus acquired. In modern times this question is recognized as a subtle one within these fields, but in the past they took less nuanced positions, ethology viewing behavior as innate, and psychology as acquired. One particular branch of psychology took an unusually extreme stance on ethogenesis, which ended up having an outsized influence on DRL. We turn now to *behaviorism*.

2.1 Behaviorism

One of the great triumphs of early *experimental* psychology was the discovery of *operant conditioning*, which involves the creation, change in strength, and extinction of *associations*[1] between sensory stimuli and *actions*, where these associations are controlled by *reinforcement*. If the reinforcement is a *reward* then the association is strengthened, if it is a *punishment* it is weakened. Using operant conditioning it is possible to "shape"

[1] This is a central concept in the theory of learning and memory. In brief, an association is some kind of *link* between two mental items or actions, such that the stimulation of the first item more-or-less automatically triggers the stimulation of the second.

an animal's behavior gradually by rewarding the right actions and pushing the wrong ones. Because of this, reinforcement came to be the central guiding principle of the behaviorist school of psychology.

The experimenter does not themselves trigger the behavior that must be reinforced: as B. F. Skinner said, the behavior must happen "for some other reason", which was generally considered to be "random", with the desired behavior emerging from "undifferentiated matter" [50], effectively treating the animal as a *tabula rasa*, a blank slate onto which behavior is "written" by external information. This radically empiricist form of behaviorism took the most extreme possible position on the relationship between operant conditioning and ethogenesis, treating behavior as completely specified by external stimuli and reinforcement, denying any significance to internal "cognitive" processes in the animal.

Thus, operant conditioning was raised to the status of the universal mechanism of ethogenesis. This idea, with the aid of techniques of mathematical optimization, would later form the basis of RL [51]. Though RL has adopted many *non*-behaviorist ideas, embracing internal states and cognition, the original sin remains: even *cognitive processes* are ultimately the result of indirect shaping by *external* reinforcement. In effect, RL is behaviorism, but with extra steps, making an allowance for "internal behavior" that indirectly controls "external behavior".

Notably, this was an extreme position, even within behaviorism, and it had critics from without and within. From within, researchers such as Edward Tolman studied *latent learning*, acquisition of knowledge that occurs *without* reinforcement [52] that nonetheless can help shape later ethogenesis. From without, perhaps the strongest pushback came from linguistics. Behaviorism explained language acquisition in humans in terms of the reinforcement-mediated formation of associations between words and meanings. However, as was pointed out by Chomsky, humans *routinely* produce utterances they have not and can never have heard: the *generativity* of language far outpaces the available *examples* of language [53].

2.2 Hypotheses

This is, in effect, an example of the *problem of induction* [41]. In the general behavioral context, it means that the behavior of an animal is *underspecified* by its *sensory inputs*. In the specific linguistics context it is referred to as the *poverty of the stimulus*. To solve the problem Chomsky hypothesized the existence of a *universal grammar* [54] shared by all humans, which contained parameters that, when specified, would yield the specific grammars of each natural language, allowing a *small* amount of language input to specify the right grammar. The theory remains influential but controversial, especially since no one has figured out what the universal grammar is. That said, a growing body of work suggests that language acquisition is best modeled as a process of *hypothesization* about the grammar that generates linguistic input [55], often under a Bayesian framework [56]. This means that effectively, each speaker *invents* their own grammar, or at least their own understanding of the grammar of their language.

Interestingly, a version of this view has a history outside of humans. In 1929, Lashley noted that rats solving problem-boxes alternated between distinct strategies,

systematically investigating a specific approach for some time in an apparently purposive, rather than random, way [57]. Lashley explicitly compared this to the way that a human solves problems, but lamented his inability to quantify these observations of animal behavior. The challenge was taken up by Krechevsky, who seized upon Lashley's idea and, in a sequence of studies of problem-solving in rats, quantified that they systematically varied their strategy towards a problem, *even in the absence of external reinforcement* [58–62]. Krechevsky used the word "hypotheses" for these systematic dispositions of the rat, controversial at the time, but it would later inspire work in cognitive psychology on hypotheses-based learning in humans [63]. This line of inquiry has continued into the modern day in a deterministic form in active automata learning [64], and in a probabilistic form in Bayesian accounts of cognition and brain function [65]. Bayesian accounts provide a mechanism to determine the probability of some observation given a hypothesis, allowing for different hypotheses to be compared in terms of their likelihood. Some particularly sophisticated methods such as Bayesian program learning [66] use programs as probabilistic generative models for behavior.

Should we think of these hypotheses as innate, or acquired? We know that they are underspecified by *sensory* stimuli, but it seems absurd to say that they are specified by *genes*. Here is one way of thinking about it: the genes are sufficient to specify a self-organizing system, called the brain, which has its own intrinsic dynamics that determine what its motor outputs are, which then *partially* control what its sensory inputs are, which anyway the brain is free to ignore or manipulate as it can [67]. We usually think of the brain as an organ of *computation*, and computation could be arbitrary, leaving us with the question of what computation is being performed. What if we *instead* thought of the brain as an organ of *mathematical sensation*, that interfaces with the world of abstract structure in the same way that the eye, an organ of *optic sensation*, interfaces with the world of electromagnetism? If that were the case, then all that would be necessary for the brain to "acquire" hypotheses would be for the genes to specify the brain as a system that self-organizes into such an organ. A discussion of just *how* this sensation might occur is outside the scope of this thesis.

In any event, this framing lets us bypass would otherwise be a semantic argument over the nature of *learning*. Within machine learning, the arrow of information flow is fairly obvious: information flows *from* a dataset *into* a model in order to specify it. With classically innate behaviors specified by genes, it is just as obvious that information in genes is manifesting and flowing *out* via action. If that were the end, it would be simple, but it is apparent from the ethological study of animal creativity, innovation, play, and exploration [68, 69] that a *considerable* amount of animal "learning" is *self-directed*, confusing the exact source of the information that specifies novel behavior.

One formal theory of ethogenesis that has this flavor comes from the field of *cybernetics* [70], that being Ross Ashby's concept of *ultrastability* [71]. It was cybernetics that originally formulated the view of behavior in a rigorous way in terms of dynamical systems, under which goal-directed behavior can be seen to correspond to an attractor. Ashby viewed *adaptation* in two ways: first, in terms of the existence of an attractor in virtue of which the system is homeostatic to perturbations, and second, in terms of *finding* a dynamical system with the appropriate attractor! His solution to the problem was to build a two-level system: first, a parameterized dynamical system that actually produces a behavior, and second, a dynamical system *for the parameters* which itself

only changes when the first dynamical system does not attract to the appropriate point, meaning that the only stable overall equilibrium is when the first dynamical system attracts to the goal, making it *ultrastable*, able to return to the goal despite arbitrary perturbations. Unfortunately, this return was not directed enough to occur in a timely manner, especially in higher dimensional spaces.

2.3 Deep Reinforcement Learning

Though this more cognitively-oriented approach is still being pursued, it is arguably the behaviorist framing that has been more impactful, via its influence on reinforcement learning (RL). A full history of RL is beyond the scope of this thesis, but in brief, RL owes its genesis to theories of conditioning from psychology, methods of approximation via sampling from computer science, and methods of decision making from optimal control [51]. RL is composed of several distinct approaches. The overarching goal is to select actions from an action space A to change an agent's state in a state-space S such that cumulative reward is maximized. The first broad division is between model-based and model-free RL. In model-based RL, the agent learns a model of the environment's dynamics so that it can predict what the result of its actions will be, and then plan the sequence of actions that maximize its reward. Unfortunately, planning itself is computationally expensive, and if a model isn't available (it usually isn't), it must be learned, which itself is expensive. In model-free RL, no dynamics model is needed: this can be further divided into value-based RL and policy-based RL. In value-based RL, the agent learns the (long-term) value of every state-action pair, so that at every step it can choose the best action given the state that it's in. In policy-based RL, the agent directly learns a function that maps its current state to the best action.

In practice, it is common to hybridize all these methods (indeed, policy-based RL in some sense bootstraps off of value-based RL). At the heart of all forms of RL is the Bellman equation [72], which expresses the value of an action recursively in terms of the expected value of future states, allowing the problem of finding optimal behavior to be decomposed into finding optimal "sub-behaviors". The Bellman equation was formulated to solve Markov Decision Processes [72], and is guaranteed to converge to an optimal policy *when every action can be tried in every state*, which is generally believed to occur only in *tabular* domains such as chess or Go, but becomes more problematic for continuous problems with infinite states and actions. For these problems, various approximations must be made [51]. Even without those constraints, RL suffers from serious sample efficiency issues, in part due to the inefficiency of random or even heuristic exploration [73], though this problem is beginning to be addressed via *planning* methods [74].

The most powerful current model for general behavior learning, deep reinforcement learning (DRL), uses deep learning (DL) methods to approximate the value and policy functions. In brief, DL uses deep neural networks (DNNs) as a class of parameterized functions that can be used to approximate other unknown functions, from which we have samples of corresponding inputs and outputs. The parameters are modified via gradient descent over an error function which is minimal when the DNN exactly replicates the mapping implied by the samples, with the hope being that the DNN will

"generalize" to the unseen elements of the function. DNNs themselves are composed of alternating affine and non-affine transformations. What is significant about this is that DNNs themselves must be trained slowly [75], suffer from poor and unpredictable generalization [76], and require tremendous amounts of data [77]. As reinforcement learning is already sample inefficient, DRL faces sample inefficiency both in its fundamental method and in its approximate implementation with DNNs [78, 79].

2.4 Search

RL systems can take advantage of reward information to guide them to an appropriate behavior. However, not all problems have rich external reward structure, necessitating internally motivated drives such as curiosity [35]. These drives are usually of a heuristic nature [74], oriented towards maximizing information gained. This is obviously a huge step in the right direction, but as I've already articulated, doesn't necessarily guarantee that a solution is found. To actually guarantee that a solution is found requires a *search* that is *complete*, meaning that every behavior is eventually tried. Ashby's system performed a search, but it had no way of being biased by accumulated knowledge. Reinforcement learning provides a means of accumulating knowledge, but in general no way of guaranteeing a complete search, especially not in continuous spaces.

Search itself has a long history in the field of artificial intelligence in finite, discrete domains [80]. However, explicit search algorithms were believed to be inapplicable to continuous domains, prompting the development of local heuristic methods such as gradient descent that can be used to search the parameter space of a DNN. Outside of DL, search continues to play a strong role in robotics, applied to navigation and motion planning by algorithms such as Rapidly-exploring Random Trees (RRT) [81] and Probabilistic Roadmaps [82], though these aren't meant to be models of cognitive processing. Another approach to search is to use evolutionary algorithms [83], which take inspiration from natural evolution by maintaining a population of solutions that can be mutated and selected. Like gradient descent, evolution is not guaranteed to be "complete" in the sense of searching the entire space, but *unlike* gradient descent, evolutionary algorithms are capable of taking sudden leaps through solution-space. A huge variety of methods exist for guiding evolutionary algorithms, but one of particular interest is novelty search [84], which eschews optimization of a specific loss and instead pursues novelty for the sake of open-ended development.

Another way of describing a search process is as an *enumeration* of the set of interest, which is to say, *putting it in a list*. The canonical example is the natural numbers $\{0, 1, 2, 3, ...\}$, which have an obvious enumeration induced by the *successor function*. Effectively, all enumerations of any set correspond to a mapping between that set and the natural numbers. It is possible to define these enumerations in a constructivist way. For example, in the set-theoretic definition of the natural numbers, the successor function is defined as $S(n) = n \cup \{n\}$, with $0 = \{\}$, so that $1 = \{0\} = \{\{\}\}$, $2 = \{0, 1\} = \{\{\}, \{\{\}\}\}$, etc. In this way, the *next* number in the set is directly implied by the previous number: it is possible to exactly *construct* it, as the "history" of the number is directly encoded in its definition. We could call such an object an *indexing structure*. Not all sets can be enumerated: for instance, the real numbers

cannot be enumerated [85].

However, the *rational* numbers, which are *dense* in the real numbers, *can* be enumerated [85] using the Cantor pairing function $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ [86], which maps the numerator and denominator of a rational number to its place in the enumeration. This is sort of a "global" map that assigns an explicit numeric index to each element. We can re-represent this enumeration by introducing a successor function $S_{\mathbb{Q}}((p_i, q_i)) = (p_{i+1}, q_{i+1})$, which yields the next rational number in the enumeration. Here, we can think of (q, p) as the indexing structure. What indexing structure is appropriate for enumerating the set of *behaviors*, what its properties might be based on the nature of ethogenesis, and what kinds of successor functions could actually enumerate the set of all behaviors? These are the questions we turn to now.

Chapter 3

Some Ethogenetic Puzzles

Assuming that our agent can perform a complete search over the set of all behaviors, controlling the order in which behaviors are tried is equivalent to choosing an enumeration of that set. Ideally, this will not require our agent to have an infinitely long list in its head. Instead, we want to have some kind of structure that acts like an *index* in the space of all possible behaviors, a *behavioral index structure* (BIX). In order to perform the search, we need a function, analogous to the *successor function* that defines the natural numbers, which can *increment* the BIX. What could be the nature of the BIX and its corresponding "successor function"? I have already answered in the first chapter that it is a kind of *graph*. However, rather than declare it, I will here try to show that the properties of the BIX and its successor function are natural, or at least reasonable, consequences of some ethological and psychological puzzles.

3.1 Properties of Organic Behavior

Many aspects of organic behavior are puzzling, but not all of those aspects are equally puzzling from all points of view, and I am certainly not going to give a complete list. Rather, these are aspects of behavior that have puzzled *me*, my puzzlement reflecting my own personal biases and history. However, to the best of my knowledge these are real, true properties of behavior, and so the BIX should reflect those properties.

3.1.1 Behavior can Change Quickly

The behavior of an organism can change very quickly. In fact, when faced with a novel problem, it *has* to change quickly, because there simply isn't *time* for it to change slowly. This is what Krechevsky observed in his experiments in rats, and what is observed in modern studies of rats that are placed into more complex, naturalistic environments [87]. To be clear, *skill level* does not change quickly: as pointed out in [88], what changes suddenly is *knowledge* about the problem, sudden insight can alter the overall *pattern* of behavior, *long* before that pattern is "efficient" or "effective". As an example, consider learning to ride a bicycle. While it can take considerable time to master the control of weight and force of the body to effectively balance and steer a bicycle, you immediately generate a novel (but not stream-lined) pattern of behavior that is "approximately the right shape". In the case of riding a bike, this "insight"

usually occurs because someone who already knows how to ride a bike shows you, but as research in animal creativity and innovation have shown, it is also entirely possible for this insight to come "from within" [68, 69].

Puzzle:

When using DNNs trained by stochastic gradient descent (SGD) for *any* purpose, one has to accept that training will be slow. This is a natural consequence of the highly entangled nature of DNN representations: the "burden" of representing any single aspect of something is shared by many parameters, meaning that a desired change of one aspect of a representation necessarily creates undesired changes in *other* aspects [89]. While DNNs can change rapidly, this usually induces *catastrophic forgetting* [90], in which the acquisition of new knowledge in the system erases or scrambles old knowledge. Though there are techniques for overcoming catastrophic forgetting [91, 92], these are techniques for retaining previous learning while *slowly* acquiring new knowledge, they do *not* allow for rapid "saltations" in the network's behavior [93].

Solution:

The BIX is not a DNN trained by SGD, but is instead some other data structure that can change rapidly.

3.1.2 You Shouldn't Repeat Your Mistakes

This is fairly obvious provided that the environment is deterministic and fully observable. If the environment is stochastic and/or partially hidden, then repeating a behavior that has already failed is useful for probing the nature of that environment. However, given those provisions, not repeating what has already failed is the heart of trial-and-error learning. In the words of William James [94], "nothing is more natural than that, when the easiest means of effecting a purpose prove fruitless, other means should be tried. Every failure must keep the animal in a state of disappointment, which will lead to all sorts of new trials and devices; and tranquility will not ensure till one of these, by a happy stroke, achieves the wished-for end." Indeed! As we will see next, simply not doing what has already failed is not, on its own, sufficient to guarantee that you will eventually try something that works, but it is certainly *necessary*. This is, in effect, a novelty drive, such as described in [84].

Puzzle:

Really, it's the same as the last puzzle: because DNNs can only (non-catastrophically) change very slowly, they are prone to repeating the same (or very similar) mistakes for a long time, longer than a human or an animal would in the same situation. As such, DNNs (the premier architecture to creating intelligence) seem to not be a very good candidate for the BIX.

Solution:

Ditto the above solution, with the addendum that the successor function is fundamentally *driven* by the *memory* of failed behaviors, implying that the BIX, in some capacity, remembers which behaviors have already been tried, and whether they have failed.

3.1.3 Behavior Admits Infinite Multiscale Variation

Here, it will be convenient to temporarily adopt some notation. For any behavior b you can imagine, *however* you imagine behavior to be defined, it should be possible to compare that behavior to another, b', and determine how dissimilar they are, $d(b, b')$, *whichever* notion of dissimilarity that you use. Assuming that behavior is occurring in continuous space and time, then for any arbitrarily chosen degree of dissimilarity, ϵ, there will be *uncountably infinitely many* behaviors that are less than ϵ-dissimilar to b. In addition, there are uncountably infinitely many behaviors that are *more* than ϵ-dissimilar to b. In short, for any given behavior, there are uncountably infinitely many "very similar" behaviors, and also uncountably infinitely many "very different" behaviors.

Puzzle:

This becomes a conundrum in the context of the last observation: sure, you shouldn't repeat your mistakes, implying you should try something different. But in combination with *this* observation, we are forced to ask: how different is different enough? And, could we try something that is *too* different? An example can illustrate. Imagine, if you will, trying to squeeze through a gap in an old wooden fence, but getting your shirt caught on a stray nail or burr, stopping you from passing through the fence. A very small adjustment of your trajectory would likely help you succeed, perhaps by simply pushing yourself through the gap a little to the side. Of course, this example leverages your prior knowledge about fences, squeezing, and interpretation of visual and tactile information, the benefit of which our hyperadaptable agent doesn't have! Instead, it simply has to try different "similar" variants until one works.

Now we are faced with a very severe problem: while *you* may know that you can squeeze through the fence-gap with just the right kind of squirming, our *agent* has no such certainty. Suppose that the fence-gap is just too tight, and there is no way through. In that case, all of the uncountably infinite number of minorly tweaked alternative plans our agent could try would all lead to nothing. If this gap is too narrow, you may have to search for a different gap; if none exists, you may have to climb over the fence; if the fence is too tall, you may need to find a ladder or stack some crates to boost yourself over it; if such are not available, you may have to simply find a way *around* the fence; if the fence encloses you, you may need some implement to break through it, which itself may involve a lengthy quest to find a crowbar, axe, or battering ram. Supposing our hyperadaptable agent abandons the initial gap in the fence to begin trying these alternatives, there is a catch: it can't be certain that any of the uncountably infinite "very different" plans will work either: there is still a chance that the only way through is some tightly coordinated wiggle through the

initial fence-gap.

The only way to really ensure that the hyperadaptable agent can guarantee it finds a way past the fence is to basically keep all possibilities open, forever: until it actually finds a solution, it can never completely commit to trying to get through the gap, and it can never completely commit not trying to get through the gap. This logic, of course, applies to all possible plans. If it fails to get through the fence using an axe, well, maybe it just need to swing harder, or at a different angle, or in a different place? If it failed to climb over the fence *here*, what about *there*? What if it changed its climbing style? What if it pole-vaulted?

Solution:

This may seem like a totally unmanageable mess, but there is actually a fairly straightforward solution: provisional generalization. After a plan has failed, *choose* a region of nearby "plan-space" and provisionally *conjecture* that all other plans in that region will also fail. This is effectively an *ad hoc generalization*, with the chosen size of the region corresponding to the degree of the generalization. A priori, there is no way to know if the region was an over-generalization or not, but temporarily, we can use the provisional guess to exclude a local region of plan-space to help our agent focus on more distinct plans that, by virtue of not being similar to a plan that already failed, are more likely to succeed. For each of these more dissimilar plans, the same logic applies: if they fail, we provisionally generalize that failure to a local region of plan-space of a chosen size.

Once all "major" options have been exhausted, it is then sensible to go back and "probe" the agent's initial guesses, basically "splitting" the region and systematically investigating what was before assumed to be a homogeneously bad zone of plan-space. Now, the same logic applies as before, but the *scope* of the agent's new generalization can be smaller, that is, it will generalize failure *less*. If our agent carefully controls its degree of generalization over time, we can ensure a certain "scale-invariance" of our agents exploration, so that there isn't a particular bias towards a specific region of plan-space. In other words, we can use ad-hoc generalization to prevent our agent from getting "stuck in the weeds", and the provisional nature of those generalization prevents our agent from getting lost on a "wild goose chase".

We can conclude, then, that the BIX is somehow able to represent "regions" of behavior-space, these regions can be of different sizes, and larger regions can be refined into smaller ones.

3.1.4 Behavior can be "Good Enough"

In a world with limited time and resources, we very often have to accept imperfect solutions to our problems. As an example, imagine that you are being chased by a rhinoceros, and you come to a large crevice in the ground. Obviously, you jump over the crevice to escape the rhinoceros. In principle, you can imagine an "ideal" jump, one with perfect form and timing, launched at just the right angle, that minimizes energy expenditure, lands you safely on the other side of the crevice, on your feet, with conserved forward momentum so you can keep running, causing yourself no injuries, etc. However, the "ideal" jump is a very specific behavior, one that may not be immediately

accessible to you, if you have not spent many years jumping over crevices while running from rhinoceroses. In this situation, it is obviously more important to get over the crevice, *at all*. Essentially, there is a minimum set of requirements for your jump, or rather, there is an *explicit goal*.

Puzzle:

This probably doesn't really seem like a puzzle at all, except that it differs dramatically from the ontology of RL. The *aim* of ethogenesis in RL is to find the *ideal* behavior, ideal meaning that it maximizes some reward function that implicitly defines the problem to be solved. This is usually only achieved approximately even under the most favorable circumstances, and is nearly impossible to achieve in an unbounded, continuous, unstable world with which we have limited interactions. Defined in this way, there is no clear criterion by which an agent can decide that it has succeeded in completing its task.

Solution:

Rather than define problems *implicitly* via a reward function, I define task-structure *explicitly*, making the aim of behavior to reach a chosen *state* or *region of state-space* from one's current state. The state-space could be the arrangements of a chess-board, the configurations of a robot's servos and sensors, or physical space. In those cases, goals could be any position with your opponent in check-mate, a specific move of a dance, or a used bookstore, respectively. For any value that you might think should be optimized, like time or energy spent, it is possible to fold the accumulation of these values into the state-space, so that part of a goal's definition is to reach a state both in the more conventional sense (being on the other side of the crevice) and in a more extended sense, such as "being on the other side of the crevice and having spent less than 100 Joules of energy".

This gives us our operational definition of a *problem*: being in one state, and wanting to be in another. From the perspective of the busy agent trying to stay alive, *any* behavior that solves the problem will suffice [95], at least for now. If there will be many more interactions with future instances of the problem, then *optimality* becomes a laudable aim to strive for, but before then it is folly. Indeed, this explicit definition of problems is also used in RL practice, manifesting as a *termination* flag that triggers when the agent reaches a terminal state, indicating that the episode has completed with the problem being solved. This means that RL represents each problem in both an explicit and implicit way. However, the implicit component is what RL practice and theory focus on. Whatever implicit problem you could imagine from RL, such as "jump as far as possible", will in my formulation be replaced with an explicit problem, such as "jump over this crevice".

3.1.5 Behavior is Compositional

To begin, I have to clarify that I am using the word "compositional" to be effectively synonymous with the words "composable", "decomposable", and "modular", which is to say, when I say something "is compositional", I only mean that it involves the

Figure 3.1: A green penguin wearing a sombrero.

relationship between big things and the little things they are made out of or can be split into. In essence, I am referring purely to *structural compositionality*. This meaning is fairly common in artificial intelligence [96] and mathematics [97], but is substantially looser than that used in linguistics [98], where a system is only said to be compositional if the *meaning* or *function* of a system follows directly from the structural compositionality of its components and *their* meaning or function, a definition which *precludes* emergence, which *my* definition does *not*.

Behaviors can be split into shorter "sub-behaviors", and many different behaviors can be combined together into one larger behavior. This occurs most obviously in language, where words can be combined into sentences, which can be combined into paragraphs, which can be combined into chapters, which can be combined into a thesis, etc. Artistic behaviors such as music, drawing, and dance all provide excellent examples, in which notes, strokes, and physical motions can all be combined into a larger behavior. It is reasonable to suppose that the compositionality of behavior is a result, in part, of the compositionality of the mental structures that *generate* behavior, so properties of the one should be analogous to properties of the other. With this in mind, consider the green penguin wearing a sombrero in figure X. While this picture has specified one such entity, it is easy to imagine other instantiations (different penguin species, different shade of green, different sombrero), it is easy to further specify it (green penguin wearing a sombrero while playing the saxophone, the sombrero has a little hole in it out of which is peering a smaller penguin, etc.) or alter it completely, i.e. change green for magenta, the sombrero for a top-hat, the penguin for a koala, etc.The ease with which you can alter your internal specification of a compositional representation indicates that there is a high degree of *arbitrariness* and *flexibility* in the representation of mental objects, and consequently, of behaviors.

Puzzle:

This poses a conundrum from several different perspectives. First, from a DL perspective, compositionality is a desirable but difficult-to-achieve property of internal representations, and thus behavior. As has been pointed out in [89], DNNs do not tend to produce nicely factorized representations that would support such a high degree of flexible compositionality. When compositionality is achieved in the vector-space representation that DNNs use, it can happen in one of two ways: either through producing a sort of semantic "grid" [99], or through a sort of embedded "fractal" in the initial state of an RNN, which maps different points on the fractal to different compositional output sequences [100]. While these do clearly allow for compositional representation and behavior, it is less clear to me that these can be flexible in the way that the green penguin example illustrates. Indeed, the systems that can most closely approximate this example are modern image-generators that use the transformer architecture to *iteratively* map from the *non-parametric* space of *strings* to images. The compositionality of the images comes directly from the manifest compositionality of the words, sentences, and paragraphs in the transformer's context-window!

Second, within ethology and, to a large degree, within RL and DRL, there exists the notion of "motor primitives" [101, 102], which can be combined in arbitrary ways to produce complex behaviors. The puzzle from this perspective would be "what are the primitives?". In our penguin example, you might imagine that the hat is a primitive, but then it seems that the hat itself can be subject to endless combination, decomposition, property modification, part swapping, etc. A more transparently "motoric" example would be the arbitrary and convoluted locomotory gaits produced by the employees of the Ministry of Silly Walks [103]. For any "silly walk" you can imagine, if you were to hypothesize specific sequential elements of that walk as primitive, it would be possible to take those parts and *complexify* them, necessarily "breaking" the primitives into... more primitive primitives, and so-on.

Solution:

To resolve this puzzle, we have to synthesize the solutions to several other puzzles, as well as their consequences. First, we have established that behavior is goal-oriented, in the sense of having an explicit goal. We also know that behavior is compositional, in the sense that it can be decomposed into arbitrary sub-behaviors, which themselves can be re-composed into new behaviors. As behaviors, they must *also* be goal-oriented. In effect, this means that any behavior must correspond to a sequence of sub-goals. We also know that our agent is trying different behaviors until one works, implying that a behavior could fail, in other words, not reach its goal.

Logically then, behaviors have at least two components, one that is *aspirational*, what the agent *hopes* will happen, and the other *actual*, what occurs when the behavior is actually *tried*. So, the goal is part of the definition of the behavior, and behaviors can have sub-goals. It's only a tiny leap to suggest that a sequence of sub-goals *is* the aspiration for the behavior, i.e. a *plan*, and that some low-level controller tries to reach consecutive sub-goals in order to execute the plan to produce the *actual* behavior, a *trajectory* through some *state-space*. This is actually a pretty widespread idea in the field of robotic control [81].

From the perspective of compositionality, this solves some problems. By representing a behavior (or rather, a *plan* for a behavior) as a sequence of sub-goals, arbitrary changes to a behavior can be transparently achieved by arbitrarily modifying the sub-goal sequence by removing, adding, swapping, or inserting sub-goals or even whole sub-plans. A "simple" behavior has few sub-goals, so the shape of its trajectory is primarily determined by the low-level controller that is executing the behavior, while a "complex" behavior has many sub-goals, so the shape of the trajectory can be pulled around in arbitrary and convoluted ways. I will call the simplest plans, which just have a start-state and a target-state, "1-plans", as they only involve a single transition.

One major benefit of this notion of behavior is that it allows for precise control over the localization of failure, meaning that when a plan fails, it is trivial to determine *why* it failed, because a specific component 1-plan will have failed. This effectively dissolves what is called the *credit-assignment problem* in RL, a consequence of representing behaviors not in terms of a *state-space* trajectory, but instead in terms of an *action-space* trajectory. One virtue of the action-trajectory notion is that it is direct, in the sense that we usually allow an agent to freely choose its actions, but we don't allow it to freely choose its state, whereas the state-trajectory notion requires the intermediary of a low-level controller. However, many methods exist, including various RL methods, for acquiring such a controller.

Any given plan can technically be part of an infinite number of other "larger" plans. If a given plan has failed, we can be sure that no other plan containing it as a sub-plan will succeed, and likewise, if a given plan has *succeeded*, we at least know that it will not be at fault for any failure in a plan containing it. This is incredibly powerful! Due to the compositionality of plans, we can learn about the feasibility of an infinite number of plans simply from the failure of a single plan! As an example, let's say that I'm trying to make some tea, and I get the strange idea that putting water in the freezer will boil it. To my surprise, that part of the plan fails. Now since I know that any plan that includes boiled water will fail if it uses the "freezer" sub-plan, when I later want to try and make spaghetti, I know to try something different.

This also has some implications for the BIX. We know that the BIX has to remember the failure or success (feasibility) of every plan tried, but actually, it only has to *directly* record the feasibility of the *simplest* plans (1-plans) that have been tried, as the feasibility of every "complex" plan can be directly inferred from the feasibility of its constituent 1-plans. Furthermore, even though there are infinitely many possible plans, our agent will only ever try a finite number of plans, so we can infer that the BIX itself will be finite, but growing. Furthermore, it makes sense that when a 1-plan works it should be re-used, when appropriate, in more complex plans. In addition, we know that complex plans are "chains" of sub-goals, so if 1-plans are shared, it implies that multiple 1-plans will share either their start state or their target states, implying that the set of 1-plans stored by the BIX *forms a network*. In other words, the data-structure that the BIX corresponds to has the properties of a *graph*, with vertices corresponding to sub-goals, and edges corresponding to 1-plans, storing statistics about the feasibility of those 1-plans.

Recall from the solution to the puzzle resulting from the observation that "Behavior Admits Infinite Multiscale Variation" that the BIX also represents "regions" of behavior-space for generalization: in combination with what we have just deduced

about the BIX, we can say that the "sub-goals" are actually more accurately thought of as sub-goal *regions*: larger regions correspond to bolder and broader *provisional* generalizations, while smaller regions correspond to more cautious and narrow provisional generalizations. This means that each edge actually corresponds to a *bundle* of 1-plans: our agent generalizes about *all* of those 1-plans from observing the outcome of trying only a *few* 1-plans. If an edge has always succeeded, then it can be left alone: if it has always failed, there isn't much reason to suspect that deeper probing would yield a benefit. If there has been a mixture of successes and failures, and there are no *better* options, it could make sense to take that edge, an infinite bundle of 1-plans, and *refine* it to see exactly which plans work and which don't. Conversely, adding new sub-goal regions outside the graph *extends* the range of possible courses of action our agent can consider.

This formulation bears a resemblance to the *options* framework [104], with the major difference being that rather than trying to learn a controller that can achieve a specific set of goals, we *assume* a controller, and figure out which "options" that controller can achieve, and achieve more *distant* goals by daisy-chaining options together into complex behaviors.

3.2 Representation of Graphs

There are several different schemes for representing graphs in a neural system. There has been for a long time substantial interest in getting recurrent neural networks (RNNs) to reproduce the dynamics of finite state machines [105–107], which themselves correspond to a type of labeled directed multigraph. This does work, but suffers, as do all methods in DL, from the problem of very slow training, making graphical representations by RNNs (at least by training via gradient descent) an ineffective tool for representing graphs that are capable of *rapid change*, as we know we must have. A related variant to this method is the construction of dynamical systems whose attractors form a heteroclinic network [108], which can be achieved either through learning [109] or by explicit construction [110]. The only problem with this approach is that in my experiments I already have to perform numerical integration for the physical behavior, and *also* performing numerical integration for the neural substrate would have resulted in very slow experiments. In a system without those constraints, these would be very attractive methods.

Another prominent family of technique for representing structured relationships in general, and graphs as a specific example, is *vector-symbolic architectures* (VSAs). The original VSA is Smolensky's tensor-product representation [111], which tensors to represent concepts, and tensor-products to represent the *binding* between concepts. While fully general, binding produces new representations that are multiplicatively larger, making direct comparison between representations at different "levels" difficult. To resolve this, a whole host of fixed-sized VSA methods have been developed [112]. I will only mention one as an example, Plate's Holographic Reduced Representations (HRRs) [113]. In this scheme, to bind two vector representations together, one circularly convolve them, which produces a new vector of the same size as the originals. Circularly correlating the composite vector with either constituent retrieves a noisy

version of the other constituent, allowing for "unbinding". HRRs, and VSAs in general (including the tensor-product representation), suffer from one basic issue, which is that while they enable a complex composite symbol to fit into a single vector, *modification* of that symbol requires it to be "unpacked", which is cumbersome, especially since doing so requires independent knowledge of its structure. This makes VSAs a poor choice for representing our indexing structure for behavioral space.

Graph neural networks [114] do not actually *represent* a graph: they instead define a neural network that acts on vectors associated with vertices of a graph, with each edge being modeled with the same DNN. Thus, graph neural networks act as models for the *dynamics* of a problem that involves many parts interacting in identical ways, rather than as a direct representation of graphical structure itself, which is encoded *outside* the neural network.

Another option is to use a *hetero-associative memory system* (HAM) [115]. A HAM is simply a memory system that is able to associate different items together, similar to an associative array: this is in contrast to an *auto*-associative memory system, which is able to retrieve a full pattern from a corrupted or partial pattern. Running this with analogy, we may say that a HAM stores association between a *key vector* and a *value vector*. As an example, one kind of HAM called fast-weight memory [116] uses an outer product between the key and value vectors to store their association, similar to the tensor-product representation. To store multiple associations, these tensors (in this case, must matrices) are simply added together. If the key-vectors are orthogonal, then these associations will minimally interfere with each other. If every pair of vertex and edge is given its own orthogonal vector, then it is possible to store every edge of a graph in a HAM, selectively retrieving the "target" of an edge by constructing the appropriate key vector.

The benefit of using a HAM, especially one that allows for one-shot memory updates, is that it gives us exquisitely flexible neural representations of compositional structures. Complex objects, such as the BIX, can be modified at any point, and in fact, if we somehow had multiple BIXs, we could potentially *combine* them by simply forming an association between a node of one and a node of another. This level of flexibility can only be achieved, to the best of my knowledge, with a HAM.

3.3 Cognitive Graphs and Cognitive Maps

This scheme I have described, of the BIX being a kind of graph with vertices that correspond to regions of state-space, is not purely pulled from my imagination. Actually, it corresponds rather directly to the concept of a *cognitive graph*, an idea first proposed by [117], who were inspired by work on human memory and animal behavior. These cognitive graphs could be considered to be a more specific and structured version of the idea of a *cognitive map*, proposed by Edward Tolman (a teacher of Krechevsky) in his 1948 paper "Cognitive Maps in Rats and Men" [52]. This idea still receives considerable attention [12], with the overall vision being that cognitive maps and graphs help structure an animals learning and inference [118, 119]. This is a very empiricist description, which you may guess I do not fully endorse.

For for many years after Tolman first hypothesized their existence, it was unknown

how or where in the brain cognitive maps (or graphs) might be implemented. It was with the discovery of *place cells* [120] in the hippocampus and later *grid-cells* [121] in the nearby entorhinal cortex that this question was largely resolved. Place cells, pyramidal cells in the hippocampus, were found to fire whenever the rat was in a specific region of space, called the cell's *place field*, in analogy to the *receptive fields* of *visual neurons*. Thus, place cells were found to coarsely represent the location of the rat. These fields are induced by the place cells' connectivity with grid cells, which rather than showing a spatially localized firing response, instead show a *periodic firing response*, each cell having a firing field with roughly the shape of a hexagonal lattice, different cells having different phase-offsets, orientations, and frequencies. Together the activity of many grid cells constructively and destructively interfere to produce the localized response of the place cell.

When the rat moves around its environment, it moves from place to place, causing different place cells to fire in sequence. Later, sequences of place cells that fired during wakefulness will fire again during sleep, a phenomena called *replay* [122], a word that we will come to find was poorly-chosen. However, these replay events provide a mechanistic explanation for the hippocampus' role in episodic memory, which had been earlier established in the study of humans [123]. Because they happen during sleep, replays are implicated in the *consolidation* of memory [122]. Replay events were eventually found to occur during wakefulness as well [124], while the rat was sitting still, as if the rat were recalling what it had just done. Soon, people observed sequences that were the *reverse* of those experiences, so-called *reverse replays* [125]. Strange, but somewhat understandable, a sort of "working backwards" from a goal-location, perhaps, which has been interpreted in terms of value-updates for RL. Eventually, it was discovered that some "replays" were actually composed of snippets of sequences stitched together, which had never actually been experienced by the rat [126]! Even worse, it was found that some sequences experienced when first exploring the maze had been played *before* the rat had ever entered the maze, called *preplay* [127]!

Some "replay" events strongly resemble planning, for example at a branch-point in a maze, a rat will often stop and sit for a second, while in its hippocampus, a sequence of place cells will fire starting from the rat's current location down the left branch, followed by a similar sequence down the right branch, followed by an apparent decision by the rat to go either left or right, often called *vicarious trial-and-error* [128]. While the conclusion is not irrefutable, many people take this evidence to indicate that place-cells in the hippocampus encode a graphical representation of a problem, which is "attached" to a "metric" representation of the same by the grid cells in the entorhinal cortex. Place-cells receive inputs from grid cells (localizing them) and other regions (context, sensory cues) and form connections with each other to encode the topology of the environment. This allows the hippocampus to stitch together different episodic memories into a comprehensive "map" that allows the animal to learn, reason, and plan efficiently.

While experimental research on cognitive maps and graphs has focused on spatial navigation in rats, it has not been restricted to it. Indeed, cells in the hippocampus have been found that respond to sensory stimuli such as smells [129], and social cues such as familial or hierarchical relationship [130]. Cells with the same firing properties as found in hippocampus and entorhinal cortex have *also* been found in the parietal

lobe [131], grid-like encodings of visual space were found in entorhinal cortex [132], spatially-tuned cells have been found in the visual cortex [133], and grid-like codes for abstract stimuli dimensions such as the lengths of bird legs and necks have been found [134]. Tolman believed that cognitive maps might be used to organize more abstract forms of behavior and learning, and he was not wrong, as cognitive graphs for non-spatial tasks have been found in orbitofrontal [135, 136] and anterior cingulate [137] cortex. Additionally, the hippocampus is part of the archicortex, an ancient region of the brain that is at least 450 million years old, and is shared with our most distant vertebrate cousins, the teleost fish[1], which have a structure homologous to the hippocampus that exhibits similar patterns of activity, and which is used for similar navigation and problem-solving tasks [138–140].

3.4 Summing Up

So, in-brief, the Behavioral Index Structure (BIX) is a graph, with vertices that correspond to regions of state-space, and edges that remember the feasibility of 1-plans between their source and target regions, providing a mechanism to provisionally generalize about *all* 1-plans therein. The provisional nature of these generalizations warrants our agent to *avoid* regions of plan-space that have failed so that it can focus on more promising alternatives, but if that promise is hollow, the agent can always go back and refine previously failed regions so that it can search more carefully for a solution. By adding new vertices "outside" the graph, the agent can explore totally new behaviors. A behavior, or rather, a plan for a behavior, is just a sequence of sub-goals, which corresponds to a path over the graph. When a plan fails, it is simply a matter of finding a different path over the graph that avoids the edge that just failed. In the next chapter, I will provide a more formal and detailed overview of my theory of behavioral search.

[1] The ray-finned fish, rather than our nearer relatives the lobe-finned fish.

Chapter 4

Overview of Behavioral Search Theory

Our goal in chapters 5, 6, 7, and 8 is to lay out, in detail, a theory of how an agent can eventually solve any problem, regardless of prior knowledge. To do this, we have to imagine what it means to have *no* prior knowledge. Any random behavior that an agent tries is almost sure to fail: how can the agent, from so little, eventually find a solution to its problem? The key is in the word "find": as the agent has no prior knowledge, it cannot rely on simply *knowing* the solution, it has to perform a *search* for a solution.

4.1 Problems, Behaviors, and Solutions

We begin by establishing the formal setting we are working with. Imagine an agent in a statespace; the statespace could be physical space, it could be the configuration space of a body, it could be an abstract space. The agent has a *problem* whenever it is not currently in the state it wants to be. Under this view, a *solution* is simply a behavior that takes the agent to its goal. To formally characterize a behavior, we make a distinction between a *plan*, or high-level strategy for reaching a goal, and an *execution*, the physical trajectory generated by following the plan using a controller. This endows behaviors with a teleological aspect, allowing our agent to try a behavior and either *succeed* or *fail*. To solve a given problem merely requires that a single solution be found, so all plans that solve the same problem are considered to be equivalent.

4.2 Hyperadaptability and Behavioral Search

For an agent to be *hyperadaptable*, it must be able to find a solution to any problem in finite time. Doing this without prior knowledge seems to necessitate an exhaustive search over the set of all plans, modulo the equivalence of solutions to the same problem. But plan-space is uncountably infinite! Thankfully, we can construct a set of plans that is countably infinite and dense in the set of all plans, having much the same relationship as that between the rational and real numbers. Using this set of "rational" plans, we can approximate to arbitrary precision any "real" behavior. Ultimately, this

construction involves systematically growing a set of waypoints, corresponding to subgoals of the agent's plans, which in the limit is dense in the agent's statespace. Addition of waypoints can occur in two qualitatively distinct ways: extension and refinement. Extension operates by adding waypoints that are "far" from existing waypoints, having the effect of enabling dramatically different behaviors. Refinement operates by adding new points that are "close" to existing waypoints, having the effect of enabling more precise variation on existing behaviors.

4.2.1 Segraphs

To regulate the enumeration of plans by taking into account factors such as the simplicity, generality, and probably feasibility of a plan, we track the observed number of successes and failures for each 1-plan between hyperballs. We can think of each ordered pair of hyperballs as corresponding to an *edge* that stores this information, with each hyperball then corresponding to a *vertex*, thus giving us a graph that segments state-space, a *segraph*. Paths over the segraph correspond to infinite bundles of plans, from which individual samples can be sampled in order to estimate the robustness and reliability of a plan. In order to make mutation of the segraph responsive to the needs and goals of the agent, *where* segraph mutation occurs depends on *where* the agent is. Since where the agent is in the segraph is determined by which paths it selects, and which paths it selects is a function of the structure of the segraph, there is a tight usage-mutation feedback loop. This feedback loop must be properly balanced to ensure hyperadaptability.

4.3 Segraph Self-Organization

A naive enumeration, while technically possible, would be extremely inefficient. There is at least one simple and problem-agnostic way to bias the search over behaviors *without* making it incomplete, which relies on the fact that many plans can share the same subplan. If a subplan fails, any plan containing that subplan will also fail, and any *other* plan containing a *similar* subplan will be *likely* to fail. Lacking prior knowledge, our agent has no idea what the relationship between subplan similarity and subplan failure likelihood is, so it is forced to make a non-binding *guess* about the appropriate scale of generalization. After a subplan fails, the agent will temporarily generalize the failure to similar subplans and avoid re-using them, *until* other options have been exhausted. At this point, the agent may revisit failed subplans in more detail.

4.4 Notational Conventions

Scalars are denoted by normal lower-case italic letters (a), vectors by bold lower-case italic letters ($\boldsymbol{a}$), while matrices are denoted by bold upper-case letters ($\mathbf{A}$). Ordered sequences (and conceptually adjacent objects) are denoted by lower-case Fraktur-font letters ($\mathfrak{a}$). Sets of such sequences are denoted by upper-case Fraktur-font letters ($\mathfrak{A}$). If $\mathfrak{a}$ is a sequence, or $\boldsymbol{a}$ is a vector, then $\mathfrak{a}[j]$ and $\boldsymbol{a}[j]$ are the 0-based j$^{\text{th}}$ index of $\mathfrak{a}$ and $\boldsymbol{a}$, respectively. For finite-length $\mathfrak{a}$, if $j < 0$ we let $\mathfrak{a}[j] = \mathfrak{a}[|\mathfrak{a}| + j]$, so that $\mathfrak{a}[\text{-}1]$

is the last item of $\mathfrak{a}$, $\mathfrak{a}[\text{-}2]$ the second-to-last item of $\mathfrak{a}$, etc. When appropriate, we will sometimes extend this notation to subscripts, so we may write that a sequence $\mathfrak{a} = (a_0, a_1, a_2, ..., a_{\text{-}1})$, with $a_{\text{-}1}$ indicating the last element of the sequence. If we have a set of sets $A = \{A_1, A_2, ...\}$, then we let $\overline{\overline{A}} = \bigcup_{A' \in A} A'$. We use the same notation to refer to a union of elements (sets or otherwise) of a sequence.

Chapter 5

Problems, Behaviors, and Solutions

Overview

An agent inhabits some kind of state-space; a problem is simply the task of getting from one state to another (a goal). We model goal-directed behavior with *plans* (sequences of subgoals or *waypoints*) executed by a low-level controller. A plan that when executed reaches the goal is a solution; if it still succeeds under bounded perturbation, it is *robust*. Over the entire set of possible robust plans (robust plan-space), we define solutions to be equivalent if they solve the same problem. This equivalence relation partitions the robust plan-space into *achievement classes*, so solving a problem reduces to finding just one representative from the class corresponding to that problem.

Notation

$\mathcal{A}$	An agent
S	A Euclidean **state-space**
S_0	The subset of S physically accessible from the agent's starting state
$\boldsymbol{s}, \boldsymbol{s}^\star$	A state in S, and a goal-state in S (together, a **problem**)
K	A stateful, goal-oriented **controller** that tries to reach a goal-state
${}^{+}\mathsf{x}(\boldsymbol{h}_t)$	Controller **success** (0 for not yet, 1 for success)
${}^{-}\mathsf{x}(\boldsymbol{h}_t)$	Controller **failure** (0 for not yet, 1 for failure)
$T^K_{\boldsymbol{s},\boldsymbol{s}^\star}$	Termination time (either ${}^{+}\mathsf{x}(\boldsymbol{h}_t) = 1$ or ${}^{-}\mathsf{x}(\boldsymbol{h}_t) = 1$) for controller K trying to reach $\boldsymbol{s}^\star$ from $\boldsymbol{s}$
$\mathfrak{t}_K(\boldsymbol{s}, \boldsymbol{s}^\star)$	Finite physical **trajectory** generated by controller K going from $\boldsymbol{s}$ to $\boldsymbol{s}^\star$
$r_K(\boldsymbol{s}, \boldsymbol{s}^\star)$	**Reachability** for controller K from $\boldsymbol{s}$ to $\boldsymbol{s}^\star$ (failure=0, success=1)
$\mathrm{con}_K(\boldsymbol{s})$	The points reachable by controller K from state $\boldsymbol{s}$
$\mathrm{pre}_K(\boldsymbol{s})$	The points that can be reached from state $\boldsymbol{s}$ using controller K
$\mathfrak{s}$	A **plan**, or sequence of states (waypoints)
$\mathfrak{t}_K(\mathfrak{s})$	Finite physical **trajectory** generated by controller K following plan $\mathfrak{s}$
$r_K(\mathfrak{s})$	**Reachability** for controller K following plan $\mathfrak{s}$ (failure=0, success=1)
$B_\epsilon(\boldsymbol{s})$	A hyperball around $\boldsymbol{s}$ with radius ϵ, $B_\epsilon(\boldsymbol{s}) = \{\boldsymbol{s}' \in S : \|s' - s\| \leq \epsilon\}$

$\mathfrak{S}$	The **set of all plans**. This symbol admits many variations. The set of plans of length n is $\mathfrak{S}^n$, of any length, $\mathfrak{S}^*$. The set of plans with waypoints in the set A is $\mathfrak{S}_A$. The set of robust plans is ${}^+\mathfrak{S}$. The set of plans that start at $\boldsymbol{s}$ and end at $\boldsymbol{s}'$ is $\mathfrak{S}(\boldsymbol{s}, \boldsymbol{s}')$
$\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$	The radius-$\boldsymbol{\epsilon}$ tube around plan $\mathfrak{s}$ of alternative plans
${}^+\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$	The subset of $\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$ that is successful
$r_K^{\boldsymbol{\epsilon}}(\mathfrak{s})$	The fraction of $\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$ that succeeds, if $r_K^{\boldsymbol{\epsilon}}(\mathfrak{s}) = 1$ for $\boldsymbol{\epsilon} > 0$ then $\mathfrak{s}$ is **robust**
S_K	The subset of state-space reachable using the controller K
$\mathfrak{s}_1 \sim \mathfrak{s}_2$	Plans are equivalent if they are both solutions to the same problem, starting at the same point $\boldsymbol{s}^\circ$, ending at the same point $\boldsymbol{s}^\star$, and succeeding, meaning $r_K(\mathfrak{s}_1) = 1$ and $r_K(\mathfrak{s}_2) = 1$. Equivalently, $\mathfrak{s}_1, \mathfrak{s}_2 \in {}^+\mathfrak{S}^*(\boldsymbol{s}^\circ, \boldsymbol{s}^\star)$
$\xi_{\boldsymbol{s},\boldsymbol{s}'}^{S_K}$	Equivalence class of plans from $\boldsymbol{s}$ to $\boldsymbol{s}'$ with waypoints in S_K, equal to ${}^+\mathfrak{S}_{S_K}^*(\boldsymbol{s}, \boldsymbol{s}')$
Ξ_{S_K}	The set ${}^+\mathfrak{S}_{S_K}^*$ partitioned by $\sim$, i.e. the **set of solveable problems** in S_K, or the set of all equivalence classes of plans with waypoints in S_K
$\Xi_{S_K}^{\mathcal{A}}$	The set of equivalence classes of solutions with waypoints in S_K that the agent $\mathcal{A}$ can generate

Imagine that an agent $\mathcal{A}$ inhabits a Euclidean state-space S, upon which it can act through an action space A via some deterministic dynamics. In this work, we do not consider the problem of *finding* nor *accessing knowledge about* such a state-space, nor do we consider the problem of *noise*, instead granting the agent direct access to S and the immediate consequences of its actions on its state. A *problem* is when the agent *wants* to be at a specific state $\boldsymbol{s}^\star \in S$, but is currently in a different state $\boldsymbol{s} \in S, \boldsymbol{s} \neq \boldsymbol{s}^\star$. A *solution* is a *behavior* that takes the agent from $\boldsymbol{s}$ to $\boldsymbol{s}^\star$. In general, not all of state-space may be physically accessible to our agent. We refer to the subset of S accessible from the agent's starting state as $S_0 \subseteq S$. While our agent may have a specific goal it wants to reach, a hyperadaptable agent should be able to eventually find the right behavior to reach *any* goal in S_0.

What exactly *is* a behavior? Ultimately, we will use the word to loosely refer to three different levels of abstraction:

1. A physically executed trajectory.
2. A plan that deterministically generates a physically executed trajectory.
3. A probability distribution over plans.

At first glance, the first sense would seem to be sufficient, but later we will want to speak of specific behaviors "failing" or "succeeding", giving *behaviors* a teleological component that is lacking in a mere *physical trajectory*. Tautologically, a physical trajectory reaches the final point it reaches, it cannot "fail" or "succeed" to reach that point, as there is no intrinsic aspirational quality to such a trajectory: by definition it reaches its final point. In order to grant behaviors a proper teleology, we restrict

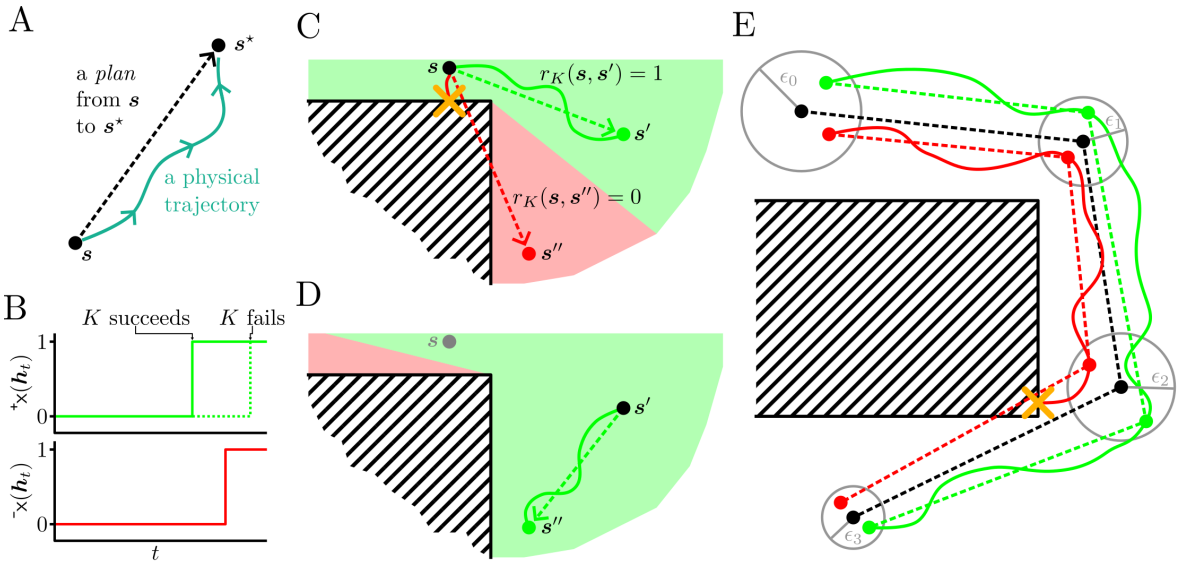


Figure 5.1: (A) A plan is a teleological object, representing an intention to go from one state to another. Using a controller to follow a plan generates a physical trajectory. If the physical trajectory reaches the target, the plan succeeds, otherwise it fails. (B) Termination is ultimately determined by the relative timing of the success and failure functions, $^{+}\mathsf{x}$ and $^{-}\mathsf{x}$. (C) Some points, such as $\boldsymbol{s}'$, can be reached by the controller from $\boldsymbol{s}$. Many points such as $\boldsymbol{s}''$ cannot, however. (D) Some points that cannot be reached in one step can be reached in multiple. If the agent first goes to $\boldsymbol{s}'$, then it can subsequently reach $\boldsymbol{s}''$. (E) Complex plans (black dashed line) can be created by sequentially composing waypoints (black dots), generating more complex physical trajectories. For each plan we can define a neighborhood of nearby plans, called its $\boldsymbol{\epsilon}$-tube. Alternative plans can be sampled from the $\boldsymbol{\epsilon}$-tube. Some may succeed (green behavior), others may fail (red behavior), the more that succeed, the more robust the plan is.

our consideration to behaviors that are generated by a stateful, goal-directed low-level controller:

Definition 5.1 (Controller). A controller is a tuple $(K, {}^{+}\mathsf{x}, {}^{-}\mathsf{x})$, with:

- K a function $\boldsymbol{a}_t, \boldsymbol{h}_{t+1} = K(\boldsymbol{s}_t, \boldsymbol{s}^\star, \boldsymbol{o}_t, \boldsymbol{h}_t)$ which maps a current state $\boldsymbol{s}_t$, a target state $\boldsymbol{s}$, any additional observations $\boldsymbol{o}_t$, and a hidden state $\boldsymbol{h}_t$ to an action $\boldsymbol{a}_t$ and the controller's next hidden state $\boldsymbol{h}_{t+1}$.
- $^{+}\mathsf{x}$ an indicator function that decides if the controller has reached its target. If it has, then $^{+}\mathsf{x}(\boldsymbol{h}_t) = 1$, otherwise, $^{+}\mathsf{x}(\boldsymbol{h}_t) = 0$.
- $^{-}\mathsf{x}$ an indicator function that decides if the controller should *give up* on trying to reach its target. If so, then $^{-}\mathsf{x}(\boldsymbol{h}_t) = 1$, otherwise $^{-}\mathsf{x}(\boldsymbol{h}_t) = 0$.

This definition can be suitably modified for continuous-time domains, which anyways must be discretized for computer implementation. We generally refer to the whole controller $(K, {}^{+}\mathsf{x}, {}^{-}\mathsf{x})$ simply as K for brevity, mentioning $^{+}\mathsf{x}$ and $^{-}\mathsf{x}$ only when necessary.

The start, target point-pair $(\boldsymbol{s}, \boldsymbol{s}^\star)$ corresponds to a very simple plan (Figure 5.1A).

While both $^{+}\mathsf{x}(\boldsymbol{h}_t) = 0$ and $^{-}\mathsf{x}(\boldsymbol{h}_t) = 0$, the controller generates actions to reach the target, generating a physical trajectory or *execution*. As soon as $^{+}\mathsf{x}(\boldsymbol{h}_t) = 1$ or $^{-}\mathsf{x}(\boldsymbol{h}_t) = 1$, the execution is terminated, and the plan either *succeeds* or *fails*, respectively (Figure 5.1B). $^{-}\mathsf{x}$ is defined so that it will always eventually trigger, meaning that every execution ends at some finite time $T^K_{\boldsymbol{s},\boldsymbol{s}^\star} > 0$, called the *termination time*. We denote the execution by K of the plan to reach $\boldsymbol{s}^\star$ from $\boldsymbol{s}$ as

$$\mathfrak{t}_K(\boldsymbol{s}, \boldsymbol{s}^\star) : [0, T^K_{\boldsymbol{s},\boldsymbol{s}^\star}] \to S \tag{5.2}$$

This is a "behavior" in the first sense. We define a *reachability function* $r_{\scriptscriptstyle K}(\boldsymbol{s}, \boldsymbol{s}^\star)$ to indicate whether or not K reaches the goal $\boldsymbol{s}^\star$:

$$r_{\scriptscriptstyle K}(\boldsymbol{s}, \boldsymbol{s}^\star) = \begin{cases} 0, & {}^{-}\mathsf{x}(\boldsymbol{h}_{T^K_{\boldsymbol{s},\boldsymbol{s}^\star}}) = 1 \\ 1, & {}^{+}\mathsf{x}(\boldsymbol{h}_{T^K_{\boldsymbol{s},\boldsymbol{s}^\star}}) = 1 \end{cases} \tag{5.3}$$

See (Figure 5.1C) for an illustration. $r_{\scriptscriptstyle K}(\boldsymbol{s}, \boldsymbol{s}^\star) = 0$ implies the controller K failed to go from $\boldsymbol{s}$ to $\boldsymbol{s}^\star$, while $r_{\scriptscriptstyle K}(\boldsymbol{s}, \boldsymbol{s}^\star) = 1$ implies it succeeded. Our method is comparable to and, formally, a specific manifestation of, the *options framework* [104].

We define $\mathrm{con}_K(\boldsymbol{s}) = \{\boldsymbol{s}^\star \in S : r_{\scriptscriptstyle K}(\boldsymbol{s}, \boldsymbol{s}^\star) = 1\}$ as the set of points that K can reach from $\boldsymbol{s}$, and $\mathrm{pre}_K(\boldsymbol{s}^\star) = \{\boldsymbol{s} \in S : r_{\scriptscriptstyle K}(\boldsymbol{s}, \boldsymbol{s}^\star) = 1\}$ as the set of points from which $\boldsymbol{s}^\star$ can be reached by K. For any sufficiently complex environment, the controller will not be able to reach most targets from most starting points, that is, $\forall \boldsymbol{s} \in S_0, \mathrm{con}_K(\boldsymbol{s}) \subset S_0$ (Figure 5.1C). It may be the case, however, that after reaching a point $\boldsymbol{s}'$ in $\mathrm{con}_K(\boldsymbol{s})$, *new* points are reachable (Figure 5.1D). That is, it is possible that $\exists \boldsymbol{s}' \in \mathrm{con}_K(\boldsymbol{s})$ such that $\mathrm{con}_K(\boldsymbol{s}') - \mathrm{con}_K(\boldsymbol{s}) \neq \varnothing$. If this is the case, then a point $\boldsymbol{s}' \in \mathrm{con}_K(\boldsymbol{s}) \cap \mathrm{pre}_K(\boldsymbol{s}'')$ can act as a *waypoint* between $\boldsymbol{s}$ and $\boldsymbol{s}''$. We call a sequence of waypoints $\mathfrak{s} = (\boldsymbol{s}, \boldsymbol{s}_1, \boldsymbol{s}_2, ...\boldsymbol{s}_{n-1}, \boldsymbol{s}^\star)$ a *plan* from $\boldsymbol{s}$ to $\boldsymbol{s}^\star$ (Figure 5.1E). A plan is a behavior in the second sense.

We denote the composite-execution generated by K following the plan $\mathfrak{s}$ as $\mathfrak{t}_K(\mathfrak{s})$, which like a "single-waypoint" execution, always terminates at some finite time $T^K_{\mathfrak{s}}$. We say that the plan succeeds if every sub-plan succeeds, but if any of the sub-plans fails (a waypoint isn't reached), the entire plan fails, and the execution terminates at a non-target point. We denote success in reaching the final target as $r_{\scriptscriptstyle K}(\mathfrak{s}) = 1$ and failure as $r_{\scriptscriptstyle K}(\mathfrak{s}) = 0$. We encapsulate this information with a "try" function:

Definition 5.4 (Trying a Behavior). We formalize an agent trying a behavior $\mathfrak{s}$ using the *try functions*.

The first function maps a plan to the final state of its execution:

$$\mathtt{try}_{\boldsymbol{s}} : \mathfrak{S}^*_S \to S \qquad\qquad \mathtt{try}_{\boldsymbol{s}} : \mathfrak{s} \mapsto \mathfrak{t}_K(\mathfrak{s})(T^K_{\mathfrak{s}}) \tag{5.5}$$

The second function maps a plan to the subplan (if any) it failed on:

$$\mathtt{try}_{\mathrm{x}} : \mathfrak{S}^*_S \to \mathfrak{S}^1_S \cup \{(\,)\} \qquad\qquad \mathtt{try}_{\mathrm{x}} : \mathfrak{s} \mapsto \begin{cases} (\,) & r_{\scriptscriptstyle K}(\mathfrak{s}) = 1 \\ (\boldsymbol{s}_\circ, \boldsymbol{s}_\triangleright) & r_{\scriptscriptstyle K}(\mathfrak{s}) = 0 \end{cases} \tag{5.6}$$

Where $(\boldsymbol{s}_\circ, \boldsymbol{s}_\triangleright)$ is the subplan $\mathfrak{s}$ failed on *if it failed*, and $(\,)$ is the "empty" plan

returned if $\mathfrak{s}$ succeeded.

We say that a plan $\mathfrak{s}$ is *robust* if every "nearby" plan $\mathfrak{s}$ succeeds. To formalize this, we first consider the set of length n plans with waypoints in S:

$$\mathfrak{S}_S^n = \{(\boldsymbol{s}_i)_{i=0}^n : \boldsymbol{s}_i \in S\} \tag{5.7}$$

We can imagine ascribing to each waypoint $\boldsymbol{s}_i$ in a plan $\mathfrak{s}$ a "tolerance" or "nearness-threshold", ϵ_i, defining the ϵ-neighborhood around $\boldsymbol{s}_i$ as:

$$B_\epsilon(\boldsymbol{s}) = \{\boldsymbol{s}' \in S : \|\boldsymbol{s} - \boldsymbol{s}'\| \leq \epsilon\} \tag{5.8}$$

... a hyperball with radius $\epsilon > 0$ centered on $\boldsymbol{s}$. We can now introduce the $\boldsymbol{\epsilon}$*-tube*:

Definition 5.9 ($\boldsymbol{\epsilon}$-tube). Let $\mathfrak{s} = (\boldsymbol{s}_i)_{i=0}^n$ be a plan, and $\boldsymbol{\epsilon} = (\epsilon_i)_{i=0}^n$ a corresponding sequence of non-negative numbers. Then

$$\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s}) = \{(\boldsymbol{s}'_i)_{i=0}^n \in \mathfrak{S}_S^n : \forall i \in [0\,..\,n], \boldsymbol{s}'_i \in B_{\epsilon_i}(\boldsymbol{s}_i)\}$$

is the $\boldsymbol{\epsilon}$*-tube* around $\mathfrak{s}$, a bundle of "nearby" or "similar" plans. (See Figure 5.1E for example).

$\boldsymbol{\epsilon}$-tubes will play a primarily *epistemic* role for our agent, allowing us to organize our agent's assumptions about the navigability of the state-space S. For now, we will use them to specify what a robust behavior is. Let:

$${}^{+}\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s}) = \{\mathfrak{s}' \in \mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s}) : r_{\scriptscriptstyle K}(\mathfrak{s}') = 1\} \tag{5.10}$$

be the set of all plans in the $\boldsymbol{\epsilon}$-tube of $\mathfrak{s}$ which *succeed*. We extend the function $r_{\scriptscriptstyle K}(\mathfrak{s})$ (the "reachability" of the plan) to its $\boldsymbol{\epsilon}$-tube, and refer to the $\boldsymbol{\epsilon}$*-reliability* of $\mathfrak{s}$ as:

$$r_{\scriptscriptstyle K}^{\boldsymbol{\epsilon}}(\mathfrak{s}) = \frac{|{}^{+}\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})|}{|\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})|} \tag{5.11}$$

which is the fraction of successful plans in the $\boldsymbol{\epsilon}$-tube of $\mathfrak{s}$. It is helpful to consider different possible choices of $\boldsymbol{\epsilon}$: to this end, we construct $\mathfrak{E}^n = \{(\epsilon_i)_{i=0}^n : \epsilon_i > 0\}$ as the set of possible $\boldsymbol{\epsilon}$-sequences of length n. This construction allows us to formally delineate between plans that are *robust* and those that are *not*:

Definition 5.12 (Plan Robustness).

$$\mathrm{R}_{\scriptscriptstyle K}(\mathfrak{s}) = \begin{cases} 1, & \exists \boldsymbol{\epsilon} \in \mathfrak{E}^{\ell(\mathfrak{s})} \text{ s.t. } r_{\scriptscriptstyle K}^{\boldsymbol{\epsilon}}(\mathfrak{s}) = 1 \\ 0, & \nexists \boldsymbol{\epsilon} \in \mathfrak{E}^{\ell(\mathfrak{s})} \text{ s.t. } r_{\scriptscriptstyle K}^{\boldsymbol{\epsilon}}(\mathfrak{s}) = 1 \end{cases}$$

A plan $\mathfrak{s}$ is *robust* if $\mathrm{R}_{\scriptscriptstyle K}(\mathfrak{s}) = 1$, otherwise it is *not*.

Now, we let ${}^{+}\mathfrak{G}_S^n$ be the set of all robust plans of length n, and ${}^{+}\mathfrak{G}_S^*$ the set of all robust plans of any finite length. Formally:

$$ {}^{+}\mathfrak{G}_S^n = \{\mathfrak{s} \in \mathfrak{G}_S^n : \mathrm{R}_K(\mathfrak{s}) = 1\} \tag{5.13} $$

$$ {}^{+}\mathfrak{G}_S^* = \bigcup_{n=1}^{\infty} {}^{+}\mathfrak{G}_S^n \tag{5.14} $$

Then, slightly overloading notation, we say that $\mathrm{con}_K^*(\boldsymbol{s}) \subset S$ is the set of states that can be reached by K via some plan in ${}^{+}\mathfrak{G}_S^*$. If $\boldsymbol{s}_0$ is the initial state of the agent $\mathcal{A}$, then $S_K = \mathrm{con}_K^*(\boldsymbol{s}_0)$ is the subset of S that $\mathcal{A}$ can reach. For the rest of this work, we assume that there are neither "Garden of Eden" nor "black-hole" states[1], meaning that $\forall \boldsymbol{s}' \in \mathrm{con}_K^*(\boldsymbol{s}), \mathrm{con}_K^*(\boldsymbol{s}') = \mathrm{con}_K^*(\boldsymbol{s})$. Any state in S_K is a potential problem that is physically solvable by our agent. An agent that is *hyperadaptable* can eventually solve any problem (reach any state) in S_K, meaning that it must be able to eventually find *a plan that reaches the goal*.

${}^{+}\mathfrak{G}_{S_K}^* \subset \mathfrak{G}_{S_K}^*$ is the set of all possible robust plans with waypoints in S_K, with ${}^{+}\mathfrak{G}_{S_K}^*(\boldsymbol{s}, \boldsymbol{s}')$ the restriction of this set to only plans which start at $\boldsymbol{s}$ and end at $\boldsymbol{s}'$. Formally:

$$ {}^{+}\mathfrak{G}_{S_K}^*(\boldsymbol{s}, \boldsymbol{s}') = \{\mathfrak{s} \in {}^{+}\mathfrak{G}_{S_K}^* : \mathfrak{s}[0] = \boldsymbol{s}, \mathfrak{s}[\text{-}1] = \boldsymbol{s}'\} \tag{5.15} $$

By definition, every plan in ${}^{+}\mathfrak{G}_{S_K}^*(\boldsymbol{s}, \boldsymbol{s}')$ is a viable way to go from $\boldsymbol{s}$ to $\boldsymbol{s}'$, so from the perspective of solving a problem (going from one point to another), every plan in ${}^{+}\mathfrak{G}_{S_K}^*(\boldsymbol{s}, \boldsymbol{s}')$ is equivalent (Figure 6.1A). We say that $\mathfrak{s} \sim \mathfrak{s}' \iff \mathfrak{s}, \mathfrak{s}' \in {}^{+}\mathfrak{G}_{S_K}^*(\boldsymbol{s}, \boldsymbol{s}')$, and refer to the equivalence class ${}^{+}\mathfrak{G}_{S_K}^*(\boldsymbol{s}, \boldsymbol{s}')$ as an *achievement class*, denoted $\xi_{\boldsymbol{s},\boldsymbol{s}'}^{S_K}$. The set of all achievement classes partitions ${}^{+}\mathfrak{G}_{S_K}^*$, yielding a quotient set (Figure 6.1B):

$$ \Xi_{S_K} = {}^{+}\mathfrak{G}_{S_K}^* \backslash \sim = \{\xi_{\boldsymbol{s},\boldsymbol{s}'}^{S_K} : \boldsymbol{s}, \boldsymbol{s}' \in S_K\} \tag{5.16} $$

which effectively represents the existence of solutions to problems inside of S_K, called the *capability set*[2] of S_K. If $\mathcal{A}$ is an agent, and ${}^{+}\mathfrak{G}_{\mathcal{A}}^*$ is the set of behaviors that can be generated by $\mathcal{A}$, then $\Xi_{S_K}^{\mathcal{A}} = {}^{+}\mathfrak{G}_{\mathcal{A}}^* \backslash \sim$ represents the *capability of* $\mathcal{A}$. Notice that we have not yet described *how* $\mathcal{A}$ generates behaviors.

5.1 Notation for Plans

To simplify further exposition, it will help to introduce some notation and terminology related to plans.

We have already used $\mathfrak{s}[i]$ to indicate the i^{th} element of $\mathfrak{s}$, where $\mathfrak{s}[0]$ is the starting point of a plan, $\mathfrak{s}[1]$ is the first waypoint in the sequence, and $\mathfrak{s}[\text{-}1]$ is the final waypoint or "goal" of the plan.

[1] States or regions which once left, can never be returned to, and states or regions which once entered, can never be left.

[2] A grislier but more evocative name would be the *cat-skinning set*.

If we have a plan $\mathfrak{s} = (\boldsymbol{s}_i)_{i=0}^{n}$, then we will say that $\ell(\mathfrak{s}) = n$, so that the "goal" of a plan is $\mathfrak{s}[\ell(\mathfrak{s})]$. Notice that $|\mathfrak{s}| = \ell(\mathfrak{s}) + 1$, because $|\mathfrak{s}|$ is the number of *states* in $\mathfrak{s}$, whereas $\ell(\mathfrak{s})$ corresponds to the number of state-to-state "transitions" represented by $\mathfrak{s}$. These correspond to two different notions of the "length" of a plan, which this notation disambiguates.

If we have a plan $\mathfrak{s}$, and $\boldsymbol{s}$ is one of the states in the sequence, we will say that $\boldsymbol{s}$ is in $\mathfrak{s}$, written "$\boldsymbol{s} \in \mathfrak{s}$". This notation is slightly abusive, as it implicitly casts $\mathfrak{s}$ as a sort of set, which it is not.

If $\boldsymbol{s} \in \mathfrak{s}$, and $\mathfrak{s}[i] = \boldsymbol{s}$, we will say that $\texttt{idx}_{\mathfrak{s}}(\boldsymbol{s}) = i$ to indicate the index of $\boldsymbol{s}$ inside the plan $\mathfrak{s}$.

If one behavior $\mathfrak{s}'$ is contained by another plan $\mathfrak{s}$, we will say that $\mathfrak{s}'$ is a "sub-behavior" of $\mathfrak{s}$, written $\mathfrak{s}' \Subset \mathfrak{s}$, and that $\mathfrak{s}$ "contains" $\mathfrak{s}'$, written $\mathfrak{s} \Supset \mathfrak{s}'$. Formally:

$$\mathfrak{s}' \Subset \mathfrak{s} \iff \exists k \in \mathbb{N} \text{ such that } \forall i \in [0 \,..\, \ell(\mathfrak{s}')], \mathfrak{s}'[i] = \mathfrak{s}[i+k] \iff \mathfrak{s} \Supset \mathfrak{s}' \tag{5.17}$$

If two behaviors $\mathfrak{s}'$ and $\mathfrak{s}''$ are sub-behaviors of $\mathfrak{s}$, and $\mathfrak{s}''$ happens "before" $\mathfrak{s}'$ inside of $\mathfrak{s}$, we say that $\mathfrak{s}''$ is a pre-condition of $\mathfrak{s}'$ within $\mathfrak{s}$, which we write as $\mathfrak{s}'' \underset{\mathfrak{s}}{\prec} \mathfrak{s}'$. Formally:

$$\mathfrak{s}'' \underset{\mathfrak{s}}{\prec} \mathfrak{s}' \iff (\mathfrak{s}' \Subset \mathfrak{s}) \wedge (\mathfrak{s}'' \Subset \mathfrak{s}) \wedge (\forall \boldsymbol{s}'' \in \mathfrak{s}'', \boldsymbol{s}' \in \mathfrak{s}' \text{ s.t. } \texttt{idx}_{\mathfrak{s}}(\boldsymbol{s}'') \leq \texttt{idx}_{\mathfrak{s}}(\boldsymbol{s}')) \tag{5.18}$$

Chapter 6

Hyperadaptability and Behavioral Search

Overview

An agent is defined as *hyperadaptable* if, for every solvable problem, it can find a solution in finite time. Without prior knowledge, this requires search, or *enumeration*. Because plan-space is uncountably infinite, direct enumeration is impossible. Instead, we construct a countable set of "rational" plans that is dense in plan-space. We start with a finite set of waypoints, generating finitely many plans. Each waypoint has a corresponding hyperball, with which we can define *mutation* operations to grow the set of waypoints. A hyperball can be *extended* by adding new larger hyperballs around it, placing waypoints in new regions of statespace, or it can be *refined*, replaced with new smaller hyperballs inside it to increase the local density of waypoints. A *balanced* sequence of these mutations in the limit makes the waypoint set dense in the statespace, ensuring that search over the corresponding plan-set will eventually visit at least one member of every achievement class. Finally, because plans share sub-plans, the agent can intelligently bias the search by deprioritising candidates that reuse failed segments, making an otherwise intractable search a practical problem-solving strategy.

Notation

$\mathcal{S}$	A discrete set of waypoints in the statespace S
E	A function assigning an ϵ to each waypoint in $\mathcal{S}$
B_E	A function assigning a hyperball to each waypoint in $\mathcal{S}$ with radius given by E
$\mathbf{b}_{\boldsymbol{s}}$	The hyperball assigned to the waypoint $\boldsymbol{s}$
$\mathcal{B}$	A set of hyperballs. $\mathcal{B}(\mathcal{S})$ are the hyperballs corresponding to points in $\mathcal{S}$
$\mathfrak{b}$	A sequence of hyperballs, or **path**. If $\mathfrak{s}$ is a plan drawn from $\mathfrak{S}^*_{\mathcal{S}}$, then $\mathfrak{b}_{\mathfrak{s}}$ is the corresponding sequence of hyperballs
$[\![\mathfrak{b}]\!]$	The bundle of plans ($\boldsymbol{\epsilon}$-tube) represented by the hyperball sequence $\mathfrak{b}$
$r_{_K}(\mathfrak{b})$	The $\boldsymbol{\epsilon}_{\mathfrak{b}}$-reliability of $\mathfrak{b}$, where $\boldsymbol{\epsilon}_{\mathfrak{b}}$ is the vector of corresponding hyperball radii. $\mathfrak{b}$ is **robust** if $r_{_K}(\mathfrak{b}) = 1$

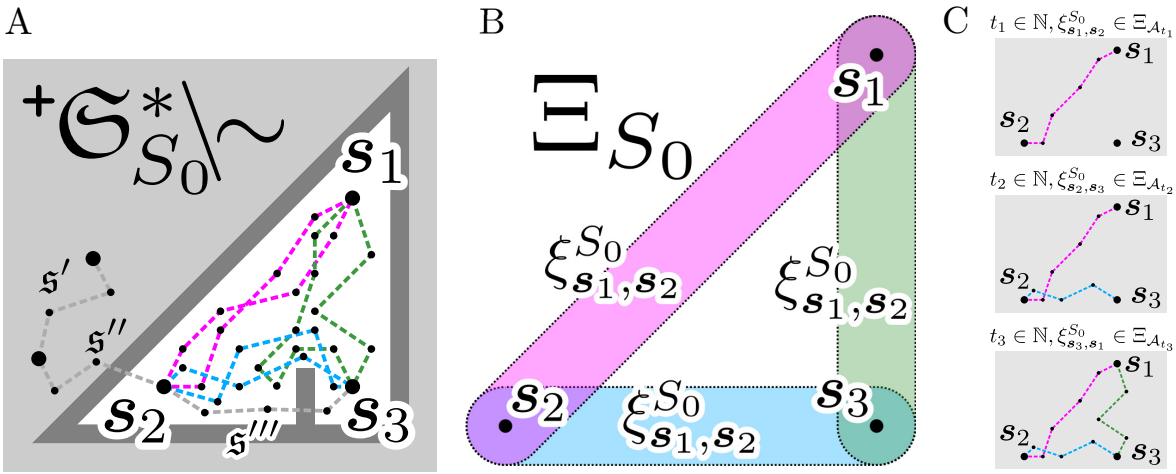


Figure 6.1: The set of all behaviors (plans) through state-space is uncountably infinite, but there is internal structure to this set which can be taken advantage of. (A) The set of interest for our agent is the set of all robust plans of any length with waypoints inside the reachable region of state-space, $^{+}\mathfrak{S}^{*}_{S_K} \subset \mathfrak{S}^{*}_{S_K} \subset \mathfrak{S}^{*}_{S}$, which excludes plans containing waypoints outside of S_K (plans $\mathfrak{s}'$ and $\mathfrak{s}''$), and any plan that fails or otherwise isn't robust ($\mathfrak{s}''$ and $\mathfrak{s}'''$). All plans linking the same two points form an equivalence class (plans in the same equivalence class have the same color). The equivalence relation partitions the set of robust plans through S_K, $^{+}\mathfrak{S}^{*}_{S_K}\backslash\sim$. (B) This partition is equivalent to the set of solvable problems Ξ_{S_K} in the reachable state-space, each achievement class ξ, or set of equivalent behaviors, corresponds to a solvable problem. (C) An agent is hyperadaptable if, for every solvable problem in Ξ_{S_K}, the agent finds in finite time a solution to that problem.

`refine`	The **refinement mutation**, replaces a hyperball with several smaller hyperballs in the same area, increasing the local resolution of $\mathcal{B}(\mathcal{S})$
$\mathfrak{B}$	The **set of all paths**. This symbol admits many variations. The set of paths of length n is $\mathfrak{B}^n$, of any length, $\mathfrak{B}^*$. The set of paths with hyperballs in the set A is $\mathfrak{B}_A$, all paths that are robust is $^{+}\mathfrak{B}$. All paths from $\mathbf{b}_1$ to $\mathbf{b}_2$ is $\mathfrak{B}(\mathbf{b}_1, \mathbf{b}_2)$
$\mathfrak{b}_1 \sim \mathfrak{b}_2$	Paths are equivalent if they are both robust, start at the same hyperball, and end at the same hyperball. Equivalently, $\mathfrak{b}_1, \mathfrak{b}_2 \in {}^{+}\mathfrak{B}^*(\mathbf{b}, \mathbf{b}')$
$\xi^{\mathcal{B}}_{\mathbf{b},\mathbf{b}'}$	Equivalence class of paths from $\mathbf{b}$ to $\mathbf{b}'$ with hyperballs in $\mathcal{B}$, equal to $^{+}\mathfrak{B}^{*}_{\mathcal{B}}(\mathbf{b}, \mathbf{b}')$
$\Xi^{\mathcal{B}}_{\mathcal{B}}$	The set $^{+}\mathfrak{B}^{*}_{\mathcal{B}}$ partitioned by $\sim$, i.e. the set of all equivalence classes of *paths* with hyperballs in $\mathcal{B}$
$\Xi^{\mathcal{B}}_{S_K}$	The set of equivalence classes of *plans* with members inside a path in $^{+}\mathfrak{B}^{*}_{\mathcal{B}}$
`extend`	The **extension mutation**, adds new larger hyperballs around an existing hyperball, increasing the area of state-space covered by $\mathcal{B}(\mathcal{S})$

Before, we gave an informal definition of a hyperadaptable agent as one that can always eventually solve any solvable problem. Now, we can now provide a formal definition of *hyperadaptability*:

Definition 6.1 (Hyperadaptability). Let $\mathcal{A}_t$ be an agent at time $t \in \mathbb{N}$.

$$\mathcal{A} \text{ is } \textit{hyperadaptable} \iff \forall \xi^{S_K}_{\boldsymbol{s},\boldsymbol{s}'} \in \Xi_{S_K}, \exists t \in \mathbb{N} \text{ s.t. } \xi^{S_K}_{\boldsymbol{s},\boldsymbol{s}'} \in \Xi^{\mathcal{A}_t}_{S_K}$$

In other words, an agent is hyperadaptable if and only if for any achievement class, there is some finite time at which that class is part of the agent's capability (Figure 6.1C).

You may notice that this definition makes no mention of any priors (crystal intelligence) or inference ability (fluid intelligence), as indeed, hyperadaptability is independent of these notions. Fluid intelligence might allow a hyperadaptable agent to make more complex inferences about which behaviors could work, prior knowledge might be able to narrow down the set of possible behaviors from the start, but hyperadaptability cannot depend on either of these concepts. In fact, because of this, an agent can only be hyperadaptable if it can perform an *exhaustive search* of $\mathfrak{S}^*_{S_K}$, as otherwise it would be possible for it to miss a necessary behavior. We immediately face a stark problem: $\mathfrak{S}^*_{S_K}$ is uncountably infinite (a property it inherits from S), meaning that it cannot be enumerated and that it is mathematically *impossible* to exhaustively search.

Thankfully, the situation is not hopeless. First, we can construct a countably infinite set of behaviors that is dense in $\mathfrak{S}^*_{S_K}$, which *can*, mathematically, be exhaustively searched. Second, even for this countable subset, our agent only needs to find one behavior for each achievement class: after finding one behavior $\mathfrak{s} \in \xi^{S_K}_{\boldsymbol{s},\boldsymbol{s}'}$, it is not necessary to try any other behaviors in this class. Third, there is significant modular structure in $\mathfrak{S}^*_{S_K}$ that we can take advantage of: basically, every sub-plan of a plan is shared with infinitely many other plans, a fact that can be used to de-prioritize plans that are unlikely to succeed based on their substructure. Ultimately, the necessity to construct a set which is dense in $\mathfrak{S}^*_{S_K}$ and the ability to leverage shared sub-plan structure will provide us with the outlines of a recipe for performing an efficient search across the set of behaviors.

Suppose that our agent has a finite set of waypoints $\mathcal{S} \subset S$ from which it can construct a behavior $\mathfrak{s}$. If this behavior fails, then in the deterministic conditions that we are assuming, the next behavior our agent tries should *not* be $\mathfrak{s}$ again, as it will definitely fail. Suppose that $\mathfrak{s} = (\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3)$, so that $\mathfrak{s}$ naturally decomposes into three "sub-behaviors", $\mathfrak{s}_1 = (\boldsymbol{s}_0, \boldsymbol{s}_1)$, $\mathfrak{s}_2 = (\boldsymbol{s}_1, \boldsymbol{s}_2)$, and $\mathfrak{s}_3 = (\boldsymbol{s}_2, \boldsymbol{s}_3)$ (Figure 6.2A). Suppose that $r_K(\mathfrak{s}_1) = 1$ and $r_K(\mathfrak{s}_2) = 0$, meaning that $\mathfrak{s}_1$ succeeded but $\mathfrak{s}_2$ failed, and that the agent never got a chance to actually try $\mathfrak{s}_3$. By definition, if any sub-behavior of a behavior fails, the entire behavior fails, and since we are assuming that environment dynamics are deterministic, it follows that

1. any behavior containing $\mathfrak{s}_1$ will at least not fail at $\mathfrak{s}_1$:

$$r_K(\mathfrak{s}') = 1 \implies \forall \mathfrak{s} \ni \mathfrak{s}', \mathtt{try}_{\mathsf{x}}(\mathfrak{s}) \neq \mathfrak{s}'$$

2. and any behavior containing $\mathfrak{s}_2$ will fail at $\mathfrak{s}_2$ or an earlier sub-behavior:

$$r_K(\mathfrak{s}') = 0 \implies \forall \mathfrak{s} \ni \mathfrak{s}', r_K(\mathfrak{s}) = 0 \wedge (\mathtt{try}_{\mathsf{x}}(\mathfrak{s}) \Subset \mathfrak{s}' \vee \mathtt{try}_{\mathsf{x}}(\mathfrak{s}) \leftarrow\!\!\prec_{\mathfrak{s}} \mathfrak{s}')$$

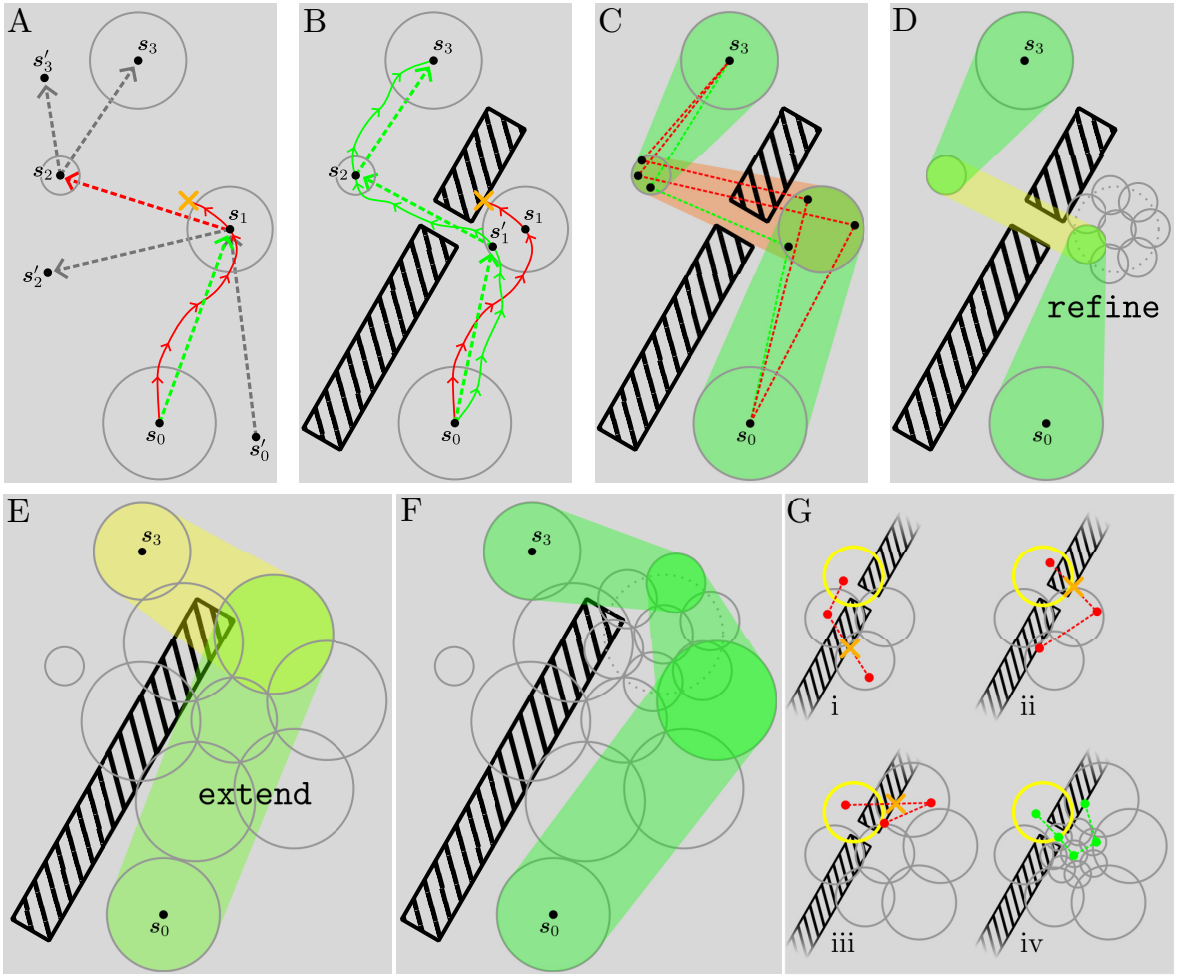


Figure 6.2: Where along a plan the plan fails contains rich information about other plans. (A) If the plan $\mathfrak{s} = (\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3)$ fails between $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, then we know that any other plan that contains $(\boldsymbol{s}_1, \boldsymbol{s}_2)$, such as $(\boldsymbol{s}'_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}'_3)$, will also fail at $(\boldsymbol{s}_1, \boldsymbol{s}_2)$. Conversely, any plan containing $(\boldsymbol{s}_0, \boldsymbol{s}_1)$ will at least not fail at $(\boldsymbol{s}_0, \boldsymbol{s}_1)$, such as $(\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}'_2)$. (B) Just because $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ failed, doesn't mean that other plans inside its $\boldsymbol{\epsilon}$-tube will also fail: it could be that $(\boldsymbol{s}_1, \boldsymbol{s}_2)$ was a "near miss". (C) If there is a solution in $\mathfrak{s}$'s $\boldsymbol{\epsilon}$-tube $[\![\mathfrak{b}_\mathfrak{s}]\!]$, then randomly sampling plans can eventually find a solution. (D) A better way is to replace the single $\boldsymbol{\epsilon}$-tube with several higher-resolution $\boldsymbol{\epsilon}$-tubes by refining one of the waypoint's hyperballs. (E) However, if a valid solution isn't inside the original $\boldsymbol{\epsilon}$-tube, then the agent can add qualitatively new waypoints through extension. (F) Extension followed by refinement can create arbitrarily complex plans. (G) Ideally, the agent should attempt simpler and coarser plans before more complex and fine-grained plans (i → ii → iii → iv).

In other words, under the deterministic conditions we are assuming, if the sub-behavior $\mathfrak{s}'$ failed, it would be senseless to choose any behavior $\mathfrak{s} \ni \mathfrak{s}'$ as our next attempt. Now technically, for any $\boldsymbol{\epsilon} \in \mathfrak{E}^{\ell(\mathfrak{s})}$, any *other* behavior $\mathfrak{s}'$ in $\mathfrak{s}$'s $\boldsymbol{\epsilon}$-tube *could* be successful (Figure 6.2B). Essentially, it's possible that $\mathfrak{s}$ was a near-miss, and if the agent looks for long enough in $\mathcal{N}_{\boldsymbol{\epsilon}}(\mathfrak{s})$ for a plan, it will eventually find the solution (Figure 6.2C).

At this point, it is helpful to change our perspective somewhat: rather than think about *waypoints* with some arbitrary tolerance, we will instead reason about *wayregions* which have an *intrinsic* tolerance. Suppose that each waypoint $\boldsymbol{s} \in \mathcal{S}$ has a fixed ϵ, encoded by a function $E : \boldsymbol{s} \mapsto \epsilon_{\boldsymbol{s}}$. Then each waypoint $\boldsymbol{s} \in \mathcal{S}$ can be treated as a hyperball, encoded by a function $B_E : \boldsymbol{s} \mapsto \mathbf{b}_{\boldsymbol{s}}$, with $\mathbf{b}_{\boldsymbol{s}} = B_{\epsilon_{\boldsymbol{s}}}(\boldsymbol{s})$. For brevity, we will sometimes substitute $\mathbf{b}_i$ for $\mathbf{b}_{\boldsymbol{s}_i}$. We denote the set of hyperballs corresponding to waypoints in $\mathcal{S}$ as $\mathcal{B}(\mathcal{S})$. We extend these functions to sequences, letting $E : \mathfrak{s} \mapsto \boldsymbol{\epsilon}_{\mathfrak{s}}$, with $\boldsymbol{\epsilon}_{\mathfrak{s}} = (\epsilon_{\boldsymbol{s}_0}, \epsilon_{\boldsymbol{s}_1}, ...\epsilon_{\boldsymbol{s}_{\text{-}1}})$ being the vector of corresponding ϵ's and $B_E : \mathfrak{s} \mapsto \mathfrak{b}_{\mathfrak{s}}$, with $\mathfrak{b}_{\mathfrak{s}} = (\mathbf{b}_{\boldsymbol{s}_0}, \mathbf{b}_{\boldsymbol{s}_1}, ...\mathbf{b}_{\boldsymbol{s}_{\text{-}1}})$ being the sequence of corresponding hyperballs. In the reverse direction, we let $\boldsymbol{s}(\mathbf{b}_i) = \boldsymbol{s}_i$ be the center-point of the hyperball $\mathbf{b}_i$, and $\epsilon(\mathbf{b}_i) = \epsilon_{\boldsymbol{s}_i}$ the radius of $\mathbf{b}_i$. Extending again to sequences, we let $\boldsymbol{\epsilon}_{\mathfrak{b}} = (\epsilon_{\boldsymbol{s}_0}, \epsilon_{\boldsymbol{s}_1}, ...\epsilon_{\boldsymbol{s}_{\text{-}1}})$ be the vector of corresponding hyperball radii, and $\mathfrak{s}_{\mathfrak{b}} = (\boldsymbol{s}_0, \boldsymbol{s}_1, ...\boldsymbol{s}_{\text{-}1})$ the plan of corresponding hyperball centers.

This allows us to re-represent the $\boldsymbol{\epsilon}$-tube around $\mathfrak{s}$, $\mathcal{N}_{\boldsymbol{\epsilon}(\mathfrak{s})}(\mathfrak{s})$ as:

$$[\![\mathfrak{b}_{\mathfrak{s}}]\!] = \bigtimes_{\mathbf{b}_i \in \mathfrak{b}_{\mathfrak{s}}} \mathbf{b}_i = \mathbf{b}_0 \times \mathbf{b}_1 \times ... \times \mathbf{b}_{\text{-}1} = \{(\boldsymbol{s}'_i)_{i=0}^{\ell(()\mathfrak{s})} : \boldsymbol{s}'_i \in \mathbf{b}_i\} = \mathcal{N}_{\boldsymbol{\epsilon}(\mathfrak{s})}(\mathfrak{s}) \tag{6.2}$$

Where we define $[\![\mathfrak{b}_{\mathfrak{s}}]\!]$ to be the Cartesian product of the hyperballs in $\mathfrak{b}_{\mathfrak{s}}$. If $\exists \mathfrak{s}^\star \in [\![\mathfrak{b}_{\mathfrak{s}}]\!]$ with $r_{\scriptscriptstyle K}(\mathfrak{s}^\star) = 1$, then the agent could randomly sample plans from $[\![\mathfrak{b}_{\mathfrak{s}}]\!]$ until $\mathfrak{s}^\star$ is found, but that would be haphazard, and how would the agent repeat $\mathfrak{s}^\star$? Instead, the agent can take a more systematic approach, taking some or all of its hyperballs and *refining* them, that is, adding *new* smaller hyperballs:

Definition 6.3 (Refinement). The refinement function `refine` takes in two arguments, $\mathbf{b}$ a hyperball and $a \in (0, 1)$ a scaling parameter, and returns a set of new "denser" hyperballs:

$$\texttt{refine}(\mathbf{b}, a) = \{\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, ...\mathbf{b}^{(k)}\}$$

with the following properties:

1. $\boldsymbol{s}(\mathbf{b}^{(1)}) = \boldsymbol{s}(\mathbf{b})$
2. $\forall \mathbf{b}' \in \texttt{refine}(\mathbf{b}, a), \epsilon(\mathbf{b}') = a \cdot \epsilon(\mathbf{b})$
3. $\mathbf{b} \subset \bigcup \texttt{refine}(\mathbf{b}, a)$

We can use the `refine` function to add several new "more precise" options in the area of an existing hyperball in $\mathfrak{b}_{\mathfrak{s}}$, so that the agent has more control over the behaviors it executes (Figure 6.2D). If the original tube was $\mathfrak{b}_{\mathfrak{s}} = (\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$, and the agent

refined $\mathbf{b}_2$, then the agent would have a new set of tubes $\mathfrak{B} = \{(\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2', \mathbf{b}_3) : \mathbf{b}_2' \in \texttt{refine}(\mathbf{b}_2, a)\}$, with the property that $\forall \mathfrak{s}' \in [\![\mathfrak{b}_\mathfrak{s}]\!], \exists \mathfrak{b}' \in \mathfrak{B}$ s.t. $\mathfrak{s}' \in [\![\mathfrak{b}']\!]$. Recalling the definition of $\boldsymbol{\epsilon}$-*reliability* (Equation 5.11), we can denote the reliability of a sequence of hyperballs (or a *path*) as $r_{S_K}(\mathfrak{b}) = r_{S_K}^{\boldsymbol{\epsilon}_\mathfrak{b}}(\mathfrak{s}_\mathfrak{b})$. Thus, when sampling plans from a path, a path corresponds to a behavior in the third sense.

If $\mathcal{B}$ is a set of hyperballs, and $\mathfrak{B}_\mathcal{B}^*$ the set of all paths of any length using hyperballs in $\mathcal{B}$, then ${}^+\mathfrak{B}_\mathcal{B}^* = \{\mathfrak{b} \in \mathfrak{B}_\mathcal{B}^* : r_{S_K}(\mathfrak{b}) = 1\}$ is the set of all robust paths in $\mathfrak{B}_\mathcal{B}^*$. If two paths $\mathfrak{b}_1, \mathfrak{b}_2 \in {}^+\mathfrak{B}_\mathcal{B}^*$ start at $\mathbf{b}$ and end at $\mathbf{b}^\star$, then $\mathfrak{b}_1 \sim \mathfrak{b}_2$, so we say they are both in the equivalence class $\xi_{\mathbf{b},\mathbf{b}^\star}^\mathcal{B}$. This is an equivalence class of *paths*. We let $\Xi_\mathcal{B}^\mathcal{B} = \{\xi_{\mathbf{b},\mathbf{b}^\star}^\mathcal{B} : \mathbf{b}, \mathbf{b}^\star \in \mathcal{B}\}$ be the set of all equivalence classes of paths. Then, $\Xi_{S_K}^\mathcal{B} = \{\xi_{\boldsymbol{s},\boldsymbol{s}^\star}^{S_K} \in \Xi_{S_K} : \exists \xi_{\mathbf{b},\mathbf{b}^\star}^\mathcal{B} \in \Xi_\mathcal{B}^\mathcal{B}$ s.t. $\boldsymbol{s} \in \mathbf{b}$ and $\boldsymbol{s}^\star \in \mathbf{b}^\star\}$ represents the total set of problems that can reliably be solved using $\mathcal{B}$. Suppose that there is a path $\mathfrak{b} \in \mathfrak{B}_\mathcal{B}^*$ for which $\mathfrak{s}_\mathfrak{b}$ has failed, with $\mathfrak{s}_\mathfrak{b}[0] = \boldsymbol{s}_0$ and $\mathfrak{s}_\mathfrak{b}[-1] = \boldsymbol{s}^\star$, but $\exists \mathfrak{s}^\star \in [\![\mathfrak{b}_\mathfrak{s}]\!]$ such that $r_{S_K}^{\boldsymbol{\epsilon}}(\mathfrak{s}^\star) = 1$. If $\mathcal{B}$ is repeatedly refined, producing an infinite sequence $\mathcal{B}_1$, $\mathcal{B}_2$, ... such that eventually every hyperball is refined, then it is guaranteed that $\exists n \in \mathbb{N}$ such that $\xi_{\boldsymbol{s}_0,\boldsymbol{s}^\star}^{S_K} \in \Xi_{S_K}^{\mathcal{B}_n}$, which is to say, if there *is* a robust plan in $[\![\mathfrak{b}]\!]$, our agent is sure to find it. However, if there *isn't*, our agent could spend forever trying various minute variations on $\mathfrak{s}_\mathfrak{b}$. If there is no solution inside of $[\![\mathfrak{b}]\!]$, then the agent will waste an infinite amount of time. Imagine someone bushwhacking through dense jungle coming across a steep cliff, occluded by foliage to either side. They could spend an infinite amount of time attempting different variations on climbing the cliff, without ever checking for a shallower slope past the foliage.

Alternatively, there are an infinite number of behaviors outside of $[\![\mathfrak{b}]\!]$, in $\mathfrak{S}_{S_K} - [\![\mathfrak{b}]\!]$. If there is no solution inside of $[\![\mathfrak{b}]\!]$, then some necessary waypoints for a solution lie outside of $[\![\mathfrak{b}]\!]$, and the agent has no hope of sampling those plans, that is, *unless* it can add hyperballs that cover new regions of state-space, that is, *extending* the scope of its behaviors:

Definition 6.4 (Extension). The extension function `extend` takes in three arguments, a hyperball $\mathbf{b}$ and two scaling parameters $a > 1$ and $c > 1$, and returns a new set of hyperballs covering "new" regions of S:

$$\texttt{extend}(\mathbf{b}, a, c) = \{\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, ...\mathbf{b}^{(k)}\}$$

with the following properties:

1. $\forall \mathbf{b}' \in \texttt{extend}(\mathbf{b}, a, c), \epsilon(\mathbf{b}') = a \cdot \epsilon(\mathbf{b})$
2. $B_{c \cdot \epsilon(\mathbf{b})}(\boldsymbol{s}(\mathbf{b})) \subset \mathbf{b} \cup (\bigcup \texttt{extend}(\mathbf{b}, a, c))$

We can use the `extend` function to add qualitatively new options for our agent: if a waypoint is needed for a solution and isn't currently inside of one of our agent's hyperballs, then repeated extension can expand the scope of the agent's behaviors (Figure 6.2E). This certainly doesn't guarantee that once the waypoint is covered by a hyperball, it can be incorporated into a robust behavior, and if our agent only ever extends, it will almost certainly miss important behaviors... but repeated refinement

can fix this (Figure 6.2F). In fact, these two operations give us the tools to deliver on our earlier promise of a countably infinite set of plans dense in $\mathfrak{S}^*_{S_\kappa}$.

Consider the set of hyperballs $\mathcal{B}$, with $\mathcal{S}(\mathcal{B})$ the set of center-points of those hyperballs. When we refine a hyperball $\mathbf{b} \in \mathcal{B}$, we essentially "update" or "mutate" $\mathcal{B}$, as $\mathcal{B}_{i+1} = \mathcal{B}_i \cup \texttt{refine}(\mathbf{b}, a)$. Similarly, when we extend a hyperball $\mathbf{b} \in \mathcal{B}$, we are also mutating $\mathcal{B}$, as $\mathcal{B}_{i+1} = \mathcal{B}_i \cup \texttt{extend}(\mathbf{b}, a, c)$. Repeated application of extension and refinement will produce an infinitely long but countable sequence of sets of hyperballs, $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, ...$, where $\forall i$, $|\mathcal{B}_i|$ is finite. We can express the infinite sequence of functions with a *polymutator function* $\mathcal{B}_i = \hat{\hat{m}}^i(\mathcal{B}_0)$. If $\hat{\hat{m}}$ is *fair*, then it never stops mutating by refinement and never stops mutating by extension, and it is *balanced* if every hyperball produced in the sequence is itself eventually selected for both extension and refinement. The set of waypoints of hyperballs at a stage of this sequence, $\mathcal{S}_i = \mathcal{S}(\hat{\hat{m}}^i(\mathcal{B}_0))$, can be used to construct a set of plans, $\mathfrak{S}^*_{\mathcal{S}_i}$. We denote the limit of this set as i goes to infinity as:

$$\mathfrak{S}^*_{\hat{\hat{m}}} = \bigcup_{i \in \mathbb{N}} \mathfrak{S}^*_{\mathcal{S}_i} \tag{6.5}$$

With $X = \bigcup_{i \in \mathbb{N}} \mathcal{S}_i$, the closure of $\mathfrak{S}^*_{\hat{\hat{m}}}$ is given by $\overline{\mathfrak{S}^*_{\hat{\hat{m}}}} = \mathfrak{S}^*_{\overline{X}}$. $\mathfrak{S}^*_{\hat{\hat{m}}}$ has an extremely important property: it is a *rational set of behaviors*:

Theorem 6.6 (The Polymutator Behavior Set is Rational). If $\hat{\hat{m}}$ is a fair and balanced polymutator function, and $\mathfrak{S}^*_{\hat{\hat{m}}} = \bigcup_{i=0}^{\infty} \mathfrak{S}^*_{\mathcal{S}(\hat{\hat{m}}^i(\mathcal{B}_0))}$, then:

1. $|\mathfrak{S}^*_{\hat{\hat{m}}}| = \aleph_0$ ($\mathfrak{S}^*_{\hat{\hat{m}}}$ is countably infinite) (By A.70)
2. $\overline{\mathfrak{S}^*_{\hat{\hat{m}}}} = \mathfrak{S}^*_S$ ($\mathfrak{S}^*_{\hat{\hat{m}}}$ is dense in $\mathfrak{S}^*_S$) (By A.76)

Making $\mathfrak{S}^*_{\hat{\hat{m}}}$ a set of *rational behaviors*.

The detailed proof of this theorem can be found in appendix A, here I will only give some broad intuition. Because the polymutator $\hat{\hat{m}}$ is *fair*, it never "locks-in" to a specific mutation, either `extend` or `refine`, meaning that for any set $\mathcal{B}_i = \hat{\hat{m}}^i(\mathcal{B}_0)$, there will be later sets that are extended, and later sets that are refined. Because $\hat{\hat{m}}$ is *balanced*, for any ball in any set $\mathcal{B}_i$, that ball will eventually be both extended and refined. This has two consequences: first, for any plan $\mathfrak{s}$, there will eventually be an index $k^\star_\blacktriangle(\mathfrak{s}) \in \mathbb{N}$ such that $\mathfrak{s}$ is *covered* by $\mathcal{B}_{k^\star_\blacktriangle(\mathfrak{s})}$, meaning that every subgoal of $\mathfrak{s}$ is contained in the union of hyperballs in $\mathcal{B}_{k^\star_\blacktriangle(\mathfrak{s})}$, written $\mathfrak{s} \Subset \mathcal{B}_{k^\star_\blacktriangle(\mathfrak{s})}$. Mind you, this doesn't guarantee that $\mathfrak{s}$ or something sufficiently similar can be reliably produced. For this we need the index $\psi(\mathcal{B}_0, \mathfrak{s}, \epsilon) = k^\star_\blacktriangledown(\mathcal{B}_{k^\star_\blacktriangle(\mathfrak{s})}, \epsilon) \in \mathbb{N}$, which yields the following property:

$$\forall \mathfrak{s} \in \mathfrak{S}^*_S, \forall \epsilon > 0 \; ; \; \exists \mathfrak{s}' \in \mathfrak{S}^*_{\mathcal{S}_{\psi(\mathcal{B}_0, \mathfrak{s}, \epsilon)}} : \mathfrak{s} \in \mathcal{N}_\epsilon(\mathfrak{s}') \tag{6.7}$$

This means that for any plan in the set of all possible plans, and even for any starting set of hyperballs (or waypoints with assigned generalization radii), we can choose an arbitrary degree of similarity, $\epsilon > 0$, and there will be a natural number, to which there is a corresponding finite set of waypoints. From this set, we can construct a plan that is within ϵ of our desired plan. In plainer language, our agent can perform an exhaustive

search over this set of behaviors, guaranteeing that any robust behavior that *needs* to be found *can* be found. First, unlike in most reinforcement learning, we are assuming that there is no external reset of the environment, meaning that just like in real life, our agent cannot "teleport". This provides a severe constraint on which behaviors can even be tried in the first place, that is, every plan has to start where the agent currently is. Additionally, the agent really only needs to actually enumerate $\Xi_{\mathcal{B}_i}^{\mathcal{B}_i}$, so there are some tricks we can use to dramatically reduce the scope of the search. Basically, if a sub-path failed, it might be a good idea to defer trying *other* paths that use a similar subpath, until other options have been exhausted (Figure 6.2G). The question is, how do we actually organize such a search in practical terms?

Chapter 7

Segraphs

Overview

In order to organize the enumeration of plans and addition of new waypoints, we use a graph whose vertices are hyperballs that segment state-space, a *segraph*. The segraph's edges track the empirical reliability of the agent's low-level controller for traversing between hyperballs. By recording successful and failed traversals over edges, the agent is able to both avoid unreliable edges and select novel paths to drive plan enumeration while simultaneously regulating segraph mutation. This creates a tight feedback loop: path enumeration supplies experiences that guide graph mutation, and each mutation expands the future path-set, allowing the agent to adaptively search the space of achievement classes without repeating failed sub-behaviors. Because any mutated segraph contains its parent, disconnected subgraphs that once blocked enumeration are eventually contained by better-connected segraphs, ensuring that local exploration results in global problem-solving.

Notation

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A **segraph**, a graph with directed edges $\mathcal{E}$ that connect vertices $\mathcal{V}$ that segment state-space, which is used to organize the enumeration of plans
$\mathbf{v}_i$	A segraph **vertex** $\mathbf{v}_i$ representing a "chunk" of state-space, a corresponding hyperball $\mathbf{b}_i \subset S$
$\mathbf{e}_{i,j}$	The directed segraph **edge** between vertices $\mathbf{v}_i$ and $\mathbf{v}_j$, representing a chunk of connection-space $\mathbf{c}_{i,j} = \mathbf{b}_i \times \mathbf{b}_j \subset S^2$
M	The **measure** of a vertex or edge, $M(\mathbf{v}_i) = \lvert\mathbf{b}_i\rvert$, $M(\mathbf{e}_{i,j}) = \lvert\mathbf{c}_{i,j}\rvert$
$r_{\kappa}(\mathbf{e}_{i,j})$	The theoretical **reliability** of the edge $\mathbf{e}_{i,j}$, equal to $r_{\kappa}((\mathbf{b}_i, \mathbf{b}_j))$
$n_s(\mathbf{e})$	The number of **successful traversals** by $\mathcal{A}$ over $\mathbf{e}$
$n_f(\mathbf{e})$	The number of **failed traversals** by $\mathcal{A}$ over $\mathbf{e}$
$n(\mathbf{e})$	The total number of **traversals** by $\mathcal{A}$ over $\mathbf{e}$, equal to $n_s(\mathbf{e}) + n_f(\mathbf{e})$
$\tilde{r}_{\kappa}(\mathbf{e})$	The empirically **estimated reliability** of $\mathbf{e}$
$\mathfrak{p}$	A **path**, a sequence of unique vertices

$\mathfrak{P}$	The **set of all paths**. This symbol admits many variations. The set of paths of length n is $\mathfrak{P}^n$, of any length, $\mathfrak{P}^*$. The set of paths with vertices in the set A is $\mathfrak{P}_A$. The set of robust paths is ${}^+\mathfrak{P}$. The set of paths that start at $\mathbf{v}$ and end at $\mathbf{v}'$ is $\mathfrak{P}(\mathbf{v}, \mathbf{v}')$
$\mathcal{G}$	The probability distribution over goals used to represent the way an agent selects problems to try to solve
$\mathcal{P}$	The probability distribution over paths used to represent the way an agent selects plans to try to solve a problem
$r_{\kappa}(\mathfrak{p})$	The reliability of a path, equal to the reliability of the corresponding hyperball sequence. When $r_{\kappa}(\mathfrak{p}) = 1$, the path is **robust**
$\xi^{\mathcal{G}}_{\mathbf{v},\mathbf{v}'}$	The equivalence class of robust paths over $\mathcal{G}$ between $\mathbf{v}$ and $\mathbf{v}'$
$\Xi^{\mathcal{G}}_{\mathcal{V}}$	The set of all equivalence classes of robust paths over $\mathcal{G}$
$\texttt{try}(\mathfrak{p})$	A function modeling the action of an agent **trying to follow a path**
$\mathcal{E}^{\times}$	The set of edges that fail when an agent tries to follow a path. If $\mathcal{E}^{\times} = \varnothing$, then the path succeeds
$\mathcal{E}^{\downarrow}$	The set of edges that are inhibited (excluded from pathfinding by the agent) due to having failed while trying paths
$\mathcal{E}^{+}$	The set of edges that are not inhibited and can be used in pathfinding

If a transition between two hyperballs $\mathbf{b}_1$ and $\mathbf{b}_2$ along a path has failed, then to deprioritize paths that use that transition, our agent needs some way to record the fact of the failure. Indeed, if the transition succeeded, this would be evidence that other paths using that transition might also succeed, and our agent would want to record the fact of the success as well. This information can easily be represented by a *graph*, where each *vertex* is identified with a hyperball, and each directed *edge* from one vertex to another encodes information about the reliability of transitioning between the corresponding hyperballs. As the vertices of this graph effectively segment state-space, we call it a *segraph*:

Definition 7.1 (Segraph). A *segraph* is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with:

- $\mathcal{V}$ - a finite set of vertices; each vertex $\mathbf{v}_i \in \mathcal{V}$ is a tuple $(i, \mathbf{b}_i)$ where $i \in \mathbb{N}$ is an identifier and $\mathbf{b}_i$ is the corresponding hyperball, called the vertex's *field* in state-space. The measure (size) of this field is denoted $M_i = M(\mathbf{v}_i) = |\mathbf{b}_i|$.
- $\mathcal{E}$ - a finite set of directed edges; each edge $\mathbf{e} \in \mathcal{E}$ is an ordered tuple $(\mathbf{v}_i, \mathbf{v}_j)$ representing a connection from $\mathbf{v}_i$ to $\mathbf{v}_j$. We say that $\mathbf{c}_{i,j} = \mathbf{b}_i \times \mathbf{b}_j \subset S^2$ is the edge's *field* in connection-space. The measure (size) of this field is denoted $M_{i,j} = M(\mathbf{e}_{i,j}) = \left|\mathbf{c}_{i,j}\right|$.

We let $S(\mathcal{G}) = \bigcup_{\mathbf{v}_i \in \mathcal{V}} \mathbf{b}_i$ be the *coverage* of $\mathcal{G}$.

The definitions of `refine` and `extend` can be seamlessly extended to segraph vertices, with the only additional machinery being that for pragmatic reasons, the segraph

enforces a redundancy constraint on added vertices, which is to say, if there is an existing vertex (hyperball) in the segraph that is very similar to a new vertex that *would* be added by refinement or extension, then that new vertex is actually unnecessary and is not added. Additionally, we do not assume that $\mathcal{G}$ is fully-connected, so to ensure that every possible plan can actually be generated, we also use a linking function `link` that creates an edge between two vertices.

The segraph has two deeply intertwined jobs: the first is to help the agent navigate to distant goals by providing robust *paths* (sequences of vertices), the second is to regulate its own mutation to produce a balanced sequence of segraphs, so that eventually the agent can generate any behavior necessary to solve any problem. We accomplish this by establishing an extremely tight bidirectional feedback loop between how the segraph is *used* by the agent and how the segraph gets *mutated*. Over time, the segraph self-organizes in response to the needs of the agent, in much the same way that bone, muscle, or nervous tissue respond to usage by an organism.

The feedback loop between *usage* of the segraph and *mutation* of the segraph is absolutely critical: as *using* the segraph just means following a path sampled from it (selecting a goal and finding a path to it), repeated *use* of a given segraph is (or should be) equivalent to *enumerating paths* over the segraph. Likewise, *mutating* the graph just means creating a modified segraph, repeated *mutation* creates a sequence of segraphs; in other words, the enumeration of segraphs. The enumeration of segraphs shapes the set of paths that can be enumerated at each stage, and the enumeration of paths collects information that guides or biases the enumeration of segraphs.

For either the usage or mutation of the segraph to be *adaptive*, our agent has to be able to remember information about its experiences while using the segraph. Ultimately, both mutation and usage will hinge on the agent having an estimate of the reliability of each edge of the segraph, which it accomplishes by storing the number of failed and successful transitions over each edge $\mathbf{e}_{i,j}$ to empirically estimate the true reliability $r_{\scriptscriptstyle K}(\mathbf{e}_{i,j}) = r_{\scriptscriptstyle K}((\mathbf{b}_i, \mathbf{b}_j))$ as:

$$\tilde{r}_{\scriptscriptstyle K}(\mathbf{e}_{i,j}) = \frac{n_s(\mathbf{e}_{i,j})+\tilde{n}_s}{n_s(\mathbf{e}_{i,j})+\tilde{n}_s+n_f(\mathbf{e}_{i,j})+\tilde{n}_f} = \frac{n_s(\mathbf{e}_{i,j})+\tilde{n}_s}{n(\mathbf{e}_{i,j})+\tilde{n}_s+\tilde{n}_f} \tag{7.2}$$

where $n_s(\mathbf{e}_{i,j})$ and $n_f(\mathbf{e}_{i,j})$ are simply the empirical counts of successful and failed traversals by K over the edge $\mathbf{e}_{i,j}$, with $\tilde{n}_s$ and $\tilde{n}_f$ pseudo-counts encoding the agent's initial guess about $\mathbf{e}_{i,j}$'s reliability, and $n(\mathbf{e}) = n_s(\mathbf{e}) + n_f(\mathbf{e})$. These traversal counts will ultimately control how the segraph gets mutated (segraph enumeration), as well as how goals and paths are selected (path enumeration).

7.1 Enumerating Paths

As each plan must start where the agent currently is, the main degrees of freedom available to our agent while using the segraph are which *targets* it tries to reach and which *paths* it chooses to take to those targets. We model the agent's *selection of goals* with the distribution $\mathcal{G}$ over $\mathcal{V}$. A path $\mathfrak{p} = (\mathbf{v}_0, \mathbf{v}_1, ...\mathbf{v}^\star)$ from $\mathbf{v}_0$ to $\mathbf{v}^\star$ is just a sequence of vertices, with no vertex repeated. As the agent is *at* a specific state, it has to choose a starting vertex whose hyperball contains its state. Let $\mathfrak{P}^*_{\mathcal{G}} = \mathfrak{P}^*_{(\mathcal{V},\mathcal{E})}$

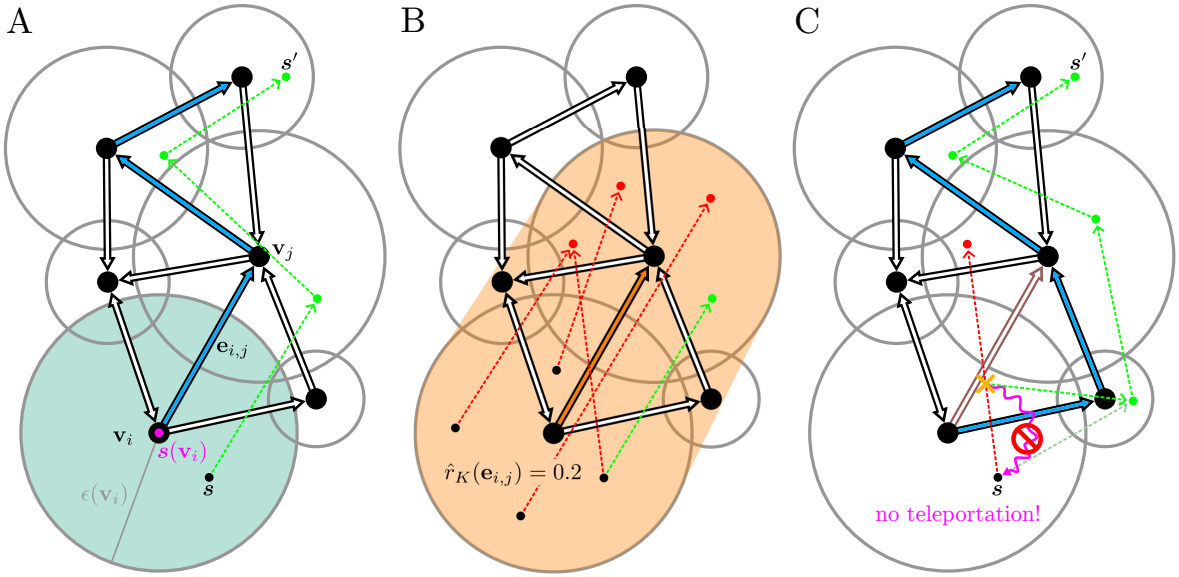


Figure 7.1: (A) A segraph is a directed graph whose vertices segment state-space. Each vertex's segment of state-space, or field, is a hyperball centered at $\boldsymbol{s}(\mathbf{v}_i)$ with radius $\boldsymbol{\epsilon}(\mathbf{v}_i)$. An edge represents a connection between two regions of state-space. In order to help the agent reach a distant goal such as $\boldsymbol{s}'$ the segraph will find a path (sequence of vertices highlighted in blue) between the vertex the agent is currently in and a vertex containing the goal. Then, the agent will sample waypoints from the fields of the vertices along the path, generating a plan from its current state to the target state (green dots connected by dashed line). (B) Experience trying to cross the edges of the segraph allows the segraph to estimate the reliability of each of its edges. (C) Unreliable edges (such as the one marked in orange), can be inhibited, prevented from use in pathfinding, which can allow for potentially better plans to be tried (example alternative path highlighted in orange). The new plan that is generated has to start where the last (failed) plan left the agent, as there are no external resets.

be the set of all paths over the segraph $\mathcal{G}$. We model the agent's *selection of a path* with the distribution $\mathcal{P}^{\mathbf{v}\to\mathbf{v}^\star}_{(\mathcal{V},\mathcal{E})}$ over $\mathfrak{P}^*_{(\mathcal{V},\mathcal{E})}$. In a way, $\mathfrak{P}^*_{\mathcal{G}}$ acts like a discretized version of $\mathfrak{S}^*_{S(\mathcal{G})} \subset \mathfrak{S}^*_S$, which in the infinite limit it approaches. As a finite set, we can choose any order for enumerating it, but efficient problem solving requires an *intelligent* order.

The first thing to realize is that, just as our agent needs to exhaust Ξ_{S_K} rather than ${}^+\mathfrak{S}^*_{S_K}$, for every $\mathbf{v}, \mathbf{v}' \in \mathcal{V}$ it is sufficient for our agent to find a *single* path $\mathfrak{p}$ between them with $r_K(\mathfrak{p}) = 1$. Let $\mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{v}') = \{\mathfrak{p} \in \mathfrak{P}^*_{\mathcal{G}} : \mathfrak{p}[0] = \mathbf{v}, \mathfrak{p}[\text{-}1] = \mathbf{v}'\}$ be the set of paths connecting $\mathbf{v}$ to $\mathbf{v}'$, and ${}^+\mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{v}') = \{\mathfrak{p} \in \mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{v}') : r_K(\mathfrak{p}) = 1\}$ the set of robust paths from $\mathbf{v}$ to $\mathbf{v}'$ over $\mathcal{G}$. You may recall the construction of plan-equivalence from chapter 5: here, two *paths* $\mathfrak{p}_i, \mathfrak{p}_j$ are equivalent if $\mathfrak{p}_i, \mathfrak{p}_j \in {}^+\mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{v}')$, we denote the equivalence class as $\xi^{\mathcal{G}}_{\mathbf{v},\mathbf{v}'}$, and the set of all such classes as $\Xi^{\mathcal{G}}_{\mathcal{V}} = \{\xi^{\mathcal{G}}_{\mathbf{v},\mathbf{v}'} : \mathbf{v}, \mathbf{v}' \in \mathcal{V}\}$. It is this set that must actually be exhaustively searched, and thus *enumerated*. Naturally, our agent doesn't know a priori which paths are robust (if any are), and so must perform a (non-exhaustive) search over $\mathfrak{P}^*_{\mathcal{G}}$. Ultimately, this enumeration will be performed via the *inhibition* of different sets of edges.

7.2 Try and Try Again: Failure-Driven Enumeration

One of the basic insights of this paper is that, after our agent has tried a behavior and failed, the most basic way that our agent can adapt is to simply not repeat the failed behavior (or failed sub-behavior). As the novelist Rita Mae Brown said (misattributed to Albert Einstein), "Insanity is doing the same thing over and over and expecting different results." In this sense, the necessarily gradual weight updates of deep networks make them nearly insane. By never repeating failed behavior, our agent is in some sense guaranteed to eventually find a successful behavior. We accomplish this by simply *inhibiting* any edge that the agent fails to traverse. Imagine that our agent started at the vertex $\mathbf{v}$, tried to reach the goal $\mathbf{v}^\star$, and tried to follow the path $\mathfrak{p}$. We can express this with the function:

$$\texttt{try}(\mathfrak{p}) \mapsto \mathbf{v}', \mathcal{E}^\times \tag{7.3}$$

where $\mathbf{v}'$ is the final state of the agent after trying to follow $\mathfrak{p}$, and $\mathcal{E}^\times$ is the set of edges (if any) that failed while following the path. If the agent succeeded, then $\mathbf{v}' = \mathbf{v}^\star$ and $\mathcal{E}^\times = \varnothing$, otherwise, $\mathbf{v}' \neq \mathbf{v}^\star$ and $\mathcal{E}^\times \neq \varnothing$. Recall that the goal at this stage for our agent is to enumerate $\Xi^{\mathcal{G}}_{\mathcal{V}}$, *not* $\mathfrak{P}^*_{\mathcal{G}}$. If $\mathfrak{p}$ failed, then we know that $r_K(\mathfrak{p}) < 1$, meaning that $\mathfrak{p}$ cannot be a representative of $\xi^{\mathcal{G}}_{\mathbf{v},\mathbf{v}^\star}$. Indeed, *no* path $\mathfrak{p}'$ containing an edge in $\mathcal{E}^\times$ can be a member of *any* class in $\Xi^{\mathcal{G}}_{\mathcal{V}}$, so all such paths can be *safely excluded* from our agent's enumeration.

To do this, we simply preclude the use of failed edges in future paths. If $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we accomplish this by splitting $\mathcal{E}$ into two disjoint sets, $\mathcal{E}^\downarrow$ and $\mathcal{E}^+$, where $\mathcal{E}^\downarrow$ are *inhibited*, and $\mathcal{E}^+$ are *disinhibited*. Initially, we let $\mathcal{E}^+ = \mathcal{E}$ and $\mathcal{E}^\downarrow = \varnothing$, then after every call of `try`, we perform the updates: $\mathcal{E}^\downarrow \leftarrow \mathcal{E}^\downarrow \cup \mathcal{E}^\times$, $\mathcal{E}^+ \leftarrow \mathcal{E}^+ - \mathcal{E}^\times$. Basically, the agent *inhibits* every edge that failed. Instead of sampling paths from $\mathcal{P}^{\mathbf{v}\to\mathbf{v}^\star}_{(\mathcal{V},\mathcal{E})}$, our agent samples only from $\mathcal{P}^{\mathbf{v}\to\mathbf{v}^\star}_{(\mathcal{V},\mathcal{E}^+)}$, the set of paths over $\mathcal{G}$ with only disinhibited edges. Because of this, the agent is inherently *resilient* to failure, as it merely tries again to

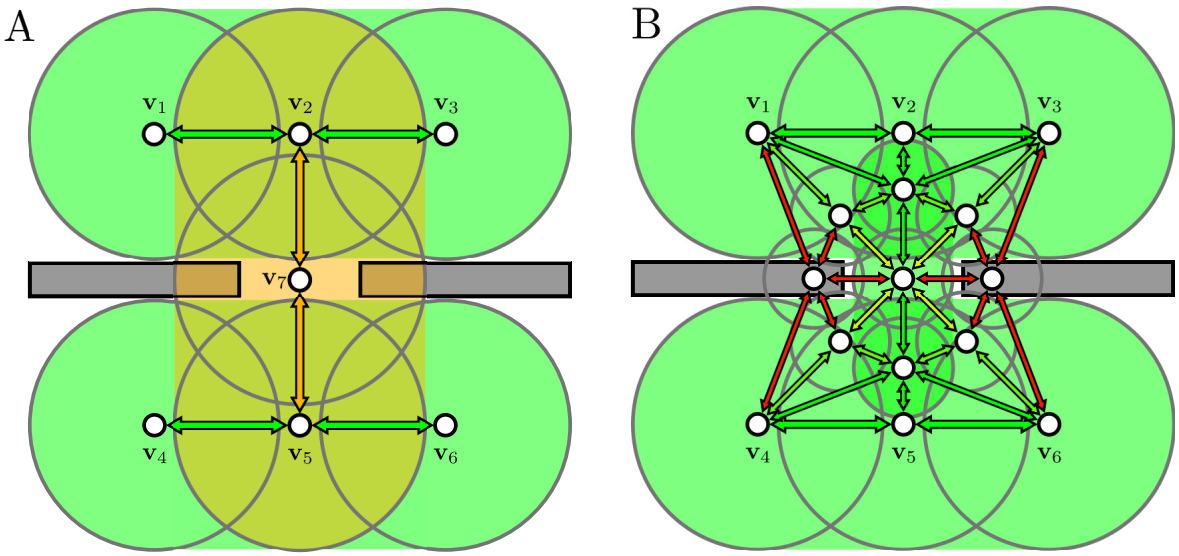


Figure 7.2: (A) For the shown segraph $\mathcal{G}$, the elements of $\Xi^{\mathcal{G}}_{\mathcal{V}}$ are not connected to each other, meaning there is no guarantee that our agent can actually enumerate $\Xi^{\mathcal{G}}_{\mathcal{V}}$, complicating the agent's task of trying every behavior. (B) However, through systematic refinement, it can be possible to construct a new segraph $\mathcal{G}'$ which connects the elements of $\mathcal{G}$, allowing the agent to enumerate $\Xi^{\mathcal{G}}_{\mathcal{V}}$, just using a different segraph.

reach the goal, now from a new location, and with some options trimmed away.

7.3 The Curse of Locality

There is a problem with the failure-driven enumeration of paths, caused by inhibition. Consider the segraph $\mathcal{G}$ shown in Figure 7.2A. All the edges among $\mathcal{V}' = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ are robust, as are all the edges among $\mathcal{V}'' = \{\mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_6\}$, whereas the edges among $\{\mathbf{v}_2, \mathbf{v}_7, \mathbf{v}_5\}$ are not robust. Due to the structure of the segraph, the capability set of the segraph can be split into two halves, $\Xi^{\mathcal{G}}_{\mathcal{V}} = \{\xi^{\mathcal{G}}_{\mathbf{v}_i,\mathbf{v}_j} : \mathbf{v}_i, \mathbf{v}_j \in \mathcal{V}'\} \cup \{\xi^{\mathcal{G}}_{\mathbf{v}_i,\mathbf{v}_j} : \mathbf{v}_i, \mathbf{v}_j \in \mathcal{V}''\}$. There is no achievement class connecting a vertex in $\mathcal{V}'$ to $\mathcal{V}''$, or vice versa! That means that if the agent tries to follow a path from, say, $\mathbf{v}_1$ to $\mathbf{v}_4$, it will likely fail at edge $\mathbf{e}_{2,7}$ or $\mathbf{e}_{7,5}$, which will cause that edge to be inhibited, thus *disconnecting* $\mathcal{G}$, making it impossible to enumerate $\Xi^{\mathcal{G}}_{\mathcal{V}}$.

This is not as catastrophic as it first appears. Recall that a path over a segraph is a sequence of hyperballs, which itself corresponds to a specific $\boldsymbol{\epsilon}$-tube, a bundle of *plans*. For a given graph, let $\mathfrak{S}^*_{\mathcal{G}} = \{\mathfrak{s} \in \mathfrak{p} : \mathfrak{p} \in \mathfrak{P}^*_{\mathcal{G}}\}$ be the set of all plans that can be sampled from any path over $\mathcal{G}$. We say that one segraph $\mathcal{G}'$ *contains* another segraph, $\mathcal{G}$, if and only if $\mathfrak{S}^*_{\mathcal{G}} \subset \mathfrak{S}^*_{\mathcal{G}'}$, which we write as $\mathcal{G}' \succcurlyeq \mathcal{G}$. What is interesting is that, for any mutation operator $\texttt{mutation} \in \{\texttt{extend}, \texttt{refine}, \texttt{link}\}$, $\texttt{mutation}(\mathcal{G}) \succcurlyeq \mathcal{G}$. A natural consequence of this is that, if $\mathcal{G}'$ is a mutation of $\mathcal{G}$, that $\Xi^{\mathcal{G}}_{S_\kappa} \subset \Xi^{\mathcal{G}'}_{S_\kappa}$. As these relationships are transitive, for any sequence of mutations leading from $\mathcal{G}_i$ to $\mathcal{G}_{i+k}$, it will be the case that $\Xi^{\mathcal{G}_i}_{S_\kappa} \subset \Xi^{\mathcal{G}_{i+k}}_{S_\kappa}$. The key takeaway here is that, even if our agent cannot enumerate $\Xi^{\mathcal{G}_i}_{\mathcal{V}}$ because it is disconnected, it could be the case at a later time that it can still find every element of $\Xi^{\mathcal{G}_i}_{\mathcal{V}}$, just as a subset of an appropriately mutated segraph with achievement set $\Xi^{\mathcal{G}_{i+k}}_{S_\kappa}$ (Figure 7.2B).

What this speaks to is the necessity that mutation occur *where the agent needs it*, because in some sense, to reach the broader graph, the agent must first be able to negotiate the "local" graph immediately around it, necessitating a sort of nested enumeration of sub-graphs, as-needed. Because mutation will ultimately happen, when it happens, wherever the agent currently is, it must be the case that in order to achieve a balanced sequence of segraphs (produced through mutation), it will be necessary to carefully control *where* the agent is *on* the segraph. In other words, while mutation of the graph will ultimately control which paths are available for the agent to explore, the primary way that we can *control* mutation is by *controlling* which paths are followed... the enumeration of plans loops back on itself.

Chapter 8

Segraph Self-Organization

Overview

A self-organizing segraph regulates its own development by coupling goal-oriented usage and stress-induced mutation together. A per-edge stress score signals where refinement (splitting a hyperball to raise statespace resolution) can increase the segraph's utility, while a dynamic affordance threshold forces pathfinding to bias the agent toward traversing edges that are either highly reliable or likely to benefit from refinement. Traversal failures actively inhibit edges and raise the threshold; when no paths are left, inhibition is reset and the threshold is lowered, completing a homeostatic loop that re-opens blocked paths as-needed. Every successfully visited vertex is extended before it can be refined, and overlapping or co-traversed vertices are linked to increase connectivity. Exploration favors edges with high epistemic value (large, under-sampled regions), and paths are tried by simplicity (shortest first), so enumeration advances systematically as failures prune the search space. These coupled mechanisms ensure that over an unbounded sequence of segraphs, coverage becomes dense and the agent ultimately acquires a robust plan for every solvable problem.

Notation

$\alpha(\mathfrak{p})$	The total affordance provided by a path $\mathfrak{p}$
$q(\mathbf{e})$	The utility of an edge $\mathbf{e}$ in terms of the incremental affordance it provides
$q_{\text{ref}}(\mathbf{e})$	The total utility of the edges produced by refining $\mathbf{e}$
$\Delta q(\mathbf{e})$	The theoretical change in utility resulting from refining $\mathbf{e}$
$\Delta\tilde{q}(\mathbf{e})$	An empirical estimate of $\Delta q(\mathbf{e})$
α'	An affordance-threshold used to regulate the order in which paths are tried
$\alpha(\mathbf{e})$	The affordance of the edge $\mathbf{e}$, $\alpha(\mathbf{e}) = r_{\kappa}(\mathbf{e}) \cdot M(\mathbf{e})$
$\tilde{\alpha}(\mathbf{e})$	The estimated affordance of the edge $\mathbf{e}$, $\tilde{\alpha}(\mathbf{e}) = \tilde{r}_{\kappa}(\mathbf{e}) \cdot M(\mathbf{e})$
$\mathcal{E}^{<}(\alpha')$	The set of edges that are below the affordance threshold α'
$\mathcal{E}^{\geq}(\alpha')$	The set of edges that are above the affordance threshold α'
$\mathfrak{P}_{\mathcal{G}}(\mathbf{v}, \mathbf{e})$	The set of all paths over $\mathcal{G}$ that start at $\mathbf{v}$ and end by crossing $\mathbf{e}$
$\mathcal{P}_{\mathcal{G}}^{\mathbf{v}\to\mathbf{e}}$	The probability distribution of paths from $\mathbf{v}$ to $\mathbf{e}$ over $\mathcal{G}$

$\Upsilon(\mathbf{e})$	The epistemic value of an edge $\mathbf{e}$, used to weigh the probability that $\mathbf{e}$ is chosen as a goal, modeled by the distribution $\mathcal{G}_\Upsilon$ over edges

In the following section, I lay out *one* possible way of arranging the self-organized usage and mutation of a segraph: there are surely other ways, and our particular method should only be considered as illustrative, rather than definitive. The development of these self-organizing mechanisms was guided primarily by intuition and experimentation, so a broader theory of different types of ordering is required in the future. We begin by laying out the conditions under which segraph refinement is adaptive, rather than detrimental or redundant, and end by describing some practical ways that paths can be sensibly ordered over a given segraph. To understand the way that usage and mutation interact to produce an infinite sequence of segraphs which, in the limit, allow the agent to generate a behavior to solve any problem, we have to reframe the issue: what is the agent trying to *maximize*? Because we want our agent to be able to eventually solve any problem (be able to go to any point from any point), we are arguably trying to maximize $|\Xi^{\mathcal{A}}_{S_K}|$, the measure of the set of points in S_K the agent can go to and from.

Technically, the ability of the agent to go from one point $\boldsymbol{s}$ to another $\boldsymbol{s}'$ using its controller K corresponds to an *affordance* [141] of the statespace. In this sense, $\text{con}_K(\boldsymbol{s})$ represents the total affordance of the agent while in the state $\boldsymbol{s}$, but as previously discussed, $|\text{con}_K(\boldsymbol{s})| \ll |S_K|$, meaning that most points in S_K cannot be reached by K from any given point. This, of course, motivated the introduction of plans, sequences of waypoints that, in effect, extend the capability of the agent by allowing it to reach more of statespace via *intermediary* steps. While we don't treat noise in this paper, successful plans that aren't robust are practically unrealizable, so we restrict our attention to plans that have a non-zero-measure "tube" of successful plans around them. This gives our agent some "wiggle room". Organizing information about the set of such plans given a finite set of waypoints motivated the introduction of segraphs.

A segraph, along with a probability distribution over paths, provides a means of selecting a path to a goal. In general, a path $\mathfrak{p}$ can be treated as a *bundle* of plans, with $[\![\mathfrak{p}]\!] = \bigtimes_{\mathbf{v}\in\mathfrak{p}} \mathbf{b}(\mathbf{v})$ representing this bundle (an $\boldsymbol{\epsilon}$-tube). The path is not a valuable object in its own right: it only matters as a means of getting from points inside of $\mathfrak{p}[0]$ to points inside of $\mathfrak{p}[-1]$. If $r_K(\mathfrak{p})$ is the reliability of $\mathfrak{p}$, then we can express the affordance of $\mathfrak{p}$ as $\alpha(\mathfrak{p}) = r_K(\mathfrak{p}) \cdot M(\mathfrak{p}[0]) \cdot M(\mathfrak{p}[-1])$. From the perspective of the agent, a path is only as useful as it is accessible: a path that is never sampled has only virtual affordance. So, the affordance of the agent is $\alpha(\mathcal{A}) = \sum_{\mathfrak{p}\in\mathfrak{P}_\mathcal{G}} \mathcal{P}(\mathfrak{p})\alpha(\mathfrak{p})$. How is $\alpha(\mathcal{A})$ effected by mutation of $\mathcal{G}$?

8.1 Refinement

While "refinement" is an operation that occurs to an individual *vertex*, the relevant information about when to refine a vertex actually exists on an *edge*, as we are ultimately interested in increasing the resolution of the corresponding $\boldsymbol{\epsilon}$-tubes. Henceforth, when we refer to "refinement", we will be talking about an operation that occurs *on an edge*, with the decision to "refine an edge" resulting in the refinement of *one of its*

vertices. Ultimately, we are interested in the effect that refinement has on our agent's affordance, $\alpha(\mathcal{A})$. While calculating the contribution of an individual edge $\mathbf{e}$ on the total $\alpha(\mathcal{A})$ is probably intractable, it is not unreasonable to assume that it is related to the total set of paths that pass through the edge. We can model the global utility of an edge as $q(\mathbf{e}) = \sum_{\mathfrak{p} \in \mathfrak{P}_{\mathcal{G}}(\mathbf{e})} \mathcal{P}(\mathfrak{p})\alpha(\mathfrak{p})$, where $\mathfrak{P}_{\mathcal{G}}(\mathbf{e})$ is the set of all paths passing over the edge $\mathbf{e}$.

Assuming that refining the edge $\mathbf{e}$ will split it into k smaller edges $\{\mathbf{e}'_1, ...\mathbf{e}'_k\}$ with disjoint fields[1], the total "utility" of these refined edges would be $q_{\text{ref}}(\mathbf{e}) = \sum_{i=1}^{k} q(\mathbf{e}'_i)$. Then, the change in overall affordance from refining the edge $\mathbf{e}$ would be $\Delta q(\mathbf{e}) = q_{\text{ref}}(\mathbf{e}) - q(\mathbf{e})$. To get a tractable closed-form expression, we have to make several assumptions, which we elaborate on in section D.3. However, using these simplifications, we get that:

$$\Delta q(\mathbf{e}) = c \cdot N_{\mathfrak{p}}(\mathbf{e}) r_{K}(\mathbf{e})(1 - r_{K}(\mathbf{e})) \tag{8.1}$$

Where $N_{\mathfrak{p}}(\mathbf{e})$ is the weighted number of paths that go through the edge. This score, which we call the edge's *stress*, is an indicator of how useful it would be to refine an edge. An edge that has never been used ($N_{\mathfrak{p}}(\mathbf{e}) = 0$) is unlikely to benefit from refinement. If an edge is completely reliable ($(1 - r_{K}(\mathbf{e})) = 0$), then the edge is fine and doesn't need to be refined. If the edge is completely unreliable ($r_{K}(\mathbf{e}) = 0$), then all of it's child-edges will *also* be completely unreliable, so nothing will have been gained. If some used paths actually pass through $\mathbf{e}$, and it is of intermediate reliability, then $\Delta q(\mathbf{e}) > 0$, because refining the edge creates the possibility that some child-edges will be *more* reliable than the parent, while others less: these lesser children can be filtered out (inhibited), allowing our agent to separate the wheat from the chaff, so to speak.

While ideally our agent would just refine edges with maximum $\Delta q(e)$, in fact, our agent only has an estimate of this score, $\Delta\tilde{q}(\mathbf{e}) = c \cdot n(\mathbf{e})\tilde{r}_{K}'(\mathbf{e})(1 - \tilde{r}_{K}'(\mathbf{e}))$, where $n(\mathbf{e})$ empirically estimates how important $\mathbf{e}$ is, and $\tilde{r}_{K}'(\mathbf{e})$ is the empirical estimate of the edge's reliability, without pseudo-counts.

8.2 Ordering Paths by Affordance to Regulate Refinement

In order to bias our agent toward refining edges with a high Δq, we need to somehow encourage our agent to traverse edges through which high-affordance paths pass. Directly choosing paths based on their affordance is intractable, but simply *enforcing* that all edges used for pathfinding are themselves high-affordance guarantees that the resulting paths are also high-affordance. The affordance of an edge is just $\alpha(\mathbf{e}) = r_{K}(\mathbf{e}) \cdot M(\mathbf{e})$, its estimated value being $\tilde{\alpha}(\mathbf{e}) = \tilde{r}_{K}(\mathbf{e}) \cdot M(\mathbf{e})$. Thresholding *edges* by their affordance rather than *paths* is an approximation, and perhaps a hack, but in practice it seems to work. By prioritizing high-affordance paths before lower-affordance paths, we increase the odds that our agent will encounter edges with high Δq (even when $\Delta\tilde{q}$ is a poor estimate), thus helping our agent to maximize $\alpha(\mathcal{A})$ in the long-run.

We accomplish this by the use of a dynamic affordance-threshold, α', which separates all edges into two sets: $\mathcal{E}^{<}(\alpha') = \{\mathbf{e} \in \mathcal{E} : \tilde{\alpha}(\mathbf{e}) < \alpha'\}$ and $\mathcal{E}^{\geq}(\alpha') = \{\mathbf{e} \in$

[1] This is a simplification, in reality they will overlap.

$\mathcal{E} : \tilde{\alpha}(\mathbf{e}) \geq \alpha'\}$, then restricting pathfinding to use only edges in $\mathcal{E}^{\geq}(\alpha')$. Recall from section 7.2 that edges that have failed are put in a set $\mathcal{E}^{\downarrow}$, with $\mathcal{E}^{+} = \mathcal{E} - \mathcal{E}^{\downarrow}$. We extend this logic, constructing the set of edges inhibited *for any reason* as $\mathcal{E}^{-} = \mathcal{E}^{<}(\alpha') \cup \mathcal{E}^{\downarrow}$, with $\mathcal{E}^{+} = \mathcal{E} - \mathcal{E}^{-} = \mathcal{E}^{\geq}(\alpha') - \mathcal{E}^{\downarrow}$. By starting α' high and lowering it if a path cannot be found, our agent $\mathcal{A}$ can prioritize higher-affordance paths.

Notably, prioritizing high-affordance paths does *not* necessarily prioritize highly *reliable* paths, so in a sense this is slightly unproductive. However, its primary purpose is not to induce a sensible order on *paths*, but instead to induce a sensible order on *refinements*. Recall that mutation (including refinement) will happen, when it happens, where the agent is, and so controlling where the agent is ultimately controls where refinement happens. Essentially this order exposes edges that are large and reliable (good for using the segraph), *or* are very large and somewhat reliable, which would benefit from refinement (good for mutating the segraph).

For now though, we have to grapple with how to adapt the threshold, α'. By setting α' high, our agent is guaranteed to sample only paths with high (estimated) potential from $\mathfrak{P}_{(\mathcal{V},\mathcal{E}^{+})}(\mathbf{v}, \mathbf{v}')$... However, if α' is *too* high, then $\mathfrak{P}_{(\mathcal{V},\mathcal{E}^{+})}(\mathbf{v}, \mathbf{v}') = \emptyset$, and no path can be found! This can be fixed by simply lowering α' until $\mathfrak{P}_{(\mathcal{V},\mathcal{E}^{+})}(\mathbf{v}, \mathbf{v}') \neq \emptyset$ and a path can be sampled. If the path succeeds, that particular problem is solved, and the agent can try to solve other problems. Otherwise, the path will fail and be inhibited, eliminating it from $\mathfrak{P}_{(\mathcal{V},\mathcal{E}^{+})}(\mathbf{v}, \mathbf{v}')$, driving enumeration until no paths are left and α' is lowered again.

But, what happens when $\alpha' = 0$ and so many edges have been inhibited that no path can be found to a goal?

8.3 Usage and Mutation Feedback-Loop

Thresholding edges by their affordance and gradually lowering the threshold can help the agent prioritize high-affordance paths, but in the end, the agent is unlikely to be able to actually enumerate $\Xi^{\mathcal{G}}_{S_\kappa}$ due to locality constraints, necessitating that it mutate the graph before trying to enumerate the paths it "missed" in the last segraph.

To handle this, we have to address a basic problem with both the failure-driven enumeration and the affordance-ordering: they are irreversible! Once an edge has been actively inhibited (added to $\mathcal{E}^{\downarrow}$) due to a traversal failure, there is no mechanism to *disinhibit* the edge, and thus, no chance of the edge ever being refined (since mutation only happens where the agent is). Likewise, once an edge has been passively disinhibited by lowering α' (moving it from $\mathcal{E}^{<}(\alpha')$ to $\mathcal{E}^{\geq}(\alpha')$), there is no mechanism to *re-inhibit* that edge. While there are many possible ways to make active inhibition and passive disinhibition reversible, we took inspiration from homeostatic mechanisms in biology.

When our agent fails to find a path because $\mathfrak{P}_{(\mathcal{V},\mathcal{E}^{+})}(\mathbf{v}, \mathbf{v}') = \emptyset$, the agent both lowers α' *and* sets $\mathcal{E}^{\downarrow} = \emptyset$, thus "clearing" the actively inhibited edges, and resetting the failure-driven enumeration. Likewise, after every traversal failure, α' is raised slightly, discouraging low-affordance (and by proxy, low-reliability) edges. This further complicates the "enumeration picture" for the agent, as now it becomes possible that edges that haven't been tried can be re-inhibited by raising α', and edges that have

already failed can fail again before other non-failed edges can even be tried. In fact, it's technically possible that some completely reliable paths will never be tried at all, because they always end up below α' before the agent can actually sample them!

Actually, it is refinement that recovers the ability of our agent to perform an enumeration. Consider that the goal for our agent, over a given segraph $\mathcal{G}$, is to enumerate $\Xi^{\mathcal{G}}_{S_\kappa}$. Suppose that to enumerate $\Xi^{\mathcal{G}}_{S_\kappa}$, the agent needs to use the edges $\mathcal{E}_{\text{req}} \subset \mathcal{E}$, with $\alpha_{\min}$ and $\alpha_{\max}$ the minimum and maximum affordances of edges in $\mathcal{E}_{\text{req}}$. By definition, $\forall \mathbf{e} \in \mathcal{E}_{\text{req}}, r_{\kappa}(\mathbf{e}) = 1$. Now suppose that there is another set $\mathcal{E}_{\text{mask}} \subset \mathcal{E}$ disjoint to $\mathcal{E}_{\text{req}}$, such that $\forall \mathbf{e} \in \mathcal{E}_{\text{mask}}, \alpha(\mathbf{e}) > \alpha_{\min}$. That is, these edges all have higher affordance than the required edge with the least affordance. It is possible that repeated failures on these edges can raise α' over $\alpha_{\min}$, masking part or all of $\mathcal{E}_{\text{req}}$. For each edge $\mathbf{e} \in \mathcal{E}_{\text{mask}}$, $r_{\kappa}(\mathbf{e}) \in (0, 1]$, since if $r_{\kappa}(\mathbf{e}) = 0$, it would mean that $\alpha(\mathbf{e}) = 0$, and thus would not be in $\mathcal{E}_{\text{mask}}$. If $r_{\kappa}(\mathbf{e}) = 1$, then there will never be a traversal failure crossing $\mathbf{e}$ to raise α'.

If $r_{\kappa}(\mathbf{e}) < 1$, then eventually, there will be an accumulation of failures and successes over $\mathbf{e}$ that will trigger its refinement. We can assume that for each child-edge, its affordance will be $r_{\text{ref}} \cdot \alpha(\mathbf{e})$, with $0 < r_{\text{ref}} < 1$. Let $\mathcal{E}_{\text{ref}}(\mathbf{e})$ be the set of "child" edges produced by refining $\mathbf{e}$. This removes $\mathbf{e}$ from $\mathcal{E}_{\text{mask}}$ and adds to $\mathcal{E}_{\text{mask}}$ each $\mathbf{e}' \in \mathcal{E}_{\text{ref}}(\mathbf{e})$ for which $\alpha(\mathbf{e}') > \alpha_{\min}$. If some of these edges have $r_{\kappa}(\mathbf{e}') = 1$, then they will never fail and thus never raise α'. Otherwise, the same logic holds, until every edge that was in $\mathcal{E}_{\text{mask}}$ is removed (because its affordance falls below $\alpha_{\min}$) or is perfectly reliable. At that point, even though there are probably new not-reachable achievement classes on this new graph, $\mathcal{G}'$, it is possible to try some subset of paths over $\mathcal{G}'$ to enumerate the achievement set of the original segraph $\mathcal{G}$, $\Xi^{\mathcal{G}}_{S_\kappa}$. The same process guarantees that, at some point, there will be a $\mathcal{G}''$ which can be used to enumerate $\Xi^{\mathcal{G}'}_{S_\kappa}$.

8.4 Extension and Linking

As long as every vertex (that needs to be refined) is eventually refined, to guarantee a balanced sequence of segraphs it is sufficient to simply extend each vertex *before* it gets refined. In practice, we modify this condition slightly, instead extending every vertex after it has been visited. This means that some vertices will actually never be extended, because they will be refined before they can be extended. However, our intuition is that, if a vertex has never been visited, then there is no reason to expect that any neighboring states can be visited. If a vertex is refined, and some of its child-vertices can be visited, then *they* can be extended. Even though not every vertex actually gets visited, the statespace (or the parts that are physically accessible) are eventually covered.

As for the linking operation, we take an approach that is, frankly, heuristic. All vertices with overlapping fields are automatically linked to each other. If our agent starts a transition in $\mathbf{v}$ trying to reach $\mathbf{v}'$, for any other vertex it passes through, $\mathbf{v}''$, it adds an edge from $\mathbf{v}$ to $\mathbf{v}''$. This provides a local mechanism for increasing the connectivity of $\mathcal{G}$ without going all the way to fully-connecting $\mathcal{G}$. This is an area for further study.

8.5 Ordering by Epistemic Value

What is "epistemic value"? In order to ultimately learn something, our agent will need to collect experiences trying to cross the edges of its cognitive graph $\mathcal{G}$, which amounts to a kind of exploration. If there are edges that haven't been traversed, the agent cannot know if they could be useful for reaching a particular goal. To ensure that "no stone is left unturned", we adopt a size-normalized count-based *epistemic-score* for edges:

$$\Upsilon(\mathbf{e}) = \frac{M(\mathbf{e})}{n(\mathbf{e}) + n_{\Upsilon}(\mathbf{e}) + 1} \tag{8.2}$$

where $n(\mathbf{e}) = n_s(\mathbf{e}) + n_f(\mathbf{e})$, 1 is added for numerical stability, and $n_{\Upsilon}(\mathbf{e})$ is an extra variable that can be used to adjust an edge's score independently of the number of actual visitations. The point of this score is simple: larger edges (greater $M(\mathbf{e})$) have more potential plans to try, and less-visited (lower $n(\mathbf{e})$) edges have less certain robustness estimates. This means that while exploring, our agent doesn't pick a *vertex* to find a path toward, but rather an *edge* $\mathbf{e}^\star$ to find a path toward. We can slightly adapt our notation to reflect this: $\mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{e}^\star)$ is the set of all paths over $\mathcal{G}$ that start at $\mathbf{v}$ and end by crossing $\mathbf{e}^\star$: if $\mathbf{e}^\star = (\mathbf{v}', \mathbf{v}^\star)$, then $\mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{e}^\star) = \{\mathfrak{p} + \mathbf{v}^\star : \mathfrak{p} \in \mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{v}')\}$.

We can model our agent's choice of a goal with the discrete probability distribution $\mathcal{G}_{\Upsilon}$ over all of the edges $\mathcal{E}$ over its segraph. Without being prematurely specific, we can say that $\mathcal{G}_{\Upsilon}(\mathbf{e})$ is a monotonically increasing function of $\Upsilon(\mathbf{e})$, so the higher $\Upsilon(\mathbf{e})$ is, the more likely $\mathbf{e}$ is to be selected as a goal. If the agent reaches the edge $\mathbf{e}$ and tries to cross it (successfully or unsuccessfully), then $n(\mathbf{e})$ will increment by 1 and $\Upsilon(\mathbf{e})$ will decrease, thereby decreasing the chance that $\mathbf{e}$ is selected as a goal. This doesn't automatically mean that the agent will attempt to reach every edge in $\mathcal{E}$ "in-order", but under some mild ergodic assumptions, it does mean that every edge will eventually be visited (or, *attempt* to be visited). However, reaching every edge is not the same as trying every path!

We model our agent's choice of a path from a starting vertex to a goal edge with the discrete probability distribution $\mathcal{P}^{\mathbf{v}\to\mathbf{e}^\star}_{\mathcal{G}}$ over the set of paths $\mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{e}^\star)$. The final form of this distribution will be determined by several factors not yet introduced, but for now, it is sufficient to begin by assuming that $\mathcal{P}^{\mathbf{v}\to\mathbf{e}^\star}_{\mathcal{G}}$ is uniform over $\mathfrak{P}^*_{\mathcal{G}}(\mathbf{v}, \mathbf{e}^\star)$, making any path from $\mathbf{v}$ to $\mathbf{e}^\star$ equally likely. If that is the case, then randomly sampling targets from $\mathcal{G}_{\Upsilon}$ and paths from $\mathcal{P}$ guarantees that eventually, every path in $\mathfrak{P}^*_{\mathcal{G}}$ is eventually tried, giving us a trivial enumeration of $\mathfrak{P}^*_{\mathcal{G}}$.

8.6 Ordering by Simplicity

All things being equal, a shorter path is a better path. If $\mathfrak{P}^k_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v}, \mathbf{e})$ is the set of all paths from $\mathbf{v}$ to $\mathbf{e}$ of length k, then $\mathfrak{P}^*_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v}, \mathbf{e}) = \bigcup_{n=1}^{\infty} \mathfrak{P}^k_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v}, \mathbf{e})$. Our agent can first try paths in $\mathfrak{P}^1_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v}, \mathbf{e})$, if a path is found that works, then a single instance of $\xi^{\mathcal{G}}_{\mathbf{v},\mathbf{e}}$ has been found and there is no need to try longer paths. If no length-1 path is found, the agent can begin trying paths in $\mathfrak{P}^2_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v}, \mathbf{e})$, and so on until a successful path is found.

In fact, in combination with the failure-driven inhibition logic (see section 7.2), our

agent can get away with *only* sampling from among the shortest available paths. Let $k_{\min}$ be the smallest k for which $\mathfrak{P}^k_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v},\mathbf{e}) \neq \emptyset$, and assume that:

$$\mathcal{P}^{\mathbf{v}\to\mathbf{e}}_{(\mathcal{V},\mathcal{E}^+)}(\mathfrak{p}) = \begin{cases} 0 & \text{if } |\mathfrak{p}| > k_{\min} \\ p(\mathfrak{p}) & \text{otherwise, with } p(\mathfrak{p}) > 0 \end{cases} \tag{8.3}$$

If a plan sampled from $\mathcal{P}^{\mathbf{v}\to\mathbf{e}}_{(\mathcal{V},\mathcal{E}^+)}$ succeeds, then the search for an element in $\xi^{\mathcal{G}}_{\mathbf{v},\mathbf{e}}$ is complete. Otherwise, every plan sampled from $\mathcal{P}^{\mathbf{v}\to\mathbf{e}}_{(\mathcal{V},\mathcal{E}^+)}$ fails, and by definition, is removed from further consideration: after each plan in $\mathfrak{P}^k_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v},\mathbf{e})$ fails, $\mathfrak{P}^k_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v},\mathbf{e})$ is empty! This means that $k_{\min} \neq k$, and instead, $k_{\min} > k$, meaning that when our agent samples a new plan, it will automatically come from the next "length-set" of plans, $\mathfrak{P}^{k+n}_{(\mathcal{V},\mathcal{E}^+)}(\mathbf{v},\mathbf{e})$. Thus, selecting shortest paths along with failure-induced inhibition automatically induces an ordering of paths by length.

Chapter 9

Adaptive Realtime Metasearch over Segraphs

To recap: we imagine an agent $\mathcal{A}$ that wants to be able to reach any point in an abstract statespace $S = \mathbb{R}^d$ from any other point in S. It is endowed with a continuous sensorimotor controller K, which cannot fully navigate S on its own. We equip $\mathcal{A}$ with a segraph $\mathcal{G}$ that decomposes S into chunks that K can more easily navigate between, reaching distant goals via paths over $\mathcal{G}$. By following a path over the segraph, the agent collects experiences that inform it about the traversability of individual edges. This information changes the probability distribution over paths, which changes which plans are tried by the agent, which subsequently, changes how the segraph itself is modified, creating a tight behavior/segraph feedback loop. Effectively, this feedback loop involves a search over paths driving (and being driven by) a search over segraphs, which occurs in real-time, and is adaptive. Because of this, we call the algorithm the Adaptive Real-time Metasearch over Segraphs, or ARMS. We describe the ARMS algorithm, providing motivation in terms of balancing segraph mutation. For the description of the algorithm itself, we actually do not assume that the agent has a fixed goal: rather, we imagine the agent's goal is simply to explore as much of the space of actions as possible, so that any particular goal can be reached on-demand.

9.1 Algorithm

Each edge has an *estimated affordance* $\tilde{\alpha}(\mathbf{e})$ which is compared to a threshold α'. If $\tilde{\alpha}(\mathbf{e}) < \alpha'$, then $\mathbf{e}$ is *inhibited*. Inhibiting an edge prevents it from being used during pathfinding, allowing entire classes of paths to be instantaneously turned off, or if an edge is disinhibited, turned back on. Edges can also be manually (dis)inhibited, independent of their α-value. Disinhibiting low α edges encourages revisiting less reliable or more-fine-grained options, and inhibiting low α edges encourages the use of more reliable paths. A major element that controls the behavior of the algorithm is the exact way that α' is adapted. Because pathfinding can be computationally expensive, trying to find a path and failing often is very expensive, so lowering α' too slowly can be catastrophic for computational efficiency. On the other hand, lowering it too fast can make pathfinding under-selective. The same is true of raising α'. We represent the rate of lowering α' by Δ^-, and the rate of raising it by Δ^+.

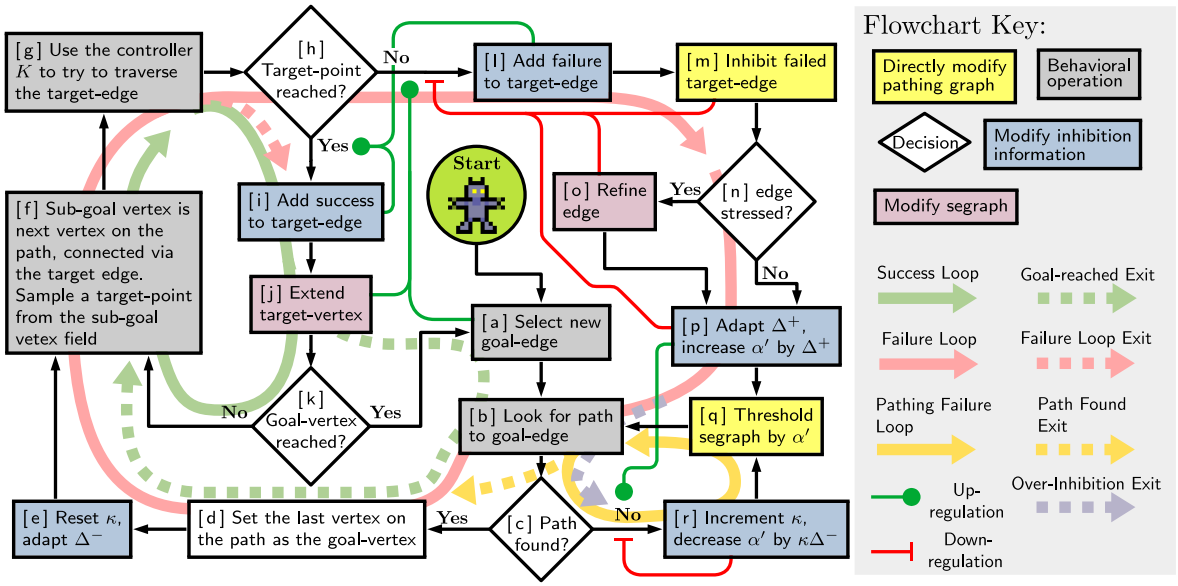


Figure 9.1: The search algorithm can be decomposed into three major interconnected loops: the first is the "success" loop (solid green arrow [f, g, h, i, j, k]). Staying on this loop results in reaching the selected goal, triggering the selection of a new goal and ideally leading back to the success loop (dashed green arrow). However, selection of ambitious goals and mismatch between $G^\star$ and the environment will usually lead to failed edge-traversals, triggering the "failure-loop" (solid red arrow, [f, g, h, l, m, n, o?, p, q, b, c, d, e]). Both the success and failure loops update the reliability estimates of edges, allowing the agent to more easily distinguish good from bad edges, which long-term, promotes the success-loop. Repeated failure raises α' and can result in a pathing failure (over-inhibition exit). Each pathing failure lowers α', so the "pathing failure loop" (yellow arrow) is self-inhibiting, and will eventually lead back to the success or failure loops.

Fig. 9.1 shows a basic conceptual overview of the algorithm as a flowchart. The agent starts out at a point inside of the field of a *start vertex* $\mathbf{v}^\circ$ (the segraph $\mathcal{G}$ may have an arbitrary initial configuration, perhaps with only a single vertex). [a] The agent calculates an *epistemic score* $\Upsilon(\mathbf{e})$ for each edge $\mathbf{e}$, equal to an edge's size divided by its total number of traversals. Then, the agent selects the edge with the highest Υ and sets it as its *goal-edge*, $\mathbf{e}^\star$, analogous to various *count-based* exploration methods [142]. $\mathbf{e}^\star$ is manually *dis*inhibited, and all manual inhibition is cleared, then [b] the agent uses its pathfinding algorithm to sample a path from $\mathcal{P}^{\mathbf{v}^\circ \to \mathbf{e}^\star}_{(\mathcal{V},\mathcal{E}^+)}$ to find a path $\mathfrak{p}$ beginning at $\mathbf{v}^\circ$ and ending (by passing through) the edge $\mathbf{e}^\star$. [c] If such a path is found, [d] the last vertex on $\mathfrak{p}$ is designated the *goal-vertex*, $\mathbf{v}^\star$. [e] Then Δ^- is adapted to regulate the number of pathfinding failures (we will return to this).

Next [f] the agent takes the next vertex from the path and sets it as its *sub-goal vertex* $\mathbf{v}^\triangleright$, the edge between $\mathcal{A}$'s current vertex $\mathbf{v}^\circ$ and the subgoal vertex is designated the *target edge* $\mathbf{e}^\triangleright$, which $\mathcal{A}$ will try to traverse next. $\mathcal{A}$ uniformly samples a target point from $F(\mathbf{v}^\triangleright)$ and passes the displacement to its controller K. Then [g] $\mathcal{A}$ tries to physically traverse the edge, ultimately ending up at a new point $\boldsymbol{s}'$. [h] If $\mathcal{A}$ reached the target point, [i] the traversal is marked as a success, and the agent updates its current vertex to $\mathbf{v}^\circ \leftarrow \mathbf{v}^\triangleright$, after which [j] the vertex the agent just reached is extended. Then [k] the agent checks if it has reached the goal vertex $\mathbf{v}^\star$. If it has, the agent selects a new goal (back to [a]), otherwise, the agent continues along the path [f].

If at [h] the agent *doesn't* reach the target point, then [l] the traversal is marked

as a failure, and the agent recalculates what vertex it is in. [m] $\mathcal{A}$ *manually inhibits* the failed edge (inhibited independently of its actual α-value) by adding it to $\mathcal{E}^{\downarrow}$ so that it can't immediately be used again in a path. If the edge is stressed ($\Delta\tilde{q}(\mathbf{e}) > 1$), the edge is refined, in the hopes that some of the child-edges are more reliable than the original edge. Here, the c term in the stress score becomes a *control parameter* that determines the overall refinement rate of the algorithm. Then, Δ^{+} is adapted to reflect the difference between the activity potential of the edge and α', and α' is increased to discourage further failures on other edges.

Then [q] the graph is thresholded by α', producing a new pathing graph so that [b] $\mathcal{A}$ can look for a new path. Repeated failures may result in α' being so high that a path cannot be found between $\mathbf{v}^{\circ}$ and the goal edge $\mathbf{e}^{\star}$, resulting in a pathing failure ([c] branches to [r]), which increments κ (the number of consecutive pathing failures) and decreases α' by $\kappa\Delta^{-}$ (decreasing the odds of another pathing failure), which then goes back to [q]. Jumping back to [e], upon pathfinding *success*, κ is reset back to 0, and Δ^{-} is adapted to reflect the difference between α'_0 (initial threshold value before pathing failure) and the current threshold value, to regulate the number of consecutive pathing failures necessary before the threshold is low enough to find a path. This controls the balance between pathfinding selectivity and pathfinding computational cost. Pathing failure also clears all manual inhibition to ensure that a path can be found. Additionally, this also triggers $n_{\Upsilon}(\mathbf{e}^{\star})$ to be incremented, to discourage "fixation" on an unreachable goal.

The ARMS algorithm has several means of self-regulation. Overall the algorithm has three interlocking loops, shown in Fig. 9.1 as thick colored arrows: the "success", "failure", and "pathing failure" loops. Edge-traversal collects information that helps the agent discriminate between edges, so long-term the success loop is up-regulated by both [i] (the success loop) and [l] (the failure loop). However, the success loop causes the addition of new edges via extension [j] and reliability-agnostic selection of goal-edges [a], which causes the success loop to up-regulate the failure loop. The failure loop down-regulates itself by (1) inhibiting an edge after failure to prevent immediate repeat-failure, (2) potentially refining an edge to create better options for pathing in the long-term, and (3) raising α' to lower the chance of other low-reliability edges being used. This, however, also up-regulates the "pathing-failure loop", as over-inhibition can prevent a path from being found. To counteract this, the pathing-failure loop down-regulates itself by *lowering* α', allowing the system to return to either the success or failure loops. See D.2 for details of α' adaptation.

9.2 Resource constraints and Prior Knowledge

One important addition of the algorithm, strictly beyond the scope of our theory, is how to deal with resource constraints. With a fixed size neural network for representing the segraph, there is an upper limit on how many vertices the segraph can have. We could just stop the algorithm at this point, but instead, we use an ad-hoc score (see D.4) to determine which vertices might be acceptable to delete. As the agent gets closer to its built-in vertex limit, it slows down the rate of refinement (by lowering c), and once it reaches the vertex-cap, it starts to delete apparently useless vertices.

The slowdown in refinement allows the segraph to naturally stabilize to a more-or-less "final" configuration, which is only further modified by better and better estimates of edge-reliability.

We were also interested in testing the incorporation of prior knowledge into the ARMS algorithm. The algorithm described above corresponds to a "naive" agent. The simplest mechanism we could think of was to allow the ARMS algorithm some ability to decide how large new vertex-fields should be, based on sensory data. If a wide area near the agent seems to be open, then the agent can decide, when extending a vertex, that it should place a larger field there: if the area is instead a barrier, it can decide to add much smaller vertices there (making them more likely to be inhibited by the affordance threshold). We call such an agent "astute". The "sensory information" that we give the astute agent is just the value $r_{K}(\boldsymbol{s}, \boldsymbol{s'})$, it then samples points $\boldsymbol{s'}$ around a field that needs to be extended, calculates $r_{K}(\boldsymbol{s}, \boldsymbol{s'})$, then stores that value in a HaRK memory (Fig 11.1A). At the same points, it performs a query at different resolutions: starting at a low resolution (blurring stored r_{K} values together on a large spatial scale), it checks for points with high average local r_{K}, adding new vertices at that scale if it's above a threshold. Any points that remain uncovered are queried at a higher and higher resolution, until the vertex is fully extended. We can model a "misled" agent by simply inverting $r_{K}(\boldsymbol{s})$.

Chapter 10

Neural and Algorithmic Instantiation

Aside from the theoretical interest I have in characterizing and mimicking the adaptive abilities of living organisms, I'm also interested in demonstrating that, at least in principle, my method of *achieving* hyperadaptability can be implemented in the nervous system. Rather than try to develop a very detailed neurobiological model, I focused on a more abstract implementation of artificial neural networks, relying heavily on linear algebra and Fourier theory.

In particular, I take inspiration from the hippocampus and entorhinal cortex. I consider that a *segraph* can be thought of as a very simplified model of a *cognitive graph* and a *cognitive map*. Place cells [120] in the hippocampus and grid cells [121] in the entorhinal cortex have been implicated as instantiating cognitive graphs and maps, providing a neurological basis for these abstract constructs. I associated place-cells with the vertices of a segraph, and I associated grid cells and band cells with the mechanism for granting vertices their fields.

I use a "binding" operator corresponding to a mathematically simple form of one-shot Hebbian learning, both to construct the graph, and to ground it in the state-space S. I choose to represent graph-vertices as one-hot vectors, as this seems to provide a reasonable facsimile of place cells, and gives us some technical benefits I will describe later. The state-space is represented by a complex-valued model of *band cells* [143]. By "binding" vertex-vectors together, we encode the *topology* of the segraph, and by binding vertex-vectors to band-cell population vectors, we endow the vertices with their *fields*.

10.1 Hebbian Learning

The segraph itself must be able to be modified rapidly in order to achieve hyperadaptability through the means I have already outlined. This rapid mutability is achieved by defining a simple Hebbian-type learning rule that can accomplish the association of arbitrary vectors in a single operation, which I refer to as *binding*.

Definition 10.1 (Binding). The *bind* between two vectors $\boldsymbol{a} \in \mathbb{C}^N$ and $\boldsymbol{b} \in \mathbb{C}^M$ is the matrix:

$$\boldsymbol{a} \triangleright \boldsymbol{b} = \frac{\boldsymbol{b}\boldsymbol{a}^*}{\|\boldsymbol{a}\|^2} \in \mathbb{C}^{M \times N}$$

where if $\boldsymbol{c} \in \mathbb{C}^N$, then $(\boldsymbol{a} \triangleright \boldsymbol{b})\boldsymbol{c} = \boldsymbol{b}\frac{\|\boldsymbol{c}\|}{\|\boldsymbol{a}\|}\langle\!\langle \boldsymbol{a}, \boldsymbol{c} \rangle\!\rangle$, with $\langle\!\langle \boldsymbol{a}, \boldsymbol{c} \rangle\!\rangle = \frac{\boldsymbol{a}^*\boldsymbol{c}}{\|\boldsymbol{a}\| \cdot \|\boldsymbol{c}\|}$ the cosine similarity between $\boldsymbol{a}$ and $\boldsymbol{c}$.

We can interpret a sum of such "bind" matrices as a dictionary, as

$$\left[\sum_{i=1}^{k} \boldsymbol{a}_i \triangleright \boldsymbol{b}_i\right] \boldsymbol{c} = \sum_{i=1}^{k} (\boldsymbol{a}_i \triangleright \boldsymbol{b}_i)\boldsymbol{c} = \sum_{i=1}^{k} \boldsymbol{b}_i \frac{\|\boldsymbol{c}\|}{\|\boldsymbol{a}_i\|} \langle\!\langle \boldsymbol{a}_i, \boldsymbol{c} \rangle\!\rangle \tag{10.2}$$

If all of the "key" vectors $\boldsymbol{a}_i$ are orthogonal to each other (have zero cosine-similarity), and $\boldsymbol{c} = \boldsymbol{a}_j$, then:

$$\left[\sum_{i=1}^{k} \boldsymbol{a}_i \triangleright \boldsymbol{b}_i\right] \boldsymbol{a}_j = \sum_{i=1}^{k} \boldsymbol{b}_i \frac{\|\boldsymbol{a}_j\|}{\|\boldsymbol{a}_i\|} \langle\!\langle \boldsymbol{a}_i, \boldsymbol{a}_j \rangle\!\rangle = \boldsymbol{b}_j \frac{\|\boldsymbol{a}_j\|}{\|\boldsymbol{a}_j\|} \langle\!\langle \boldsymbol{a}_j, \boldsymbol{a}_j \rangle\!\rangle = \boldsymbol{b}_j \tag{10.3}$$

meaning it's possible to selectively retrieve stored associations. This approach is comparable to [116] and [144]. See appendix B for details.

10.2 Graph Working Memory

Let $G = (V, E)$ be a directed graph with vertices $V$and edges E. Each vertex $\mathbf{v}_i \in V$is assigned a unique M-dimensional one-hot vector $\boldsymbol{v}_i$, with all elements 0 except for the i^{th}, which is 1. If an edge $\mathbf{e} \in E$ is an ordered tuple of vertices with one-hot vectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, then the edge $\mathbf{e}_{i,j}$ is represented by a matrix $\mathbf{E}_{i,j} = \boldsymbol{v}_i \triangleright \boldsymbol{v}_j$. The graph G is represented by the matrix $\mathbf{G} = \sum_{\mathbf{e}_{i,j} \in E} \mathbf{E}_{i,j}$, so that for any vertex $\mathbf{v}_i$, $\mathbf{G}\boldsymbol{v}_i$ will be the sum of one-hot vectors for the neighbors of $\mathbf{v}_i$ (Fig. 10.1A). This makes $\mathbf{G}$ essentially an adjacency matrix. Being represented by one-hot vectors, they are easy to individually identify.

10.3 Wave-based Pathfinding

As plans must be sampled from paths, and paths must be computed, efficient online pathfinding is essential for our agent. This algorithm could in principle be substituted for any standard pathfinding algorithm, but I designed this algorithm with two things in mind: first, taking advantage of parallelism (at least, in a physical neural substrate), and second, the intuition that when imagining a plan to reach a goal, it is common to *first* imagine some "intermediate" steps, and gradually decompose the problem into more and more fine-grained intermediate steps.

I take advantage of the one-hot-vector graph-representation to implement the $\mathcal{P}_{\circledcirc}$ pathfinding algorithm. By propagating across the graph a forward "wave-front" from a

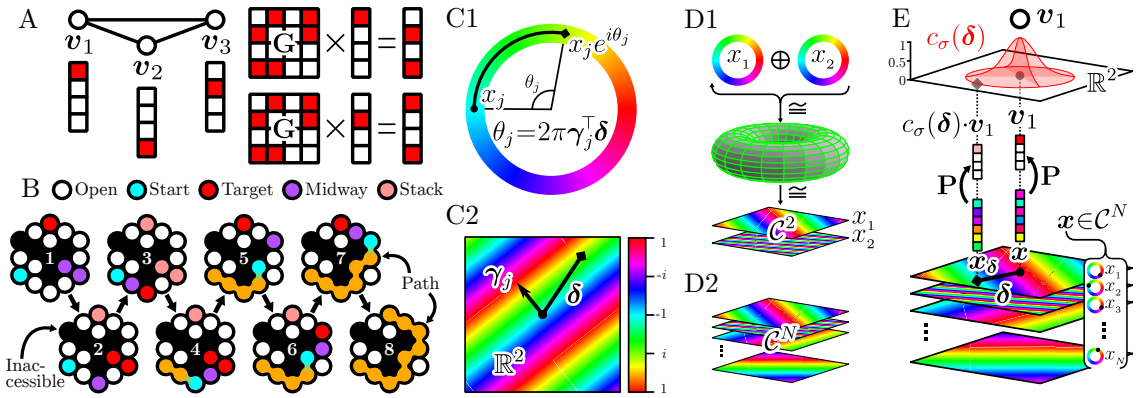


Figure 10.1: (A) Each vertex of the graph is assigned a one-hot vector, which are bound together via Hebbian learning in a matrix $\mathbf{G}$. (B) An example of the recursive decomposition used by the pathing algorithm, excluding wave-propagation for brevity. Adjacent cells are connected, pathing proceeds by recursively finding the vertices midway between some start and target vertices. (C1) With $\mathcal{C}$ the set of unit-complex numbers, each band-cell's state is $x_j \in \mathcal{C}$. An update $\boldsymbol{\delta} \in \mathbb{R}^d$ projects onto the oriented frequency $\boldsymbol{\gamma}_j$ of the band-cell. (C2) The j^{th} band-cell's state is a sinusoidal grating in $\mathbb{R}^d$ ($d = 2$ example shown). (D1) The Cartesian product of two such cells is a torus $\mathcal{C}^2$. (D2) In higher dimensions I just show the stacked gratings. (E) In order to ground the graph in space, each vertex one-hot vector is bound via Hebbian learning inside the matrix $\mathbf{P}$ to a vector of band-cell activations $\boldsymbol{x}_{\boldsymbol{p}}$ encoding a specific point in space $\boldsymbol{p} \in \mathbb{R}^d$. The Hebbian learning is modulated so that the response of the vertex-vector decays as a Gaussian with respect to displacement from the "center" of the stored location. By binarizing this activity, we get a "place-field".

set of starting vertices and a backward "wave-front" from some target vertices (compare to [145]), we can find the set of "mid-point" vertices where the waves overlap in $O(k)$ matrix multiplications, where k is the length of the shortest path between the start and target vertices. By treating the mid-point vertices as a new set of target vertices and ignoring the cost of matrix multiplication, $\mathcal{P}_{\circledcirc}$ recursively solves the pathfinding problem in time $O(k \log(k))$, and moreover, finds the *first* vertex on the path in time $O(k)$, meaning the agent can begin following the path before the whole path has been found. A toy-example is shown in Fig. 10.1B.

The pathfinding algorithm $\mathcal{P}_{\circledcirc}$ proceeds by recursive application of a simple midway-point-finding process. A combination of passive α'-thresholded inhibition and manual inhibition over a segraph $\mathcal{G}$ will yield a subgraph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}^+)$ that is used for pathfinding. These edges can be represented via Hebbian learning by a matrix $\mathbf{G}$. The $\mathcal{P}_{\circledcirc}$ algorithm takes as input a set of starting vertices $V'_s \subset V$ and a set of goal vertices $V'_g \subset V$, which have corresponding multi-hot vector representations $\boldsymbol{m}'_s$ and $\boldsymbol{m}'_g$. Then, a "forward wave-front" and "backward wave-front" propagate out from $\boldsymbol{m}'_s$ and $\boldsymbol{m}'_g$ respectively, until the two wavefronts overlap at a set of vertices represented by the multi-hot vector $\boldsymbol{m}$. The "goal" vertices $\boldsymbol{m}'_g$ are pushed to a stack, and the process repeats with $\boldsymbol{m}'_s$ and $\boldsymbol{m}$ representing a "sub-problem".

This process repeats recursively, until the sub-problem is trivial. At this point, vertices may be chosen and added to the agent's path, with the last sub-goal popped from the stack. See D.5 for details.

10.4 Harmonic Relational Keys

Having introduced my method of constructing cognitive *graphs*, I must now explain how to construct cognitive *maps* representing $\mathbb{R}^d$ state-spaces. I take inspiration from band cells, and represent the state of each band cell as a unit-magnitude complex number $x_j \in \mathbb{C}$, with an oriented frequency $\boldsymbol{\gamma}_j \in \mathbb{R}^d$ (Fig. 10.1C). The population of band cells is a vector $\boldsymbol{x} \in \mathbb{C}^N$ (N large, Fig. 10.1D), and each $\boldsymbol{\gamma}_j$ is sampled from a distribution $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$ over $\mathbb{R}^d$. If we associate the key $\boldsymbol{x}_{\boldsymbol{p}}$ with a point $\boldsymbol{p} \in \mathbb{R}^d$, then the key corresponding to the point $\boldsymbol{p} + \boldsymbol{\delta}$ is given by

$$\boldsymbol{x}_{\boldsymbol{p}+\boldsymbol{\delta}} = f_{\text{HaRK}}(\boldsymbol{x}_{\boldsymbol{p}}, \boldsymbol{\delta}) = \boldsymbol{x}_{\boldsymbol{p}} \odot e^{2\pi i \boldsymbol{\Gamma}\boldsymbol{\delta}} \tag{10.4}$$

where $\odot$ is an element-wise product, and $\boldsymbol{\Gamma}$ is the matrix of oriented frequencies $\boldsymbol{\gamma}_j$.

f_{HaRK} can assign a unique $\mathbb{C}^N$ key to each point in $\mathbb{R}^d$. We can use element-wise modulation to also control the *resolution* of information stored in $\mathbb{R}^d$. The bind matrix between an input-modulated key-vector $\boldsymbol{x} \odot \boldsymbol{\mu}$ and the value-vector $\boldsymbol{y}$ is $[(\boldsymbol{x} \odot \boldsymbol{\mu}) \triangleright \boldsymbol{y}]$. We then query this matrix with a $\boldsymbol{\delta}$-offset key $\boldsymbol{x}_{\boldsymbol{\delta}} = f_{\text{HaRK}}(\boldsymbol{x}, \boldsymbol{\delta})$ modulated by $\boldsymbol{\eta}$, yielding

$$\tilde{\boldsymbol{y}}(\boldsymbol{\delta}) = [(\boldsymbol{x} \odot \boldsymbol{\mu}) \triangleright \boldsymbol{y}](\boldsymbol{x}_{\boldsymbol{\delta}} \odot \boldsymbol{\eta}) \tag{10.5}$$

which can be decomposed as $\tilde{\boldsymbol{y}}(\boldsymbol{\delta}) = c(\boldsymbol{\delta})\boldsymbol{y}$, where $c(\boldsymbol{\delta}) = \mathcal{F}_{\boldsymbol{\gamma}}^{-1}[g(\boldsymbol{\gamma})p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})]$, with $\mathcal{F}^{-1}$ being the inverse Fourier transform and $g(\boldsymbol{\gamma})$ a function mapping oriented frequencies $\boldsymbol{\gamma}$ to modulation weights: in other words, $g(\boldsymbol{\gamma})$ determines $\boldsymbol{\mu}$ and $\boldsymbol{\eta}$. For this initial work I choose $c(\boldsymbol{\delta})$ to be an isometric Gaussian $c_\sigma(\boldsymbol{\delta})$ with width σ (Fig. 10.1E), though this is not strictly necessary. Manipulating σ can be used to control the resolution of information on storage, on retrieval, or both to achieve statespace band-pass filtering.

This approach to harmonic representations can be thought of as a generalization and simplification of the model presented in [146]. See appendix C for details.

Chapter 11

Results

Initial development of the ARMS algorithm occurred in 2D mazes, since this allowed for easy visual inspection and intuitive understanding. I began by testing the ability of the ARMS algorithm in its naive, astute, and misled variants to learn to navigate around a fixed set of regular rectangular and irregular polygonal mazes. To measure the agent's ability in a way that was irrespective of maze-size, I calculate the agent's segraphs' *reliability* $R(\mathcal{G})$, which is a distance-weighted average of the reliability of the graph for navigating between pairs of points sampled from the state-space, allowing the agent to re-path if necessary to reflect the operation of the ARMS algorithm (see Methods D.7). $R(\mathcal{G}) = 0$ means complete unreliability, $R(\mathcal{G}) = 1$ means total reliability. $R(\mathcal{G})$ can be thought of as a distance-normalized and environment-size-normalized variant of the affordance of the graph, $\alpha(\mathcal{G})$. I also checked the resilience of ARMS to shifts in the environment, its sensitivity to initial field-size guess, and its ability to handle higher-dimensional state-spaces. Finally, I tested ARMS in a dynamic environment with rewards.

11.1 2D mazes

To build mazes, I generated randomly connected 25-node graphs, converting the graph into a maze by 1) finding a random spanning tree of the graph and 2) converting the topology of the spanning tree into a continuous geometric environment with barriers. Two 5x5 mazes (5x5(1) and 5x5(2)), a triangular maze, and a pentagonal maze were randomly generated (see Methods D.6) and used to determine if 1) the ARMS algorithm could successfully learn to navigate around these mazes and 2) whether incorporating prior knowledge about environment navigability could influence the ability of the ARMS algorithm to learn these mazes (Figure 11.1A). I ran 8 simulations for each condition (maze × prior knowledge) for 200,000 timesteps, measuring $R(\mathcal{G})$ every 10,000 timesteps (because the measurement is quite computationally expensive to make). The median (solid line) and interquartile range (shaded area) are shown for each condition in Figure 11.1. Across all conditions, ARMS achieves nearly perfect navigation ability within approximately 30,000 timesteps, and as the first four rows of Table 11.1 show, by the end of each simulation, all ARMS agents reached perfect navigation ability across all mazes and all prior knowledge conditions. At the beginning, it appears that having some accurate prior knowledge does speed up learning, but

surprisingly, misleading prior knowledge only seems to have a negative effect compared to the naive agent in the Triangular maze, suggesting that our means of adding prior knowledge to ARMS is conceptually shallow and does not correspond to the kinds of prior knowledge that we might expect animals or humans to have.

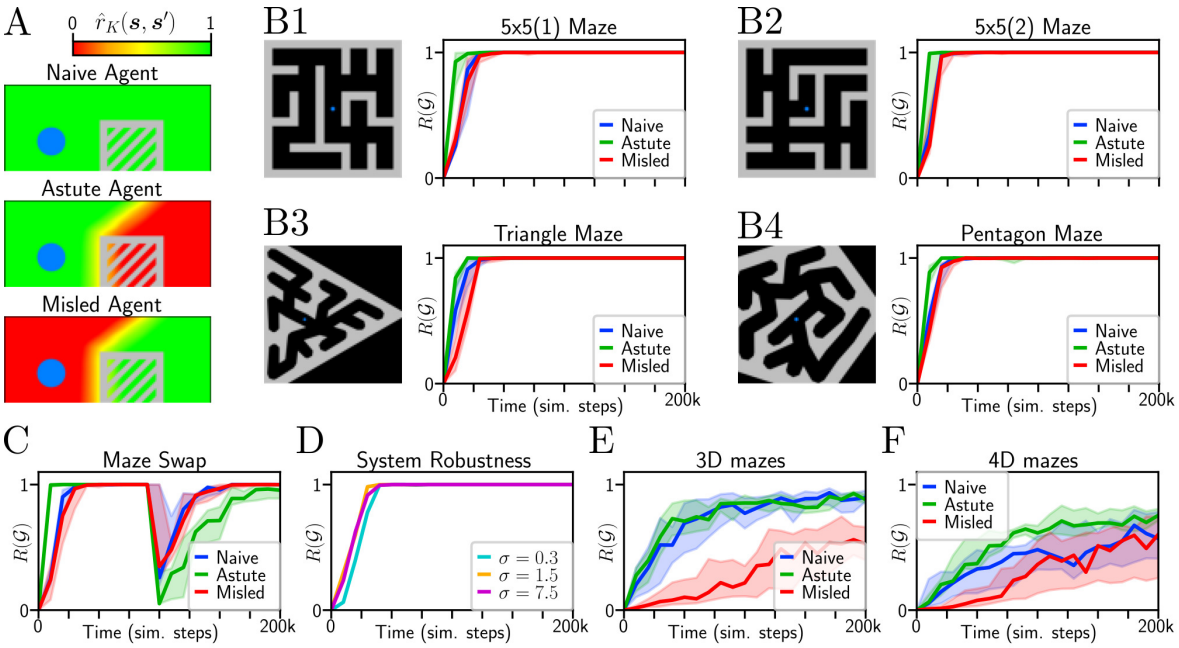


Figure 11.1: (A) Agents can have different knowledge of spatial navigability, either being Naive, Astute, or Misled. (B) Naive, Astute, and Misled agent's ability to construct a reliable segraph in different maze-environments. (C) Naive, Astute, and Misled agent's ability to recover when the maze is changed (at 100k sim. steps). (D) The robustness of the ARMS algorithm to different initial field sizes. (E) Naive, Astute, and Misled agents in random 3D mazes. (F) Naive, Astute, and Misled agents in random 4D mazes.

Next, I was interested in the ability of the ARMS algorithm to recover from major environmental perturbations. To simulate this, I started naive, astute, and misled agents inside of the 5x5(1) maze, then at 100,000 timesteps, swapped to the 5x5(2) maze, and continued the simulation for another 100,000 timesteps. As before, I measured $R(\mathcal{G})$ every 10,000 timesteps, ran 8 trials per condition, and plot median and interquartile range (Figure 11.1C). It seems that prior knowledge (the astute agent) speeds up early convergence, but it appears that this initial strength comes at a cost: the astute agent recovers from the perturbation more slowly than the naive and misled agents, which both manage to recover to perfect navigability of the new mazes by the end of the simulation (see Table 11.1).

If the ARMS algorithm creates too many tiny fields, it can waste its time on irrelevant details of the environment, while if the ARMS algorithm only creates large fields, it may never actually master the maze. The ARMS algorithm starts with an initial state with a predefined field-size, which is a hyperparameter. The size of this initial field will control to some extent the size of subsequently added fields. How sensitive is the operation of the ARMS algorithm to this initial choice? I tested the naive agent in the 5x5(1) maze across three different initial vertex sizes ($\sigma = 0.3$, $\sigma = 1.5$, $\sigma = 7.5$), measured by the size of their Gaussian activation functions. Running 8 trials

Table 11.1: $R(\mathcal{G})$ values after 200,000 time-steps of simulation, reported for the Naive, Astute, and Misled agents in each maze as (Q1, median, Q3).

Maze	Naive Agent	Astute Agent	Misled Agent
5x5(1)	(1.000, **1.000**, 1.000)	(1.000, **1.000**, 1.000)	(1.000, **1.000**, 1.000)
5x5(2)	(0.999, **1.000**, 1.000)	(1.000, **1.000**, 1.000)	(1.000, **1.000**, 1.000)
Triangle	(1.000, **1.000**, 1.000)	(1.000, **1.000**, 1.000)	(1.000, **1.000**, 1.000)
Pentagon	(1.000, **1.000**, 1.000)	(1.000, **1.000**, 1.000)	(1.000, **1.000**, 1.000)
Maze Swap	(1.000, **1.000**, 1.000)	(0.888, **0.955**, 1.000)	(0.998, **0.999**, 1.000)
3D Mazes	(0.864, **0.897**, 0.951)	(0.851, **0.880**, 0.921)	(0.397, **0.526**, 0.661)
4D Mazes	(0.410, **0.570**, 0.781)	(0.712, **0.755**, 0.806)	(0.256, **0.604**, 0.745)

per condition for 200,000 timesteps, we find that there is only a weak initial sensitivity to initial field size on performance: in all cases, the ARMS algorithm quickly achieves perfect navigation ability in the maze.

11.2 Higher dimensional Mazes

I also wanted to see if the ARMS algorithm could work in higher-dimensional state-spaces. To test this, I generated random 3D and 4D mazes composed of overlapping capsules to make collision-checking simple, with topological complexity equivalent to the 2D mazes I initially tested the ARMS algorithm on. Again, I simulated naive, astute, and misled ARMS agents for 200,000 timesteps, though this time I ran 16 trials per condition due to the high variability induced by randomizing mazes per-trial. An additional difference is that, due to the computational cost of collision-checking in these mazes, I allowed the agent to "jump" towards it's target, stopping at any obstacles along the way. This makes the effective time-scales of the physics a bit faster compared to the 2D mazes.

In smaller mazes (not shown), ARMS handily solves 3 and 4 dimensional mazes, but in 25-node 3D and 4D mazes (in theory of similar complexity to the 25-node 2D mazes that it can master) it begins to struggle. Strangely, the relationship between prior knowledge and ability begins to become less logical. In 3D mazes, it appears that the Naive agent performs better than the Astute agent, with even the Misled agent slightly surpassing the Astute agent by the end of the simulation. In 4D, the situation is even stranger: while the Astute agent performs almost as well in 4D as it did in 3D, the Naive agent's performance totally collapses, with the Misled agent splitting the difference. These patterns suggest that our method of adding prior knowledge to ARMS is conceptually inadequate. I tested an alternative method of choosing when to refine an edge, replacing the stress score $\Delta\tilde{q}$ with just the number of failures multiplied by a constant, and reran the 3D (Figure 11.1E) and 4D (Figure 11.1F) simulations, which yielded overall similar results but a more logical relationship between prior knowledge and maze-navigation ability.

11.3 MountainCar

I further tested the ARMS algorithm to see if it could perform well on a standard reinforcement-learning task, continuous MountainCar. Instead of the statespace being 2-spatial dimensions, it is 1-space dimension and 1-velocity dimension. This makes low-level control far less trivial, as it is usually impossible to just "move towards the goal", due to the coupling between these two dimensions. I used a PID controller with hand-tuned gains for ARMS. I tested against a strong baseline, Proximal Policy Optimization with Random Network Distillation, equipped with the exact same low-level controller as used by the ARMS algorithm, to keep the baseline fair. This baseline, technically PPO-RND-PID, I refer to simply as "PPO+". I found on ordinary MountainCar that both algorithms converged extremely fast, because using a PID controller, it is sufficient to pick a goal to the left, then pick a goal moving to the right in the valley, then pick a goal moving to the right on top of the hill. The presence of the wall means that the exact goal location is irrelevant, so the task becomes fairly trivial.

I made a very simple modification to the ARMS algorithm in order to convert it into an RL algorithm, allowing ARMS to spontaneously switch between an "exploration" mode (the default ARMS algorithm) and "exploitation" mode, where the agent selects as its goal the vertex with the highest time-average reward. The decision between exploration and exploitation is made by calculating a weighted measure of the mean-relative reward deviation across all vertices: if the deviation is high (there are some states with *much* higher reward than average), then the agent is more likely to choose to "exploit" the segraph, rather than exploring it. Learning still occurs during exploitation, as ARMS is otherwise unchanged, so experience about paths *to* the reward is still being collected. Basically, I converted ARMS into an RL algorithm by simply adding a reward-weighted *bias* to its exploration goals.

To more strongly test ARMS, I developed much more difficult variants of the MountainCar environment. Instead of the usual +100 reward being granted for merely crossing the goal position, I converted the goal-*position threshold* to a goal-*radius* in the (position, velocity) statespace, so that the reward is only granted to the agent when it is at the top of the hill *and* is also moving very slowly. I also extended the environment past the first hill, so that it is possible to overshoot the first hill, meaning the agent cannot simply slam into the wall to achieve the goal. I developed three levels of difficulty (see top row of Figure 11.2): the first level, MountainCar-1, is as I just described, a valley, then the hill with the reward, then another valley. The second level, MountainCar-2, has three hills with a reward in the middle (only when moving slowly), while the third level, MountainCar-3, has five hills. I did a small hyperparameter search on MountainCar-1 for ARMS, which was used for all environments, whereas for PPO+ I was obliged to do an extensive hyperparameter search for each individual environment. Median total per-episode reward (first and third quartiles shown as shaded region) over 200k time-steps of training is shown for both ARMS (orange) and PPO+ (blue) in the middle row of Figure 11.2.

To confirm that PPO+ was actually learning the more difficult tasks (since for MountainCar-3 reward never goes above 0) I also measured the reward-rate, plotted on the bottom row of Figure 11.2, which shows that PPO+ is actually gradually improving at the task and reaching the reward.

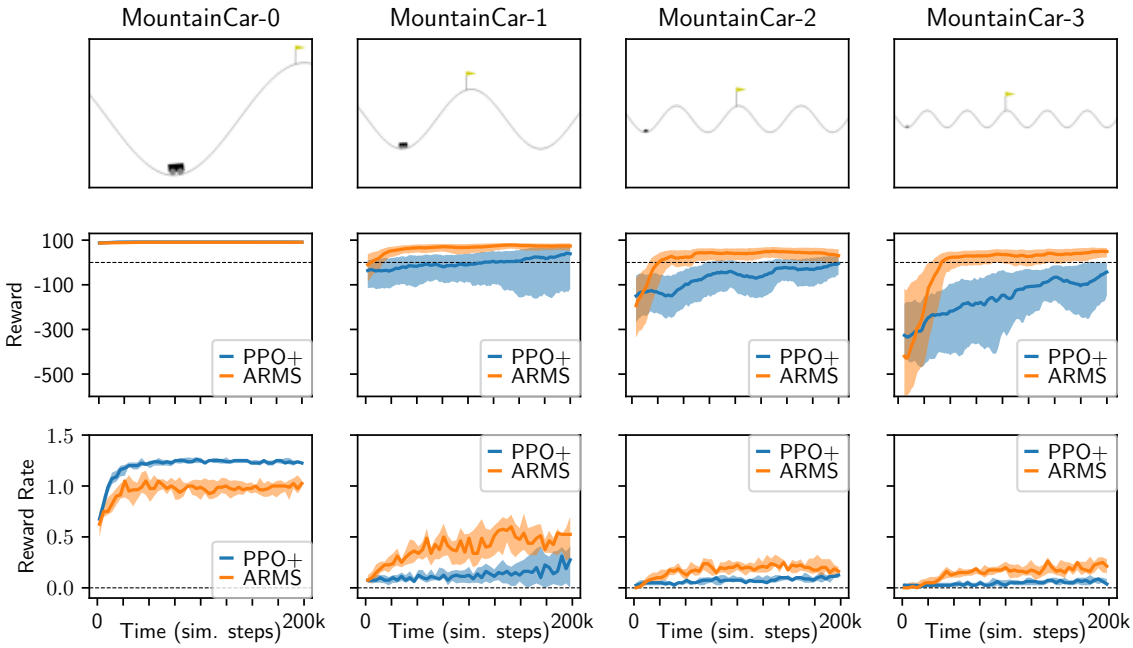


Figure 11.2: ARMS (orange) vs PPO+ (blue) on traditional ContinuousMountainCar (MountainCar-0), and variants with larger environments and stricter reward conditions, MountainCar-1, MountainCar-2, and MountainCar-3. In MountainCar-0, both ARMS and PPO+ instantly solve the problem: PPO+ is actually slighty faster, and achieves a slightly higher performances. For the harder variants, ARMS quickly edges out PPO+. In MountainCar-1, ARMS immediately performs better, and while PPO+ slowly catches up, it never quites matches ARMS within the 200k timesteps of the simulation. The difference is even more dramatic in MountainCar-2 and MountainCar-3, where ARMS rapidly achieves a large positive total reward, indicating that it reliably reaches the target, while PPO+ struggles or fails to even get a net-positive reward. PPO+ is in fact solving the problem in these cases, as can be seen by its slowly increasing reward rate in the bottom row of plots.

Chapter 12

Discussion

The core message of this thesis is that open-ended adaptation should be thought of as resulting from *search* through the space of all possible behaviors. This otherwise implausible sounding feat can be accomplished by constructing a *countable* set of behaviors which is *dense* in the set of *all possible behaviors*. Searching this set, by trying one behavior after another, guarantees that a behavior that solves the current problem will eventually be found, ensuring that successful adaptation occurs. The construction of this set follows a fairly simple and cognitively intuitive rule, summarized by the ancient advice to "leave no stone unturned". My addendum to this wisdom is that given the infinite number of "stones", one has to be sure to keep "unturning" both the larger and more distant ones, as well as the smaller and closely packed ones that lie beneath them, never pursuing one to the exclusion of the other.

My work has proceeded along three parallel avenues that have mutually informed and constrained each other, corresponding to David Marr's three levels of analysis. The theory of behavioral search, the formal elucidation of which makes up the bulk of this thesis, corresponds to the level of *computation*. The Adaptive Real-time Metasearch over Segraphs (ARMS) corresponds to the level of *algorithm*. And finally, the neural instantiation of segraphs using Hebbian learning and Harmonic Relational Key memory corresponds to the level of *implementation*. My results on each level are illustrative of possibilities, which I am hopeful can open up a new vista of research. First, I have shown that it is actually possible to meaningfully *try anything* in an intelligent, history-dependent, and ordered way. Second, I have shown that an algorithm based on this principle can solve challenging problems. And third, I have shown that artificial neurons are able to perform the operations needed for behavioral search, the implementation of which by no accident resembles a real and ancient neuroanatomical region used for a conceptually similar task.

In the rest of this chapter I will discuss several important questions raised by this work, and outline some promising directions for future research that could address them. I will highlight several application areas that could especially benefit from the ideas I've presented, which could themselves serve as sources of practical problems whose solutions would help mature behavioral search theory. Finally, I will consider some broader implications of this thesis and how they might influence the way we think about what the brain is doing.

12.1 Questions... and Answers?

Field Shape and Sampling

The segraph, the data-structure used throughout this thesis to organize the enumeration of behaviors, as its name suggests, is meant to segment state-space into chunks about which the agent can generalize. I will say a bit about why the segraph in this paper segments as it does, but intuitively, think about what might happen if you weren't quite sure what walls did, but you knew they were different from doors, whose purpose you were also unsure of. In an ideal world, after you had run into a single wall, you would make a provisional generalization about not just a small region near where you ate plaster, but would instead create a "field" that covers the entire region, the edge between the empty space of the room and that wall having a single failure associated with it, from which you might guess that all such "through-the-wall" attempts might fail, encouraging you (at least initially) to try the door.

Each segraph vertex has a corresponding hyperball "field", a segment of state-space that it represents, supporting generalization about regions of behavior-space via associated segraph edges. Hyperball fields are nice because they are easy to reason about, visualize, implement, and stochastically sample from, and naturally fall out from the simple description of the "robustness" of a behavior that I've used. They do, however, have problems. From a purely mathematical perspective, hyperballs have undesirable covering properties compared to say, hypercubes. To begin with, it is easy to see in two or three dimensions that while hypercubes can tile a space with no overlap, hyperspheres will need to overlap, leading to some *representational redundancy.* As detailed in [147], when comparing hypercubes and hyperspheres of the same diameter in higher and higher dimensions, the hypersphere "recedes" from the corners of the hypercube, necessitating more and more overlap of hyperspheres to cover a given region in higher and higher dimensions. This is a problem for hyperspheres *aside* from the curse of dimensionality, which affects any kind of scheme where we add fields of a fixed size in order to cover a given region, as occurs with both the extension and refinement mutations used in this work.

As it so happens, the basic neural mechanism that I use to endow vertices with their fields, the HaRK memory system, provides sufficient flexibility to realize relatively arbitrary-shaped fields. The fundamental equations of HaRK:

$$\begin{aligned} \boldsymbol{x}_{\boldsymbol{\delta}} &= f_{\text{HaRK}}(\boldsymbol{x}, \boldsymbol{\delta}) = \boldsymbol{x} \odot e^{2\pi i \boldsymbol{\Gamma} \boldsymbol{\delta}} \\ c(\boldsymbol{\delta}) &= \mathcal{F}_{\boldsymbol{\gamma}}^{-1}[\boldsymbol{\mu}(\boldsymbol{\gamma})\boldsymbol{\eta}(\boldsymbol{\gamma})p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})](\boldsymbol{\gamma}) \\ [(\boldsymbol{\mu} \odot \boldsymbol{x}) \triangleright \boldsymbol{y}](\boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}}) &= c(\boldsymbol{\delta}) \cdot \boldsymbol{y} \end{aligned}$$

tell us that the shape of the retrieval response $c(\boldsymbol{\delta})$ of a value $\boldsymbol{y}$ associated with a point $\boldsymbol{x}$ in an abstract space is given by the Fourier transform of the product of the distribution of frequencies in the HaRK memory function $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$ and the modulation vectors as functions of frequency, $\boldsymbol{\mu}(\boldsymbol{\gamma})$ and $\boldsymbol{\eta}(\boldsymbol{\gamma})$. In other words, we can specify $c(\boldsymbol{\delta})$, and then determine the combination of $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$, $\boldsymbol{\mu}(\boldsymbol{\gamma})$, and $\boldsymbol{\eta}(\boldsymbol{\gamma})$ that give us that shape, keeping fixed and free which parts we like.

This at least implies that we can choose the shape of the fields, which could open

up the possibility of, say, partitioning an existing field into an arbitrary number of sub-fields that exactly partition the original field, with minimal overlap, *in an arbitrary-dimensional space*. This could allow for explicit control over the curse of dimensionality. For example, rather than the number of "sub-fields" created by refinement following a law like n^d, it could instead be fixed at, say, 2, or 3, regardless of dimension. There are three major challenges to this approach, which will have to be addressed by future research. First, what should actually control the shape of these fields? It seems natural, on reflection, that they should be controlled by sensory data, but how exactly this should be done I cannot yet say. Second, what considerations should come into play when shaping these fields over longer time frames, in other words, what factors should control the *learning* of these field shapes? And third, how can fields of arbitrary shape be *sampled* from? This is a question which techniques such as normalizing flows [148] may be able to address.

Notably, place cells in the hippocampus, a strong inspiration for the neural implementation of segraphs, often have place-fields that are not radially symmetric, often elongated along hallways of mazes, and often stopping at distinctive physical and even *visual* borders. Studying how place fields gain their shape could help guide future theoretical work in this direction.

Aleatoric and Epistemic Uncertainty

Aleatoric uncertainty is "true" uncertainty: in effect, irreducible randomness. *Epistemic* uncertainty is uncertainty due to a lack of *knowledge*, which in principle can be reduced through learning. In my work I have assumed a complete absence of aleatoric uncertainty, noise, in either physical dynamics, actuation, or sensation, because my thesis is mostly concerned with epistemic uncertainty. In that context, for example, if you randomly sample several different actions from a region of behavior-space and find that some work and some don't, you can confidently conclude that there is true heterogeneity within that region with respect to the probability of success for different behaviors. This justifies, or at least motivates, the decision to *refine* such a region, because there is a reasonable expectation that this underlying heterogeneity of the region could be captured and exploited.

However, if we allow for *aleatoric* uncertainty, as we certainly should, the picture changes: we have to decide if the heterogeneity of outcomes in our finite sample of behaviors inside a region of behavior-space (an edge of the segraph) is due to an underlying heterogeneity of behavioral efficacy in that region, or due to noise. If it is due to noise, then it will have been a waste to have refined the region, as each sub-region will behave exactly as unpredictably as the parent region, leaving us in essentially the same spot. This would lead to a sort of infinite regress, where an essentially uniformly stochastic region of behavior-space would become needlessly partitioned into increasingly fine-grained "categories" of behaviors, each random failure and success being taken as evidence that just one more refinement, just one more distinction, just one more "wrinkle" in behavior will finally yield reliable results. You *could* argue that this is what occurs when certain kinds of superstition develop, for example among sports players with elaborate pre-game rituals.

There are multiple possible answers to this problem: for example, if refining a

behavior-region produces outcomes individually no-more reliable than the parent region, perhaps it would make sense to allow a region to become *unrefined*. This would make refinement less of a permanent ontological commitment and more of a temporary probe, which could be solidified by differential success or dissolved by mediocrity. Of course, this leaves open the possibility that a level of exploitable heterogeneity exists *below* the level of this "probe", so this kind of "temporary" refinement would have to be handled *very carefully* in order to achieve hyperadaptability. One possibility is that each region of behavior space that had been "un-refined" could retain a memory of *how* deep the last probe went, so that repeated failures in other parts of behavior-space could trigger a "multi-level" refinement one-level *deeper* than had previously been tried. This would amount to "re-exploring" the behavior region, which might not be so great a loss if previous attempts had been spotty.

An alternative solution would be to actually repeat some behavior exactly multiple times in order to directly test for aleatoric uncertainty. Though this would require an unrealistic level of repeatability for a physical system, some approximation thereof could nevertheless work.

Different Mutations

In this thesis I discussed two specific "mutation" operations on a segraph that were of significant conceptual and technical importance, extension and refinement, with less emphasis placed on a still extremely important mutation, "linking". These are three very specific operations, implemented in very specific ways, based on a host of pragmatic, intuitive, and theoretical considerations. However, different versions of these operations, including wholly different operations, will need to be considered and investigated in the future in order to develop a more complete version of the theory of behavioral search.

Let me start by considering a not entirely hypothetical "delinking" operation, which completely *cuts* particularly useless edges from a segraph, the intuition being that this could help to completely preclude certain paths when, for whatever reason, the affordance threshold reached 0, thus allowing "all" paths. However, the issue with delinking is that it erases all trace of the information that initially triggered delinking in the first place: the edge is completely unreliable, no path across it has ever succeeded. In deleting the edge, this fact is removed, and leaves open the possibility of a "relinking" occurring, creating the possibility for an endless pathological loop. Having considered this, I ended up just keeping terrible edges and letting them eventually "speak for themselves", as it were, keeping track of *exactly* how hopeless they were.

We could also imagine trying to be rather more clever about the way that refinement or extension works. In fact, the experiments I performed in mazes probed the possibility of performing extension in a more intelligent way, with clear success in two dimensions and mixed success in three and four dimensions. I spent a *considerable* amount of time trying different specific schemes for performing, in particular, refinement, though generally in a way that was triggered immediately by failure, rather than eventually by an accumulation of a mix of successes and failures. Though hardly a rigorous study, my observation was that any clever rule I could come up with for exactly controlling the placement of sub-fields during refinement, when run for long enough,

would eventually encounter a pathological edge-case that would create catastrophic over-refinement, meaning the agent would endlessly refine and refine at a particular point, breaking the requirement that mutation operations be *balanced*. It is, of course, possible that other mechanisms could be cooked up to counteract this tendency, but my instinct after many, *many* failed attempts was that the only truly robust policy was a policy that could not be "gamed" by some edge-case: true randomness, playing sub-fields by sampling center-points of sub-fields.

Another possible mutation would be to actually *modify* an existing field, either by changing its location, altering its shape, or changing its size. In fact, an early version of the algorithm did exactly these things! These mutations ultimately failed, and I would like to say a little bit about why I think they did. The premise of refining an edge because it has a mixture of failures and successes really relies on the fact that the statistics collected and stored about an edge refer to a stable "object", a specific and well-defined region of behavior space. However, when the shape of the region can change, it immediately invalidates any statistics collected. Even if there are good reasons to move or change the shape of a field in some clever way, the problem of the validity of collected statistics about each edge must somehow be addressed.

Noise and State Estimation

Aside from aleatoric uncertainty about the intrinsic reliability of a behavior, and epistemic uncertainty about the reliability of different regions of behavior space, there can also be aleatoric and epistemic uncertainty around the actual *state* occupied by an agent, which can be caused both by noise and by partial observability of the state-space. Noise and partial observability are both regular assumptions in machine learning, and by far are the most serious assumptions that I have had to make in my work. Thankfully, these are both extremely well-studied problems for control and navigation, so in future research we will have many choices from which to select a suitable solution.

One method I would like to highlight is the method of delay coordinate embedding [47]. For a given, potentially 1-dimensional, observation $\boldsymbol{x}(t)$ of some otherwise hidden system $\boldsymbol{s}(t)$, we augment this observation with delayed observations, $\boldsymbol{x}(t-\tau_1)$, $\boldsymbol{x}(t-\tau_2)$, $\boldsymbol{x}(t-\tau_3)$, etc., yielding a "delay coordinate" $\boldsymbol{X}(t) = (\boldsymbol{x}(t), \boldsymbol{x}(t-\tau_1), \boldsymbol{x}(t-\tau_2), ... \boldsymbol{x}(t-\tau_k))$. With sufficiently many delays, and under some mild assumptions about those delays $\tau_1, \tau_2, ...\tau_k$, the trajectory of $\boldsymbol{X}(t)$ will actually be *diffeomorphic* to the trajectory of $\boldsymbol{s}(t)$, without requiring elaborate methods of estimation or learning. This is a fact whose significance seems to be only slowly trickling into the various related fields of machine intelligence, though it is beginning to make an entrance via control theory [149, 150], and connections are even being made to the transformer architectures [151].

Now, one requirement for *perfect* reconstruction (up to diffeomorphism) is that the observation be *noiseless*. This is obviously a requirement we would like to relax, and indeed a tremendous amount of work exists developing techniques for finding good delay-coordinate embeddings even in the presence of noise [152–155]. What will be difficult is applying these methods *online*, so that state-estimation can occur in parallel with something like the ARMS algorithm. This will be tricky because an implementation like the ARMS algorithm assumes a static state-space representation, but if that representation is itself dynamic, it suggests that the meaning of the statistics collected

over edges will need to be rethought entirely. If the underlying state-space itself is shifting, it may be the case that there needs to be some way for the segraph to shift with it. A possible route is to take inspiration from place cells, whose firing field is a function both of metric-encoding grid cells, *and* information about objects and landmarks from the visual cortex. These might act as "anchors" for the segraph as the underlying state-space reconfigures itself.

Factorized Representations

While representing the state of a system in terms of a vector is extremely common practice, it has its limitations. In particular, vector-spaces are good models for systems with a fixed number of independent variables, which nonetheless are part of a singular "thing", such as the position of an object or properties like color, orientation, and lighting. But when variable numbers of independent objects come into play, with incommensurate properties, vector-space models become less parsimonious. Let me give an example. In my maze experiments, the state is just the agent's position in 2D space. Suppose that the state-space included an enemy that had to be avoided. This would mean the relevant state-space was 4D. This would probably be tractable. What if there were 10 enemies, for a 22-dimensional space. Maybe this would still be tractable, but this hardly seems like a cognitively plausible explanation, especially if we begin to imagine even more complicated environments, like cluttered rooms or the jungle. What we instead want is a truly factorized, non-parametric representation of the state-space.

One variety of factorized representation that I would like to call attention to comes from the field of Active Automata Learning (AAL), which focuses on hypothesis-based discovery of a hidden automata by carefully choosing input sequences that can falsify the current hypothesis. While these algorithms operate in a discrete setting, they have achieved impressive results in learning tasks in a way that is entirely different and parallel to DRL. Related to factorized representations, there have been proposals recently for using "automata cascades", complex automata that are built of many smaller, interconnected automata. This not only seems desirable from a cognitive and representational perspective, but also should theoretically allow for dramatically faster learning. While importing the concept of automata cascades directly into behavioral search will probably be non-trivial, as even learning via automata cascades is a new concept, appropriate groundwork could be laid by using the neural architectures developed in this thesis to instantiate continuous generalizations of more traditional AAL algorithms. AAL has a similar epistemological disposition to what I have advocated for in this thesis, so I believe that dialogue with ideas in AAL will be fruitful

Long-Term Memory

One element that has been neglected in this thesis is the subject of long-term memory. While the segraph acts, in a way, like long-term memory, there is no provision in the current incarnation of the theory of behavioral search, nor in the ARMS algorithm, for the long-term *storage* of effective segraphs and the behaviors they implicitly represent. One of the most intriguing features of the hippocampus, the brain region that partially

inspired the neural implementation in this thesis, is what is called *remapping*, which occurs when individual neurons change their spatial pattern of firing, effectively being "re-used", with varying degrees of regularity, in new environments. It is evident from recordings of place cell activity that the same combinations of place cells fire in the same ways when revisiting an environment, suggesting that in effect, a spatial map has been "recalled". While the theory of behavioral search outlined here is strong when exposed to totally new problems, there is no way currently for that knowledge to be re-used. In the brain, it is thought that the hippocampus is responsible for consolidating memories, but not, per-se, for *storing them*. Instead, that role is generally thought to be played by the neocortex. To the extent that DNNs can be thought of as a model for neocortex, perhaps they could play the role of compressing, storing, and re-initializing neural graphs.

Compressing the entire graph might be onerous. Instead, perhaps a deep neural network could somehow store a lossy, "low-resolution" version of the graph. We already have the instruments, via the HaRK memory, to encode coarse-grained structure about a graph. This would require a sort of hierarchical graph, in which somehow, coarse-grained connectivity features of the mental map are kept, but "details" are lost, left to be re-acquired through exploration. How exactly this would work I don't know, but perhaps methods from spectral graph theory could be of assistance.

Another possibility for long-term memory could entail bootstrapping a better "low-level" controller by using successful behaviors as supervised training data. In fact, I spent considerable time on this problem, and met with considerable success. However, this relies on something like a "sleep-wake" cycle, as modifying the low-level controller online would invalidate the edge statistics. For example, the controller used in the maze experiments would be unable to navigate around a corner on its own, requiring pathfinding to explicitly articulate a sequence of motions such as go past the corner, then go left. This generates a low-level motor output, which can be recorded, and if the behavior is successful, then a new controller can be trained to respond to sensory data about the corner (such as visual or depth information) to mimic the low-level motor output of the successful trajectory. However, this incurs an interesting cost: as the power of the low-level controller increases, it becomes more difficult to determine when the controller has actually failed. In my simple maze experiments, I simply triggered failure whenever the agent hit a wall, but a trained controller may never hit any walls, instead going in circles. This, in turn, may require a more general form of failure detection, such as an ETA count-down, or some kind of trained failure detection network.

Biological Realization

The neural implementation of segraphs suggests that, at least in principle, the mechanisms of the ARMS algorithm could actually be used in the brain, perhaps especially at early stages of learning where trial-and-error dominates. The neural architecture is highly inspired by the functionality of the hippocampus and entorhinal cortex, with segraph vertices playing the role of place cells, vertex-fields playing the role of place-fields, and the underlying HaRK memory system playing the role of grid cells and, more specifically, band cells. The heavy requirement for both active and passive inhibition

and disinhibition of edges in our system is interesting, because a population of cells known as *astrocytes*, common in the areas where cognitive graphs are believed to exist, actually project onto and modulate the behavior of individual synapses on both short and long timescales [156], perhaps providing the needed degree of control. A concrete biological prediction of this model is that inhibition of astrocytic activity would hurt the ability of organisms to avoid repeating obvious mistakes.

General Orders

In this thesis I have detailed a theoretical framework for enumerating behaviors in order to execute a behavioral search for solving arbitrary problems in finite time. While the theory is very clear on the necessary conditions for this enumeration to be achieved, those conditions only hold for the specific method that I chose to actually *construct* the enumeration. Future theoretical work will need to take a much more general approach: if we think about some generic countable set of behaviors that is dense in the set of all behaviors, what sorts of orders are possible on that set? Ideally, we would like to have some way to *shape* the order, perhaps by allowing prior experience solving a problem to bias the enumeration for *future* problems in a way that moves previously successful solutions "up" in the order, and previously unsuccessful solutions "down" in the order.

This will require a shift in focus, or at least an expansion of focus, from the set of all behaviors, to the set of all *orders* on countably infinite sets of all behaviors. The set of all orders or enumerations of a countably infinite set of behaviors is not, itself, countably infinite, but rather uncountably infinite. This strikes me as highly suggestive: an uncountably infinite number of orders makes the systematic manipulation of these orders that I believe would be necessary for long-term learning seem infeasible. Perhaps there are ways to construct parameterized, *countably* infinite orders, so that the set of all orders can itself be ordered and traversed in some logical way? This could enable a unique form of meta-learning that would be interesting to study in its own right, and could perhaps allow for the creation of dramatically more adaptable and intelligent systems.

Expanding only slightly on the work in this dissertation, the ARMS algorithm was designed to implicitly encourage an order that prioritized behavioral generality, behavioral feasibility, and behavioral simplicity by first dynamically thresholding the set of segraph edges by their estimated affordance, equal to the product of their measure (higher measure meaning greater generality) and their estimated reliability (higher reliability indicating the entire path is more feasible), and always selecting from among the shortest paths to encourage simpler behaviors. These were all essentially engineering choices, rather than principled mathematical ones, though they seem to have been effective, at least in the context of the experiments performed. These three measures, generality, feasibility, and simplicity, seem like excellent and perhaps even universal criteria by which to order a search through behavior-space, but finding more generic means of expressing those criteria and analyzing the consequences of different choices will constitute an important part of future work.

One additional bias, which was incorporated in earlier versions of the algorithm, is to encourage the agent to prioritize exploration of easier-to-reach goals. The current algorithm simply picks the largest, least explored edge anywhere on the segraph and

tries to find a path, but this is probably not a very efficient use of *time*. In all likelihood, there is a benefit to adding a "distance" criterion to the order over behaviors, since nearby goals can be reached more quickly, potentially allowing for more information to be gained. Alternatively, rather than selecting just *goals* based on their epistemic value, it would *also* be possible to choose the *paths* to those goals based on their epistemic gain. This would have one major advantage over the distance-criterion, because the distance-criterion is dependent on the threshold selected for edges, creating the potential for an odd and possibly undesirable feedback loop. In addition, while the epistemic criterion for paths might cause the agent to make many more poor pathing choices early on, it may accelerate learning and allow the agent to make much better choices more quickly.

A major focus of work in this area should be finding ways to use the many powerful and useful tools that *already* exist in deep learning and reinforcement learning to represent, either implicitly or explicitly, orders over behaviors. While I don't want to prematurely rule-out explicit representations, it seems like an unpromising direction. The method employed in my work, an implicit representation, has at least as an approximate exemplar the ARMS algorithm. It would be reasonable to ask if there are ways to modify an RL algorithm to operate the same machinery as ARMS does (mutation, pathfinding, arbitrating between goals, etc.). One possibility with which I had some success briefly experimenting with is using a neural network (not necessarily a deep one) to predict from sensory data the likely reliability of an edge after being created, using this information to set the prior for estimated reliability.

Regarding DL and RL methods, it would also be possible in principle to apply the concept of searching behavior space via enumeration of a countably infinite dense set to deep neural networks that represent policies, i.e., controllers. [157] showed that it is possible to use genetic algorithms to construct effective neural network controllers with variable numbers of elements. Building on this concept, it might be possible to construct a countably infinite set of weighted directed graphs (the architecture of the neural network) which is dense in the set of all such graphs, allowing for systematic search through the space of neural networks. In principle, this could be used as a method to find the low-level controller, or perhaps even the neural network that runs the search. The question of constructing general, experience-dependent, dynamic orders over countably infinite dense sets of objects seems to me an extremely rich area for future research, one that allows for a great deal of creativity while still being rigorous.

12.2 Potential Applications

Navigation

The entire ARMS algorithm was developed in the context of navigation, on the supposition that navigation can, at least abstractly, be taken as general-purpose problem solving applied to "location in space". Having solved navigation problems under the assumptions of full state-observability and noiselessness, a natural next direction is to attack real navigation problems, which will require incorporation of SLAM methods. While using already-developed SLAM solutions is an excellent starting point, I think that incorporating the SLAM problem into the more general state-space reconstruction and estimation problem will be more fruitful. This could especially involve modifica-

tions and generalizations of the underlying neural architecture, which has some latent properties that could enable novel methods of localization.

For example, a very early version of my project, which had not yet incorporated graphs into navigation (though I *was* playing around with building neural graphs with Hebbian learning), simply used a variable-resolution storage of a "visitation" trace, which when resolution was low (coarse-grained) allowed an agent to detect from a great distance far away locations of high novelty. While following that gradient, the agent could increase the resolution. This dynamic resolution concept might enable fast, "good-enough" localization. Another early experiment I did involved directly storing sensory data in a position-orientation prototype of HaRK. Combining this idea with a variable-resolution memory system could enable fast and robust "good-enough" localization, which could allow for the use of visual landmarks for navigation. In any case, real physical navigation entails a host of challenges which if met, could help to substantially refine the current theory, algorithm, and neural implementation.

Locomotion

Navigation of one's location in space is not exactly the same as locomotion, which we could think of instead as "navigation through limb-position space", with the caveat of a sort of additional dimension of "direction moved forward over T-interval of time. Moving towards solving locomotion challenges will require significant investigation of the issues the current algorithm faces in higher dimensions, which hopefully can be addressed by the points mentioned earlier. If ARMS, or a successor to it, can be made to work effectively in 3-15 dimensional spaces, then it becomes possible that "joint-configuration $\times$ distance-traveled" space can be meaningfully navigated. The benefit of such a system would be that, naturally, it could potentially very quickly find a "gait" that is effective, but even more significantly, as evidenced by the robustness of the ARMS algorithm to major reconfigurations of the state-space, it would also enable very rapid *adaptation* of gait to new conditions, such as a faulty servo, additional weight, or perhaps even impeded ground such as terrain full of obstacles.

What is interesting about locomotion from the theoretical perspective of behavioral search is that it is a fundamentally oscillatory behavior, and in general, we don't actually care about the specific configuration of the body. First, this means that the goal-state is not actually a specific bounded region, like a hyperball, but is instead a subspace, or maybe even, a hyperplane, the goal being to cross the hyperplane. This may even lead into more of an RL-framing for behavioral search, where the agent is trying to reach points in configuration-space that correspond to as great a rate of speed as possible. Additionally, the nature of locomotion is usually the maintenance of motion, involving cyclic action. On one level, we might think of this as not being so different from the situation where ARMS was used to solve the Continuous MountainCar problem, where the agent is reset to an initial state after having reached the goal. Likewise, we could imagine that every time the agent reaches the goal of having traveled, for instance, 10cm in a second, that it resets to having traveled 0cm and needs to once again try to reach the target of traveling another 10cm over the next second.

On the other hand, this seems like a bit of a "hack", as it entails careful engineering of the configuration space, and its dynamics. This is a topic that will need considerable

research and probably reconsideration of fundamental assumptions used in this thesis.

Grasping and Manipulation

Another interesting and rich source of problems from the perspective of behavioral search is reaching, grasping, and manipulation. On the face of it, reaching, grasping, and manipulation might be a little bit like navigation, with the caveat that some dimension of state-space needs to correspond to "successful grasp on object", a hack rather similar in character to the "distance traveled over 10 cm", which perhaps could be handled in ways similar to those proposed for Locomotion, adopting more techniques from reinforcement learning. Perhaps the aspect of the current theory that grasping and manipulation push against the most is the representation of the underlying state-space. In cluttered environments with many objects, it seems infeasible to represent all properties of all objects, which anyway are of a variable number, in a single vector space. How exactly this could be done has already been addressed, but grasping and manipulation represent a problem area where solutions on this front would be especially beneficial, and would have especially tangible benefits.

One reason that I think grasping and manipulation would especially benefit from the application of the ideas in this thesis is that in practice, it involves a considerable amount of online search. When you aren't paying attention to the object you reach for, you often feel around for it. After having found it, you may put your hand on it in one way, subconsciously decide that the grasp is insufficient, and try another until a firm but appropriately gentle grip is achieved. What is often phrased as an extremely difficult and ill-posed *inference* problem may instead be posed as a fairly trivial *search* problem. Many natural behaviors of animals that seem like they would rely on an exquisitely detailed internal model for *inference*, may only require a very crude internal model and a small amount of fiddling, or online-search, in order to find the right way to interact with a specific object in a given context.

Observational Learning

More of a general application, but I think that since the behavioral search framework explicitly works with behaviors, it may be uniquely well-suited to leveraging observations of others' behavior. Imagine that we are trying to teach a robot some task, and we want it to learn from watching us perform the same task. Many techniques for taking an observation of task and transforming it into the reference frame of an artificial agent. Once this is done, there is no guarantee that the transformed behavior will work exactly as intended. However, in the behavioral search paradigm, a single observed behavior could act as a sort of "seed", anchoring the search space by indicating to the agent that *some* behavior *near* the transformed behavior, if not the transformed behavior itself, will surely succeed.

12.3 General Implications

Optimal Control

While I have argued that optimal control is out-of-reach in a great range of realistic and important scenarios, thus necessitating the search paradigm outlined in this thesis, it is nevertheless desirable, especially for those problems that recur, to be able to move towards optimality by some metric over repeated exposure to the problem. Actually, we can in principle still use search to look for optimal behaviors, as even classic gradient-based optimization is technically just a kind of search guided by local heuristics. In principle we could use these local heuristics alongside the more "discrete" form of search that I've used to formulate hyperadaptability, but also, we can just use search directly to keep looking for better and better solutions (by whichever criteria we've selected), thus achieving the same aim as classical forms of optimization. There are many distinct forms of discrete optimization, and how or whether these methods can be applied to behavior search, and if basic elements of behavioral search would need to be modified to accommodate, remains to be seen.

Parallel Bases for Behavior

In this work I have chosen to represent behaviors in a specific, but not unjustified, way. However, there is evidence from work by researchers such as Dagmar Sternad that behaviors may begin by being represented in one way, and through practice come to be represented in another way, which facilitates smoothness, precision, and regularity of motion. When described in this way in an artificial intelligence context, it may seem un-parsimonious to suggest that multiple relatively independent systems exist that can represent behavior... the universalizing impulse is to imagine a single unified representation. Perhaps one way to accommodate this impulse while reconciling it with empirical observations of animal behavior is to think about behavioral representation in terms of bases. In this thesis I have mathematically characterized the physical execution of a behavior as a state-space function of time. As we know, functions can be represented in different ways. For example, we can take any function and represent it in the spectral domain using a sum of complex exponentials. We can represent the same function using any of a number of different polynomial bases. None of these bases is intrinsically better than the others, each has their distinct advantages in different contexts, and it may be the case that we should be thinking about a sort of "behavior vector space" that admits different choices of basis, corresponding to different decompositions of behavior. If such can be done, it might be possible to find a behavior by searching in one basis, but then discover that it has a much simpler representation in another basis. This would add a whole new layer to the search problem, as now search might proceed in one basis, then switch to another, then switch back again, or to an entirely different basis. For example, perhaps one basis is effectively "piecewise linear functions", which might be useful for finding the rough shape of a behavior. After finding that rough shape, it may turn out that a representation in terms of polynomials is much simpler, and after even more search, it may happen that the behavior is just a straight line in some otherwise arbitrary, moving, object-centered coordinate system.

Once such a basis is found, if found to be broadly applicable or to have some useful modular structure, it could dramatically accelerate the search for useful behaviors.

Explainable AI

Most tasks, especially critical tasks, are performed by humans, who can understand, reason, and have moral responsibility for their actions. As machine learning systems proliferate into the real world and take over more and more of these roles, it is increasingly becoming a concern that we have very little way of understanding *why* our machines make the decisions they do. While it is not the focus of this thesis, the fact that behaviors have an entirely modular and mostly explicit representation may make the kind of system presented here attractive to those interested in explainable AI.

Learning and Habit Formation

Much of current deep reinforcement learning research is, indirectly, inspired by older work on habit-formation. Technically, we could think of "problem-solving" as a kind of very sophisticated habit, and I think that this idea is implicit in the attempt to create artificial general intelligence by exposing what is, essentially, a habit forming machine (DRL) to as many examples of as many problems as possible. However, this thesis has been an investigation of an alternative perspective: that if we equate "problem-solving" with "search", then perhaps *habits* are just cached *search results*. To the extent that model-free RL can handle old situations, and model-based RL is needed for new ones, this framing dissolves the dichotomy, as "model-free" behavior is just what happens when the first result of behavioral search is correct, whereas "model-based" behavior occurs when subsequent search steps are required. Some systems have relied on search as a kind of "fall-back" mechanism when learned priors fail, but a major conceptual take-away of this thesis is that search should be front-and-center in our theoretical understanding of brain function. Under this framing, when we get something right on the first try, it's because prior searches have better-organized the search space, so that solution #1 is very likely to succeed. But, if it doesn't, the search merely continues. The more we search, the better we will get at finding... and even if we don't, we can still just keep trying.

12.4 Related Work

Our work, proceeding as it does from a theoretical analysis of behavior and problem-solving, touches on a broad range of topics, and ultimately, comes to conclusions that are echoed through the literature on reinforcement learning, problem-solving, and planning. Our specific neural implementation of cognitive graphs creates an overlap of concern, both with neurologically-inspired models of hippocampal learning, and with cognitively-inspired hybrid models. And of course, many different RL algorithms have been developed to handle the issues of sparse rewards, sample efficiency, and lack of resets.

Two prominent continuous statespace planning algorithms for robotics, Rapidly-exploring Random Trees (RRT) [81] and Probabilistic Roadmaps [82] are interesting to

compare to our algorithm. Both algorithms have good performance in high-dimensional statespaces, but both also assume that the problem can be purely represented in terms of a free-space, which can be queried at-will. This setting is quite different from the one we consider, where the only way for the agent to "query" the space is to actually physically attempt an action. This actually highlights a potential problem with our formulation of ARMS: because queries cannot be performed freely, but must be *local*, there is a real cost to exploration that is performed inefficiently. It could be the case that more careful control over which edges the agent tries to use could boost performance significantly, even under the constraints of locality.

Planning (search) and learning have been combined quite fruitfully in various RL systems, notably MuZero [158], which learns latent dynamics of a problem and performs planning using Monte-Carlo tree search. While MuZero has been successfully applied to continuous domains [159], the overall algorithm family is significantly different: it uses a learned forward model with Monte Carlo tree search to simulate different futures using sampled actions, with the tree itself being thrown away and rebuilt at every time-step. In this work, we instead grow a dynamic graph that is continuously used and updated, which stores information about the possibility of state-state transitions, with latent dynamics only implicitly represented, and actual actions handled by a low-level controller.

Related more to our usage of a neural graph as a model, the Tolman-Eichenbaum Machine (TEM) [160] represents one of the most mature models of hippocampal learning. However, they focus on learning good state-representations, compositional inference, and prediction, working in problem domains defined over a static graph. Instead, we dynamically construct a discrete structure (the segraph) *over* a continuous space, using extension and refinement to ensure that any areas or details of the state-space can eventually be represented by the edges of the segraph.

In between pure reinforcement learning and more cognitive-science considerations are works such as [161], which cast the hippocampus as a reinforcement learning engine that builds successor representations to efficiently decompose tasks in a flexible way. In our work, we adopt a significantly more primitive RL formulation for ARMS, though in principle it would be easy to propagate values over the segraph-vertices to define a value-function. On the other hand, we emphasize the potential for cognitive graphs to be used for rapid adaptation and problem solving, which is not the focus of their work. As it so happens, hippocampus contains both place cells with firing fields that stabilize with experience (more suggestive of gradually consolidated successor representations), and place cells with unstable firing fields that are "freshly" recruited even in the same contexts [162], perhaps indicative of something approximated by our proposal.

12.5 Conclusion

Learning, abstraction, and generalization are all important abilities for organisms, but in the face of constant change and fundamental limitations on knowledge, there is an inescapable need for *search*. Classically, search is viewed as only possible in discrete domains, as the real continuous world we live in contains uncountably infinitely many locations, and correspondingly, uncountably infinitely many possible behaviors.

However, there is a simple way that this problem can be circumvented: provisionally generalize about the behaviors you try so that for now, you can go try something substantially different; but always leave open the possibility to reconsider your initial generalization, so you can go back and probe more carefully there in the space of possible plans. Following this rule, it is possible to construct a subset of all behaviors which is dense and countable, meaning that it can be searched, and any behavior can be arbitrarily closely approximated in the process, guaranteeing, in a mathematical sense, eventual success. This search can be performed using a familiar tool: a cognitive graph, implemented by a neural network reminiscent of the hippocampus and entorhinal cortex, regions known to be involved in similar functions. The ability to search for a solution enables *hyperadaptability*, a theoretical limit of adaptability that offers us a different perspective on what it means to be intelligent.

References

[1] A. Baranski and J. Tani, "Life, uh, finds a way: Hyperadaptability by behavioral search," *arXiv preprint arXiv:2410.01349*, 2025.

[2] C. Darwin, *On the Origin of Species: By Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: John Murray, 1859.

[3] A. Kurakin, "The self-organizing fractal theory as a universal discovery method: the phenomenon of life," *Theoretical Biology and Medical Modelling*, vol. 8, no. 1, p. 4, 2011.

[4] T. Misteli, "Self-organization in the genome," *Proceedings of the National Academy of Sciences*, vol. 106, no. 17, pp. 6885–6886, 2009.

[5] F. Pigozzi, A. Goldstein, and M. Levin, "Associative conditioning in gene regulatory network models increases integrative causal emergence," *Communications Biology*, vol. 8, no. 1, p. 1027, 2025.

[6] C. M. A. Gómez and K. Echeverri, "Salamanders: The molecular basis of tissue regeneration and its relevance to human disease," *Current topics in developmental biology*, vol. 145, pp. 235–275, 2021.

[7] F. H. Silver, J. W. Freeman, and D. DeVore, "Viscoelastic properties of human skin and processed dermis," *Skin research and technology*, vol. 7, no. 1, pp. 18–23, 2001.

[8] H. M. Frost, "Bone "mass" and the "mechanostat": a proposal," *The anatomical record*, vol. 219, no. 1, pp. 1–9, 1987.

[9] G. Goldspink, "Gene expression in muscle in response to exercise," *Journal of Muscle Research & Cell Motility*, vol. 24, no. 2, pp. 121–126, 2003.

[10] E. J. Slijper, "Biologic-anatomical investigations on the bipedal gait and upright posture in mammals, with special reference to a little goat, born without forelegs," *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen*, vol. 45, pp. 288–295, 407–415, 1942.

[11] A. M. Turing *et al.*, "On computable numbers, with an application to the entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[12] M. Peer, I. K. Brunec, N. S. Newcombe, and R. A. Epstein, "Structuring knowledge with cognitive maps and cognitive graphs," *Trends in cognitive sciences*, vol. 25, no. 1, pp. 37–54, 2021.

[13] H. Von Helmholtz, *Handbuch der physiologischen Optik*, vol. 9. L. Voss, 1867.

[14] R. L. Gregory, "Perceptions as hypotheses," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 290, no. 1038, pp. 181–197, 1980.

[15] D. C. Knill and W. Richards, *Perception as Bayesian inference*. Cambridge University Press, 1996.

[16] R. P. Rao and D. H. Ballard, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects," *Nature neuroscience*, vol. 2, no. 1, pp. 79–87, 1999.

[17] K. Friston, "The free-energy principle: a unified brain theory?," *Nature reviews neuroscience*, vol. 11, no. 2, pp. 127–138, 2010.

[18] A. Clark, "Whatever next? predictive brains, situated agents, and the future of cognitive science," *Behavioral and brain sciences*, vol. 36, no. 3, pp. 181–204, 2013.

[19] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[20] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and brain sciences*, vol. 40, p. e253, 2017.

[21] J. R. Saffran and N. Z. Kirkham, "Infant statistical learning," *Annual review of psychology*, vol. 69, no. 1, pp. 181–203, 2018.

[22] C. M. Conway, "How does the brain learn environmental structure? ten core principles for understanding the neurocognitive mechanisms of statistical learning," *Neuroscience & Biobehavioral Reviews*, vol. 112, pp. 279–299, 2020.

[23] B. J. Kagan, "Two roads diverged: Pathways toward harnessing intelligence in neural cell cultures," *Cell Biomaterials*, vol. 1, no. 8, 2025.

[24] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[25] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," *arXiv preprint arXiv:1804.06893*, 2018.

[26] J. Farebrother, M. C. Machado, and M. Bowling, "Generalization and regularization in dqn," *arXiv preprint arXiv:1810.00123*, 2018.

[27] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *International conference on machine learning*, pp. 1282–1289, PMLR, 2019.

[28] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, "Do imagenet classifiers generalize to imagenet?," in *International conference on machine learning*, pp. 5389–5400, PMLR, 2019.

[29] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–35, 2021.

[30] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 13344–13362, 2023.

[31] A. Vettoruzzo, M.-R. Bouguelia, J. Vanschoren, T. Rögnvaldsson, and K. Santosh, "Advances and challenges in meta-learning: A technical review," *IEEE transactions on pattern analysis and machine intelligence*, vol. 46, no. 7, pp. 4763–4779, 2024.

[32] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, "Towards continual reinforcement learning: A review and perspectives," *Journal of Artificial Intelligence Research*, vol. 75, pp. 1401–1476, 2022.

[33] A. Chen, A. Sharma, S. Levine, and C. Finn, "You only live once: Single-life reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14784–14797, 2022.

[34] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *Journal of Machine Learning Research*, vol. 21, no. 181, pp. 1–50, 2020.

[35] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *arXiv preprint arXiv:1808.04355*, 2018.

[36] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, *et al.*, "Model-based reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.

[37] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[38] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song, "Assessing generalization in deep reinforcement learning," *arXiv preprint arXiv:1810.12282*, 2018.

[39] Z. Zhu and H. Zhao, "A survey of deep rl and il for autonomous driving policy learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14043–14065, 2021.

[40] M. Crichton, *Jurassic Park*. New York: Alfred A. Knopf, Oct. 1990. First edition, first printing.

[41] D. Hume and D. G. C. Macnabb, eds., *A Treatise of Human Nature*. Oxford: Clarendon press, 1739.

[42] K. R. Popper, *The Logic of Scientific Discovery*. London: Routledge, 1959. First published in German as *Logik der Forschung* (1934).

[43] P. Feyerabend, *Against Method: Outline of an Anarchistic Theory of Knowledge*. Atlantic Highlands, N.J.: Humanities Press, 1974.

[44] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, pp. 35–45, 03 1960.

[45] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[46] L. Fan, F. Zhang, H. Fan, and C. Zhang, "Brief review of image denoising techniques," *Visual computing for industry, biomedicine, and art*, vol. 2, no. 1, p. 7, 2019.

[47] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick 1980* (D. Rand and L.-S. Young, eds.), (Berlin, Heidelberg), pp. 366–381, Springer Berlin Heidelberg, 1981.

[48] N. Tinbergen, "On aims and methods of ethology," *Zeitschrift für tierpsychologie*, vol. 20, no. 4, pp. 410–433, 1963.

[49] K. Immelmann, *Introduction to Ethology*. New York, NY: Plenum Press, 1 ed., 1980. Reprinted by Springer, New York, 2012. Originally published by Plenum Press, 1980.

[50] B. Skinner, "The phylogeny and ontogeny of behavior," *Behavioral and Brain Sciences*, vol. 7, no. 4, pp. 669–677, 1984.

[51] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, Cambridge, Massachusetts: The MIT Press, 2nd ed., 2018.

[52] E. C. Tolman, "Cognitive maps in rats and men," *Psychological review*, vol. 55, no. 4, p. 189, 1948.

[53] N. Chomsky, "Rules and representations," *Behavioral and brain sciences*, vol. 3, no. 1, pp. 1–15, 1980.

[54] N. Chomsky, "Aspects ofthe theory ofsyntax," *Cambridge, MA: MITPress*, no. 1977, pp. 71–132, 1965.

[55] M. H. Christiansen and N. Chater, *Creating language: Integrating evolution, acquisition, and processing*. Mit Press, 2018.

[56] A. Perfors, J. B. Tenenbaum, and T. Regier, "The learnability of abstract syntactic principles," *Cognition*, vol. 118, no. 3, pp. 306–338, 2011.

[57] K. S. Lashley, *Brain mechanisms and intelligence*. Dover Publications New York, 1963.

[58] I. Krechevsky, *The genesis of "hypotheses" in rats*. University of California Press, 1932.

[59] I. Krechevsky, ""hypotheses" versus "chance" in the pre-solution period in sensory discrimination-learning," *University of California Publications in Psychology*, vol. 6, pp. 27–44, 1932.

[60] I. Krechevsky, ""hypotheses" in rats.," *Psychological Review*, vol. 39, no. 6, p. 516, 1932.

[61] I. Krechevsky, "A study of the continuity of the problem-solving process.," *Psychological Review*, vol. 45, no. 2, p. 107, 1938.

[62] K. Duncker and I. Krechevsky, "On solution-achievement.," *Psychological Review*, vol. 46, no. 2, p. 176, 1939.

[63] M. Levine, *A cognitive theory of learning: Research on hypothesis testing*. Routledge, 1975.

[64] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[65] H. Bottemanne, "Bayesian brain theory: Computational neuroscience of belief," *Neuroscience*, vol. 566, pp. 198–204, 2025.

[66] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[67] M. György Buzsáki, *The brain from inside out*. Oxford University Press, 2019.

[68] A. B. Kaufman and J. C. Kaufman, *Animal creativity and innovation*. Academic Press, 2015.

[69] S. M. Reader and K. N. Laland, *Animal Innovation*. Oxford University Press, 09 2003.

[70] N. Wiener, *Cybernetics, or Control and Communication in the Animal and the Machine*. Cambridge, MA: MIT Press, 1948.

[71] W. R. Ashby, *Design for a Brain: The Origin of Adaptive Behavior*. London: Chapman & Hall, first ed., 1952.

[72] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[73] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Information Fusion*, vol. 85, pp. 1–22, 2022.

[74] L. Zhang, K. Tang, and X. Yao, "Explicit planning for efficient exploration in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[75] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[76] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *arXiv preprint arXiv:1611.03530*, 2016.

[77] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[78] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," *Trends in cognitive sciences*, vol. 23, no. 5, pp. 408–422, 2019.

[79] S. S. Du, S. M. Kakade, R. Wang, and L. F. Yang, "Is a good representation sufficient for sample efficient reinforcement learning?," *arXiv preprint arXiv:1910.03016*, 2019.

[80] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[81] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[82] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[83] J. H. Holand, "Adaptation in natural and artificial systems," *Ann Arbor: The University of Michigan Press*, vol. 32, 1975.

[84] J. Lehman, K. O. Stanley, *et al.*, "Exploiting open-endedness to solve problems through the search for novelty.," in *ALIFE*, pp. 329–336, 2008.

[85] G. Cantor, "Über eine eigenschaft des inbegriffes aller reellen algebraischen zahlen," *Journal für die reine und angewandte Mathematik*, vol. 77, pp. 258–262, 1874.

[86] G. Cantor, "Ein beitrag zur mannigfaltigkeitslehre," *Journal für die reine und angewandte Mathematik (Crelles Journal)*, vol. 1878, no. 84, pp. 242–258, 1878.

[87] M. Rosenberg, T. Zhang, P. Perona, and M. Meister, "Mice in a labyrinth show rapid learning, sudden insight, and efficient exploration," *Elife*, vol. 10, p. e66175, 2021.

[88] S. Moore and K. V. Kuchibhotla, "Slow or sudden: Re-interpreting the learning curve for modern systems neuroscience," *IBRO Neuroscience Reports*, vol. 13, pp. 9–14, 2022.

[89] A. Kumar, J. Clune, J. Lehman, and K. O. Stanley, "Questioning representational optimism in deep learning: The fractured entangled representation hypothesis," *arXiv preprint arXiv:2505.11581*, 2025.

[90] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*, vol. 24, pp. 109–165, Elsevier, 1989.

[91] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[92] J. Serra, D. Suris, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task," in *International conference on machine learning*, pp. 4548–4557, PMLR, 2018.

[93] E. L. Aleixo, J. G. Colonna, M. Cristo, and E. Fernandes, "Catastrophic forgetting in deep learning: A comprehensive taxonomy," *arXiv preprint arXiv:2312.10549*, 2023.

[94] W. James, *The Principles of Psychology*. 1890. Project Gutenberg eBook, Release Date: March 2018 [EBook #57628].

[95] H. A. Simon, "Rational choice and the structure of the environment.," *Psychological review*, vol. 63, no. 2, p. 129, 1956.

[96] Y. Du, S. Li, Y. Sharma, J. Tenenbaum, and I. Mordatch, "Unsupervised learning of compositional energy concepts," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15608–15620, 2021.

[97] B. Fong and D. I. Spivak, *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge: Cambridge University Press, 2019.

[98] P. Pagin and D. Westerståhl, "Compositionality i: Definitions and variants," *Philosophy Compass*, vol. 5, no. 3, pp. 250–264, 2010.

[99] T. G. Henry, *Generalization of deep neural networks to unseen attribute combinations*. PhD thesis, Massachusetts Institute of Technology, 2020.

[100] J. Tani, "Model-based learning for mobile robot navigation from the dynamical systems perspective," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 3, pp. 421–436, 1996.

[101] E. Bizzi, F. A. Mussa-Ivaldi, and S. Giszter, "Computations underlying the execution of movement: a biological perspective," *Science*, vol. 253, no. 5017, pp. 287–291, 1991.

[102] K. A. Thoroughman and R. Shadmehr, "Learning of action through adaptive combination of motor primitives," *Nature*, vol. 407, no. 6805, pp. 742–747, 2000.

[103] Monty Python, "The ministry of silly walks," 1970. Television sketch, in *Monty Python's Flying Circus*, Series 2, Episode 1. BBC. First broadcast September 15, 1970.

[104] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[105] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural computation*, vol. 1, no. 3, pp. 372–381, 1989.

[106] A. Koul, S. Greydanus, and A. Fern, "Learning finite state representations of recurrent policy networks," *arXiv preprint arXiv:1811.12530*, 2018.

[107] M. H. Danesh, A. Koul, A. Fern, and S. Khorram, "Re-understanding finite-state representations of recurrent policy networks," in *International conference on machine learning*, pp. 2388–2397, PMLR, 2021.

[108] M. I. Rabinovich, P. Varona, I. Tristan, and V. S. Afraimovich, "Chunking dynamics: heteroclinics in mind," *Frontiers in computational neuroscience*, vol. 8, p. 22, 2014.

[109] P. Chossat and M. Krupa, "Heteroclinic cycles in hopfield networks," *Journal of Nonlinear Science*, vol. 26, no. 2, pp. 315–344, 2016.

[110] P. Ashwin and C. Postlethwaite, "On designing heteroclinic networks from graphs," *Physica D: Nonlinear Phenomena*, vol. 265, pp. 26–39, 2013.

[111] P. Smolensky, "Tensor product variable binding and the representation of symbolic structures in connectionist systems," *Artificial intelligence*, vol. 46, no. 1-2, pp. 159–216, 1990.

[112] K. Schlegel, P. Neubert, and P. Protzel, "A comparison of vector symbolic architectures," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4523–4555, 2022.

[113] T. A. Plate, *Distributed representations and nested compositional structure*. University of Toronto, Department of Computer Science, 1994.

[114] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[115] J. A. Anderson, "A simple neural network generating an interactive memory," *Mathematical biosciences*, vol. 14, no. 3-4, pp. 197–220, 1972.

[116] I. Schlag, T. Munkhdalai, and J. Schmidhuber, "Learning associative inference using fast weight memory," *arXiv preprint arXiv:2011.07831*, 2020.

[117] M. A. E. I. LIEBLICH, "Motivationnal learning of spatial behavior," *Systems neuroscience*, pp. 221–239, 1977.

[118] K. L. Stachenfeld, M. M. Botvinick, and S. J. Gershman, "The hippocampus as a predictive map," *Nature neuroscience*, vol. 20, no. 11, pp. 1643–1653, 2017.

[119] T. E. Behrens, T. H. Muller, J. C. Whittington, S. Mark, A. B. Baram, K. L. Stachenfeld, and Z. Kurth-Nelson, "What is a cognitive map? organizing knowledge for flexible behavior," *Neuron*, vol. 100, no. 2, pp. 490–509, 2018.

[120] J. O'Keefe, "Place units in the hippocampus of the freely moving rat," *Experimental neurology*, vol. 51, no. 1, pp. 78–109, 1976.

[121] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser, "Microstructure of a spatial map in the entorhinal cortex," *Nature*, vol. 436, no. 7052, pp. 801–806, 2005.

[122] M. A. Wilson and B. L. McNaughton, "Reactivation of hippocampal ensemble memories during sleep," *Science*, vol. 265, no. 5172, pp. 676–679, 1994.

[123] W. B. Scoville and B. Milner, "Loss of recent memory after bilateral hippocampal lesions," *Journal of neurology, neurosurgery, and psychiatry*, vol. 20, no. 1, p. 11, 1957.

[124] J. C. Jackson, A. Johnson, and A. D. Redish, "Hippocampal sharp waves and reactivation during awake states depend on repeated sequential experience," *Journal of Neuroscience*, vol. 26, no. 48, pp. 12415–12426, 2006.

[125] D. J. Foster and M. A. Wilson, "Reverse replay of behavioural sequences in hippocampal place cells during the awake state," *Nature*, vol. 440, no. 7084, pp. 680–683, 2006.

[126] A. S. Gupta, M. A. Van Der Meer, D. S. Touretzky, and A. D. Redish, "Hippocampal replay is not a simple function of experience," *Neuron*, vol. 65, no. 5, pp. 695–705, 2010.

[127] G. Dragoi and S. Tonegawa, "Preplay of future place cell sequences by hippocampal cellular assemblies," *Nature*, vol. 469, no. 7330, pp. 397–401, 2011.

[128] A. D. Redish, "Vicarious trial and error," *Nature Reviews Neuroscience*, vol. 17, no. 3, pp. 147–159, 2016.

[129] S. Zhang and D. Manahan-Vaughan, "Spatial olfactory learning contributes to place field formation in the hippocampus," *Cerebral cortex*, vol. 25, no. 2, pp. 423–432, 2015.

[130] M. Schafer and D. Schiller, "Navigating social space," *Neuron*, vol. 100, no. 2, pp. 476–489, 2018.

[131] X. Long and S.-J. Zhang, "A novel somatosensory spatial navigation system outside the hippocampal formation," *Cell research*, vol. 31, no. 6, pp. 649–663, 2021.

[132] M. Nau, T. Navarro Schröder, J. L. Bellmund, and C. F. Doeller, "Hexadirectional coding of visual space in human entorhinal cortex," *Nature neuroscience*, vol. 21, no. 2, pp. 188–190, 2018.

[133] T. Flossmann and N. L. Rochefort, "Spatial navigation signals in rodent visual cortex," *Current opinion in neurobiology*, vol. 67, pp. 163–173, 2021.

[134] A. O. Constantinescu, J. X. O'Reilly, and T. E. Behrens, "Organizing conceptual knowledge in humans with a gridlike code," *Science*, vol. 352, no. 6292, pp. 1464–1468, 2016.

[135] R. C. Wilson, Y. K. Takahashi, G. Schoenbaum, and Y. Niv, "Orbitofrontal cortex as a cognitive map of task space," *Neuron*, vol. 81, no. 2, pp. 267–279, 2014.

[136] N. W. Schuck, M. B. Cai, R. C. Wilson, and Y. Niv, "Human orbitofrontal cortex represents a cognitive map of state space," *Neuron*, vol. 91, no. 6, pp. 1402–1412, 2016.

[137] M. Manakov, M. Proskurin, H. Wang, E. Kuleshova, A. Lustig, R. Behnam, S. Druckmann, D. G. R. Tervo, and A. Y. Karpova, "Cognitive graphs of latent structure in rostral anterior cingulate cortex," *bioRxiv*, pp. 2025–04, 2025.

[138] H. Fotowat, C. Lee, J. J. Jun, and L. Maler, "Neural activity in a hippocampus-like region of the teleost pallium is associated with active sensing and navigation," *Elife*, vol. 8, p. e44119, 2019.

[139] F. Rodríguez, B. Quintero, L. Amores, D. Madrid, C. Salas-Peña, and C. Salas, "Spatial cognition in teleost fish: strategies and mechanisms," *Animals*, vol. 11, no. 8, p. 2271, 2021.

[140] A. Gómez, F. Ocaña, T. del Águila, F. Rodríguez, and C. Salas, "Relational memory functions of the hippocampal pallium in teleost fish," *Evolution of learning and memory mechanisms. Cambridge University Press, Cambridge*, pp. 159–175, 2022.

[141] J. J. Gibson, "The theory of affordances:(1979)," in *The people, place, and space reader*, pp. 56–60, Routledge, 2014.

[142] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," *Advances in neural information processing systems*, vol. 29, 2016.

[143] J. Krupic, N. Burgess, and J. O'Keefe, "Neural representations of location composed of spatially periodic bands," *Science*, vol. 337, no. 6096, pp. 853–857, 2012.

[144] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu, "Using fast weights to attend to the recent past," *Advances in neural information processing systems*, vol. 29, 2016.

[145] M. Soulignac and P. Taillibert, "Fast trajectory planning for multiple site surveillance through moving obstacles and wind," in *Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group*, pp. 25–33, 2006.

[146] U. Rodríguez-Domínguez and J. B. Caplan, "A hexagonal fourier model of grid cells," *Hippocampus*, vol. 29, no. 1, pp. 37–45, 2019.

[147] N. Cook, "Topics in high-dimensional probability, lecture notes," 2024.

[148] E. G. Tabak and E. Vanden-Eijnden, "Density estimation by dual ascent of the log-likelihood," *Communications in Mathematical Sciences*, vol. 8, no. 1, pp. 217–233, 2010.

[149] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature communications*, vol. 9, no. 1, p. 4950, 2018.

[150] J. Bakarji, K. Champion, J. Nathan Kutz, and S. L. Brunton, "Discovering governing equations from partial measurements with deep delay autoencoders," *Proceedings of the Royal Society A*, vol. 479, no. 2276, p. 20230422, 2023.

[151] M. Ostrow, A. Eisen, and I. Fiete, "Delay embedding theory of neural sequence models," *arXiv preprint arXiv:2406.11993*, 2024.

[152] M. Casdagli, S. Eubank, J. D. Farmer, and J. Gibson, "State space reconstruction in the presence of noise," *Physica D: Nonlinear Phenomena*, vol. 51, no. 1-3, pp. 52–98, 1991.

[153] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis*. Cambridge University Press, 2 ed., 2003.

[154] L. C. Uzal, G. L. Grinblat, and P. F. Verdes, "Optimal reconstruction of dynamical systems: A noise amplification approach," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 84, no. 1, p. 016223, 2011.

[155] E. Tan, S. Algar, D. Corrêa, M. Small, T. Stemler, and D. Walker, "Selecting embedding delays: An overview of embedding techniques and a new method using persistent homology," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 33, no. 3, 2023.

[156] P. Kofuji and A. Araque, "Astrocytes and behavior," *Annual review of neuroscience*, vol. 44, no. 1, pp. 49–67, 2021.

[157] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[158] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lock, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[159] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver, "Learning and planning in complex action spaces," in *International Conference on Machine Learning*, pp. 4476–4486, PMLR, 2021.

[160] J. C. Whittington, T. H. Muller, S. Mark, G. Chen, C. Barry, N. Burgess, and T. E. Behrens, "The tolman-eichenbaum machine: unifying space and relational memory through generalization in the hippocampal formation," *Cell*, vol. 183, no. 5, pp. 1249–1263, 2020.

[161] S. J. Gershman, "The successor representation: its computational logic and neural substrates," *Journal of Neuroscience*, vol. 38, no. 33, pp. 7193–7200, 2018.

[162] Y. Ziv, L. D. Burns, E. D. Cocker, E. O. Hamel, K. K. Ghosh, L. J. Kitch, A. E. Gamal, and M. J. Schnitzer, "Long-term dynamics of ca1 hippocampal place codes," *Nature neuroscience*, vol. 16, no. 3, pp. 264–266, 2013.

[163] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

Appendix A

Rational Behaviors

Notation

$\mathbb{O}_S$	The set of all hyperballs in S.
$s(\cdot)$	A *selector* function.
$\check{\delta}(\cdot)$	A *generator* function.
$m_{s,\check{\delta}}(\cdot)$	A *mutator* function.
M	A set $\{m_1, m_2, ..., m_n\}$ of mutator functions.
L	A *mutation schedule* for a set of mutations.
$k_{m_j}(i)$, $k_j(i)$	The schedule-index in a mutation schedule after which the mutator m_j has been called i-times, called the m_j-index.
$m_{M,L}(\cdot)$, $m^i(\cdot)$	A *polymutator* function applying a set M of mutators.
$I_j(\mathbf{b})$	The *index function* the indicates the m_j-index at which a specific hyperball $\mathbf{b}$ is selected for mutation.
$\psi(\mathcal{B}_0, \mathfrak{s}, r)$	The schedule-index after which the plan $\mathfrak{s}$ is guaranteed to be been generated by a polymutator, starting from $\mathcal{B}_0$, which is within $\mathcal{N}_r(\mathfrak{s})$.

A.1 General Mathematical Machinery

Let $\mathbb{O}_S = \{B_\epsilon(\boldsymbol{s}) : \boldsymbol{s} \in S, \epsilon \in \mathbb{R}^+\}$ be the set of all hyperballs in S, and let $\mathbb{P}(\mathbb{O}_S)$ be the set of all sets of hyperballs in S. We will imagine a fixed but arbitrary initial "seed" set of hyperballs, $\mathcal{B}_0 \in \mathbb{P}(\mathbb{O}_S)$, which will be repeatedly transformed. To speak formally about this, I have to introduce some new notions. Throughout, I will use $\mathcal{B}$ as a *generic* set of hyperballs.

Definition A.1 (Selector Function). A *selector function* $s : \mathbb{P}(\mathbb{O}_S) \to \mathbb{O}_S$ selects a single hyperball from a set of hyperballs, so that $\forall \mathcal{B} \in \mathbb{P}(\mathbb{O}_S), s(\mathcal{B}) \in \mathcal{B}$. In other words, $s : \mathcal{B} \mapsto \mathbf{b}, \mathbf{b} \in \mathcal{B}$.

Definition A.2 (Generator Function). A *generator function* $\mathcal{g} : \mathbb{O}_S \to \mathbb{P}(\mathbb{O}_S)$ maps a hyperball to a finite set of hyperballs. If $\mathcal{g} : \mathbf{b} \mapsto \mathcal{B}'$, we call $\mathcal{B}'$ the *children* by $\mathcal{g}$ of $\mathbf{b}$, or the $\mathcal{g}$-1-descendants of $\mathbf{b}$.

Definition A.3 (Mutator Function). $m_{s,\delta} : \mathbb{P}(\mathbb{O}_S) \to \mathbb{P}(\mathbb{O}_S)$ is a *mutator function*, with

$$m_{s,\delta}(\mathcal{B}) = \mathcal{B} \cup \mathcal{g}(s(\mathcal{B}))$$

When it is clear from context, we will write m_j for m_{s_j,δ_j}.

From this definition is plain to see that the operand of a mutator function is contained in its resultant, giving us:

Proposition A.4 (Mutation Containment). $\mathcal{B} \subset m_{s,\delta}(\mathcal{B})$

If we had a single mutator function m, we could recursively call it on our initial set of hyperballs as $m^i(\mathcal{B}_0)$ to model mutating the initial set of hyperballs $\mathcal{B}_0$ with the same mutator function i times. If we had multiple different mutator functions, we could choose to apply a different mutator at each step.

Definition A.5 (Mutation Schedule). Let $M = \{m_1, m_2, ..., m_n\}$ be a finite set of mutator functions, with $m_j = m_{s_j,\delta_j}$. Then $L : \mathbb{N}_1 \to M$ is a *mutation schedule.*

The *individual schedule* for each mutation function $m_j \in M$ is defined such that:

$$k_{m_j}(i) = \min(\{n \in \mathbb{N} : |\{\ell \in [1\,..\,n] : L(\ell) = m_j\}| = i\}) \tag{A.6}$$

In other words, if we increment up $L(1), L(2), L(3)...$, after $k_{m_j}(i)$ "steps", m_j will have been selected i-times.

For brevity we write k_j for k_{m_j}.

Definition A.7 (Polymutator Function). The *polymutator function* $\hat{m}^i_{M,L}(\mathcal{B}_0) = L(i)(\hat{m}^{i-1}_{M,L}(\mathcal{B}_0))$, with $\hat{m}^0_{M,L}(\mathcal{B}_0) = \mathcal{B}_0$, describes the sequential application of mutator functions to an initial set $\mathcal{B}_0$ of hyperballs.

For brevity we write $\hat{m}^i$ for $\hat{m}^i_{M,L}$, and when appropriate, let $\mathcal{B}_i = \hat{m}^i(\mathcal{B}_0)$.

Corollary A.8 (Polymutator Order). $n_1 < n_2 \implies \hat{m}^{n_1}(\mathcal{B}_0) \subset \hat{m}^{n_2}(\mathcal{B}_0)$

Definition A.9 (Fair Polymutator). A polymutator function $\hat{\hat{m}} = \hat{\hat{m}}_{M,L}$ is *fair* if $\forall m_j \in M$, $k_j(\mathbb{N}_1) = \mathbb{N}_1$, which is to say, if the individual schedule of each mutator is onto $\mathbb{N}_1$, meaning that every mutator will be called infinitely many times, eventually.

Proposition A.10 (Schedule Identity). $L(k_{m_j}(i)) = m_j$

Definition A.11 ("Powers" of Single Mutators). For any mutator function $m_j \in M$, $m_j^{[i]}(\mathcal{B}_0) = \hat{\hat{m}}^{k_j(i)}(\mathcal{B}_0)$. This means that $m_j^{[i]}(\mathcal{B}_0)$ is the result of $k_j(i)$ mutations of $\mathcal{B}_0$, *exactly* i of which were m_j-mutations.

Proposition A.12 (Mutator Order). $n_1 < n_2 \implies m^{[n_1]}(\mathcal{B}_0) \subset m^{[n_2]}(\mathcal{B}_0)$

Let $\mathcal{B}^L = \bigcup_{i=0}^{\infty} \hat{\hat{m}}^i(\mathcal{B}_0)$ be the union of all the sets of hyperballs generated by the mutator schedule L. For any given mutator function, we can ask when that mutator function will select a specific hyperball and generate new hyperballs from it, if it ever will. First, we let

$$\texttt{indices}_j(\mathbf{b}) = \left\{ i \in \mathbb{N} : s_j\left(\hat{\hat{m}}^{k_{m_j}(i)-1}(\mathcal{B}_0)\right) = \mathbf{b} \right\} \tag{A.13}$$

From this we define:

Definition A.14 (Index Function). For each mutator function $m_{s_j,\delta_j} \in M$, its *index function* $I[m_{s_j,\delta_j}] : \mathcal{B}^L \to \mathbb{N}_\infty$ maps a hyperball $\mathbf{b}$ to the integer representing the step right before it is selected by the corresponding mutator function m_{s_j,δ_j}, if ever. More formally:

$$I[m_{s_j,\delta_j}](\mathbf{b}) = \begin{cases} \min(\texttt{indices}_j(\mathbf{b})) & \texttt{indices}_j(\mathbf{b}) \neq \varnothing \\ \infty & \texttt{indices}_j(\mathbf{b}) = \varnothing \end{cases}$$

For brevity we write I_j for $I[m_{s_j,\delta_j}]$.

Notice that this means that if $I_j(\mathbf{b}) \in \mathbb{N}$, then $\mathbf{b} = s_j(\hat{\hat{m}}^{k_{m_j}(I_j(\mathbf{b}))-1}(\mathcal{B}_0))$. For the sake of brevity, we introduce the following notational short-hand:

Definition A.15 (Notation Abuse). $k_j(\mathbf{b}) = k_{m_j}(I_j(\mathbf{b}))$

This lets us re-write the previous identity as:

Proposition A.16 (Selection Index).

$$I_j(\mathbf{b}) \in \mathbb{N} \implies \mathbf{b} = s_j\left(\hat{\mathfrak{m}}^{k_j(\mathbf{b})-1}(\mathcal{B}_0)\right)$$

That $I_j(\mathbf{b})$ be a natural number is a profound requirement for our construction. To codify it as a constraint, we introduce the follow

Definition A.17 (Balanced Mutator Function). A mutator function $\mathfrak{m}_j$ is *balanced* with respect to $\mathcal{L}$ if and only if:

$$\forall \mathbf{b} \in \mathcal{B}^{\mathcal{L}},\ I[\mathfrak{m}_j](\mathbf{b}) \in \mathbb{N}$$

Henceforth, we will assume that all mutators are *balanced*, from which follows:

Lemma A.18 (Children Index). $\mathfrak{z}_j(\mathbf{b}) \subset \mathfrak{m}_j^{[I_j(\mathbf{b})]}(\mathcal{B}_0)$

Proof:

1. $\mathfrak{m}_j^{[I_j(\mathbf{b})]}(\mathcal{B}_0) = \hat{\mathfrak{m}}^{k_j(I_j(\mathbf{b}))}(\mathcal{B}_0) = \hat{\mathfrak{m}}^{k_j(\mathbf{b})}(\mathcal{B}_0)$ (By A.11)
2. $= \mathcal{L}(k_j(\mathbf{b}))\left(\hat{\mathfrak{m}}^{k_j(\mathbf{b})-1}(\mathcal{B}_0)\right)$ (By A.7)
3. $= \mathfrak{m}_j\left(\hat{\mathfrak{m}}^{k_j(\mathbf{b})-1}(\mathcal{B}_0)\right)$ (By A.10)
4. $= \hat{\mathfrak{m}}^{k_j(\mathbf{b})-1}(\mathcal{B}_0) \cup \mathfrak{z}_j\left(s_j\left(\hat{\mathfrak{m}}^{k_j(\mathbf{b})-1}(\mathcal{B}_0)\right)\right)$ (By A.3)
5. $= \hat{\mathfrak{m}}^{k_j(\mathbf{b})-1}(\mathcal{B}_0) \cup \mathfrak{z}_j(\mathbf{b})$ (By A.16)
6. $\supset \mathfrak{z}_j(\mathbf{b})$

□

Thus, all of the $\mathfrak{z}_j$-1-descendants of $\mathbf{b}$ will exist in $\mathfrak{m}_j^{[I_j(\mathbf{b})]}(\mathcal{B}_0)$, which is to say, after $k_j(I_j(\mathbf{b}))$ "steps" of the mutation schedule $\mathcal{L}$. But, what about the "grandchildren" (2-children), "great-grandchildren (3-children) and further n-descendants by $\mathfrak{z}_j$ of $\mathbf{b}$?

Definition A.19 (Powers of Generator Functions). The set of i-descendants of hyperball $\mathbf{b}$ is:

$$\mathfrak{Z}^i(\mathbf{b}) = \begin{cases} \{\mathbf{b}\} & i = 0 \\ \bigcup\{\mathfrak{Z}(\mathbf{b}') : \mathbf{b}' \in \mathfrak{Z}^{i-1}(\mathbf{b})\} & i > 0 \end{cases}$$

Note that this extension is compatible with the original definition of a generator, as

$$\begin{aligned} \mathcal{G}(\mathbf{b}) &= \mathcal{G}^1(\mathbf{b}) && \text{(A.20)}\\ &= \bigcup\{\mathcal{G}(\mathbf{b}') : \mathbf{b}' \in \mathcal{G}^0(\mathbf{b})\} && \text{(A.21)}\\ &= \bigcup\{\mathcal{G}(\mathbf{b}') : \mathbf{b}' \in \mathcal{G}^0(\mathbf{b})\} && \text{(A.22)}\\ &= \bigcup\{\mathcal{G}(\mathbf{b}') : \mathbf{b}' \in \{\mathbf{b}\}\} && \text{(A.23)}\\ &= \mathcal{G}(\mathbf{b}) && \text{(A.24)} \end{aligned}$$

We can now ask, when will every i-descendant by $\mathcal{G}_j$ of $\mathbf{b}$ exist, which is to say, be generated from its parent?

Definition A.25 (Powers of Index Functions).

$$I_j^i(\mathbf{b}) = \max(\{I_j(\mathbf{b}') : \mathbf{b}' \in \mathcal{G}_j^{i-1}(\mathbf{b})\}),\ i > 0$$

Corollary A.26 (Index Order). $n_1 < n_2 \implies I_j^{n_1}(\mathbf{b}) < I_j^{n_2}(\mathbf{b})$

From this definition we can easily derive:

Lemma A.27 (Generation Index). $\mathcal{G}_j^i(\mathbf{b}) \subset m_j^{[I_j^i(\mathbf{b})]}(\mathcal{B}_0)$

Proof:

1.		$I_j^i(\mathbf{b}) = \max(\{I_j(\mathbf{b}') : \mathbf{b}' \in \mathcal{G}_j^{i-1}(\mathbf{b})\})$	(Def. A.25)
2.		$\forall \mathbf{b}' \in \mathcal{G}_j^{i-1}(\mathbf{b})$,	
	2.a	$I_j(\mathbf{b}') \leq I_j^i(\mathbf{b})$	(By def. of max)
	2.b	$m_j^{[I_j(\mathbf{b}')]}(\mathcal{B}_0) \subset m_j^{[I_j^i(\mathbf{b})]}(\mathcal{B}_0)$	(By step 2.a and A.12)
	2.c	$\mathcal{G}_j(\mathbf{b}') \subset m_j^{[I_j(\mathbf{b}')]}(\mathcal{B}_0) \subset m_j^{[I_j^i(\mathbf{b})]}(\mathcal{B}_0)$	(By A.18)
	2.d	$\mathcal{G}_j(\mathbf{b}') \subset m_j^{[I_j^i(\mathbf{b})]}(\mathcal{B}_0)$	
3.		$\bigcup\{\mathcal{G}_j(\mathbf{b}') : \mathbf{b}' \in \mathcal{G}_j^{i-1}(\mathbf{b})\} \subset m_j^{[I_j^i(\mathbf{b})]}(\mathcal{B}_0)$	
4.		$\mathcal{G}_j^i(\mathbf{b}) \subset m_j^{[I_j^i(\mathbf{b})]}(\mathcal{B}_0)$	(By A.19)

□

To say more succinctly what this means, we introduce more notational sugar:

Definition A.28 (Powers of Notation Abuse). $k^i_j(\mathbf{b}) = k_{m_j}(I^i_j(\mathbf{b}))$

So all of $\xi^i_{\delta j}(\mathbf{b})$, the ξ_j-i-descendants of the hyperball $\mathbf{b}$, exist in $m_j^{[I^i_j(\mathbf{b})]}(\mathcal{B}_0)$, meaning that they exist by the $k^i_j(\mathbf{b})^{\text{th}}$ step of the mutation schedule $\mathcal{L}$. For a set of hyperballs $\mathcal{B}$, when do *all* of their ξ_j-i-descendants exist?

Definition A.29 (Set Generator Function). We extend the domain of a generator function to sets of hyperballs by defining

$$\xi^i_{\delta j}(\mathcal{B}) = \bigcup\{\xi^i_{\delta j}(\mathbf{b}) : \mathbf{b} \in \mathcal{B}\}$$

Definition A.30 (Set Index Function). $I^i_j(\mathcal{B}) = \max(\{I^i_j(\mathbf{b}) : \mathbf{b} \in \mathcal{B}\})$

From these definitions it is easy to show

Lemma A.31 (Set Generation Index). $\xi^i_{\delta j}(\mathcal{B}) \subset m_j^{[I^i_j(\mathcal{B})]}(\mathcal{B}_0)$

Proof:

1.		$I^i_j(\mathcal{B}) = \max(\{I^i_j(\mathbf{b}) : \mathbf{b} \in \mathcal{B}\})$	(Def. A.30)
2.		$\forall \mathbf{b} \in \mathcal{B}$,	
	2.a	$I^i_j(\mathbf{b}) \leq I^i_j(\mathcal{B})$	(By def. of max)
	2.b	$m_j^{[I^i_j(\mathbf{b})]}(\mathcal{B}_0) \subset m_j^{[I^i_j(\mathcal{B})]}(\mathcal{B}_0)$	(By A.31.2.a and A.12)
	2.c	$\xi^i_{\delta j}(\mathbf{b}) \subset m_j^{[I^i_j(\mathbf{b})]}(\mathcal{B}_0) \subset m_j^{[I^i_j(\mathcal{B})]}(\mathcal{B}_0)$	(By A.27)
	2.d	$\xi^i_{\delta j}(\mathbf{b}) \subset m_j^{[I^i_j(\mathcal{B})]}(\mathcal{B}_0)$	
3.		$\bigcup\{\xi^i_{\delta j}(\mathbf{b}) : \mathbf{b} \in \mathcal{B}\} \subset m_j^{[I^i_j(\mathcal{B})]}(\mathcal{B}_0)$	
4.		$\xi^i_{\delta j}(\mathcal{B}) \subset m_j^{[I^i_j(\mathcal{B})]}(\mathcal{B}_0)$	(By A.29)

□

Which is to say that all of the m_j-i-descendants of every hyperball in $\mathcal{B}$ exists in $m_j^{[I^i_j(\mathcal{B})]}(\mathcal{B}_0)$. Further stretching our notation, $k^i_j(\mathcal{B}) = k_{m_j}(I^i_j(\mathcal{B}))$, meaning that every m_j-i-descendant of every hyperball in $\mathcal{B}$ exists by the $k^i_j(\mathcal{B})^{\text{th}}$ step of the mutation schedule $\mathcal{L}$.

A.2 Coverage by Extension

Let's restate the definition of the *extension* function:

Definition A.32 (Extension). The extension function `extend` takes in three arguments, a hyperball $\mathbf{b}$ and two scaling parameters $a > 1$ and $c > 1$, and returns a new set of hyperballs covering "new" regions of S:

$$\texttt{extend}(\mathbf{b}, a, c) = \{\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, ...\mathbf{b}^{(k)}\}$$

with the following properties:

1. $\forall \mathbf{b}' \in \texttt{extend}(\mathbf{b}, a, c), \epsilon(\mathbf{b}') = a \cdot \epsilon(\mathbf{b})$
2. $B_{c \cdot \epsilon(\mathbf{b})}(\boldsymbol{s}(\mathbf{b})) \subset \mathbf{b} \cup (\bigcup \texttt{extend}(\mathbf{b}, a, c))$

For the sake of convenience we will use $\mathcal{Z}_{\blacktriangle}(\mathbf{b})$ for $\texttt{extend}(\mathbf{b}, a, c)$, with $m_{\blacktriangle} = m_{s_{\blacktriangle}, \delta_{\blacktriangle}}$. If we keep mutating our initial set of "seed" hyperballs $\mathcal{B}_0$, how large a region can we cover?

Definition A.33 (Family Function). The union of all descendants of a hyperball $\mathbf{b}$ up to the i^{th}-generation is referred to as the *i-family* of $\mathbf{b}$:

$$\hat{\mathcal{Z}}^i_j(\mathbf{b}) = \bigcup_{l=0}^{i} \mathcal{Z}^l_j(\mathbf{b})$$

Notice that $\hat{\mathcal{Z}}_j(\mathbf{b}) = \{\mathbf{b}\} \cup \mathcal{Z}_j(\mathbf{b})$. We denote the area covered by the i-family of $\mathbf{b}$ as:

Definition A.34 (Family Area). $A^i(\mathbf{b}) = \bigcup \hat{\mathcal{Z}}^i_{\blacktriangle}(\mathbf{b})$

From these definitions we can see that

Proposition A.35 (Single Extension Cover). $B_{c\epsilon(\mathbf{b})}(\boldsymbol{s}(\mathbf{b})) \subset A^1(\mathbf{b})$

Furthermore, we can derive the following relationship:

Lemma A.36 (Family Area Recurrence Relation).

$$A^{i+1}(\mathbf{b}) = A^i(\mathbf{b}) \cup \bigcup \{A^1(\mathbf{b}') : \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b})\}$$

Proof:

1.	A^{i+1}	$= \bigcup \hat{\zeta}^{i+1}_{\blacktriangle}(\mathbf{b})$	(Def. A.34)
2.		$= \bigcup \left\{ \bigcup_{l=0}^{i+1} \zeta^{l}_{\blacktriangle}(\mathbf{b}) \right\}$	(Def. A.33)
3.		$= \bigcup \left\{ \bigcup_{l=0}^{i} \zeta^{l}_{\blacktriangle}(\mathbf{b}) \right\} \cup \left(\bigcup \zeta^{i}_{\blacktriangle}(\mathbf{b}) \cup \bigcup \zeta^{i+1}_{\blacktriangle}(\mathbf{b}) \right)$	
4.		$= \bigcup \hat{\zeta}^{i}_{\blacktriangle}(\mathbf{b}) \cup \left(\bigcup \zeta^{i}_{\blacktriangle}(\mathbf{b}) \cup \bigcup \zeta^{i+1}_{\blacktriangle}(\mathbf{b}) \right)$	(Def. A.33)
5.		$= A^{i}(\mathbf{b}) \cup \left(\bigcup \zeta^{i}_{\blacktriangle}(\mathbf{b}) \cup \bigcup \zeta^{i+1}_{\blacktriangle}(\mathbf{b}) \right)$	(Def. A.34)
6.		$= A^{i}(\mathbf{b}) \cup \bigcup \{ \mathbf{b}' \cup \bigcup_{\zeta_{\blacktriangle}}(\mathbf{b}') : \mathbf{b}' \in \zeta^{i}_{\blacktriangle}(\mathbf{b}) \}$	
7.		$= A^{i}(\mathbf{b}) \cup \bigcup \{ \bigcup \hat{\zeta}_{\blacktriangle}(\mathbf{b}') : \mathbf{b}' \in \zeta^{i}_{\blacktriangle}(\mathbf{b}) \}$	(Def. A.33)
8.		$= A^{i}(\mathbf{b}) \cup \bigcup \{ A^{1}(\mathbf{b}') : \mathbf{b}' \in \zeta^{i}_{\blacktriangle}(\mathbf{b}) \}$	(Def. A.34)

□

Now, letting ∂X be the *boundary* for any set X, notice that:

Proposition A.37 (Family Area Boundary). $\partial A^{i}(\mathbf{b}) \subset \bigcup \zeta^{i}_{\blacktriangle}(\mathbf{b})$

This follows from the fact that $\forall i \in \mathbb{N}, \hat{\zeta}^{i-1}_{\blacktriangle}(\mathbf{b}) \subset \hat{\zeta}^{i}_{\blacktriangle}(\mathbf{b})$.

Before we go on, I need to introduce the following operation:

Definition A.38 (Dilation). $D_r(X) = \bigcup \{ B_r(\boldsymbol{s}) : \boldsymbol{s} \in X \}$

as well as several propositions about it:

Proposition A.39 (Boundary Dilation). $X \cup D_r(\partial X) = D_r(X)$

Proposition A.40 (Hyperball Dilation).

$$D_{r_2}(B_{r_1}(\boldsymbol{s})) = B_{r_1 + r_2}(\boldsymbol{s})$$

Proposition A.41 (Dilation Preserves Subsets).

$$X \subset Y \implies D_r(X) \subset D_r(Y)$$

Now, we can return to the matter of the family area. Notice that:

Proposition A.42 (Extension Hyperball Size). $\forall \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b}),\ \epsilon(\mathbf{b}') = a^i \cdot \epsilon(\mathbf{b})$

For convenience I will write $\epsilon_{\mathbf{b}}$ for $\epsilon(\mathbf{b})$, meaning that $\forall \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b}),\ \epsilon_{\mathbf{b}'} = a^i \cdot \epsilon_{\mathbf{b}}$.

Lemma A.43 (Area Lower Limit). $A^{i+1} \supset D_{a^i \epsilon_{\mathbf{b}}(c-1)}(A^i(\mathbf{b}))$

Proof:

1. $A^{i+1} = A^i(\mathbf{b}) \cup \bigcup \{A^1(\mathbf{b}') : \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b})\}$ (By A.36)
2. $\supset A^i(\mathbf{b}) \cup \bigcup \left\{B_{c\epsilon_{\mathbf{b}'}}(\boldsymbol{s}_{\mathbf{b}'}) : \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b})\right\}$ (By A.35)

2.a $c\epsilon_{\mathbf{b}'} = c\epsilon_{\mathbf{b}'} - \epsilon_{\mathbf{b}'} + \epsilon_{\mathbf{b}'}$

2.b $= \epsilon_{\mathbf{b}'}(c-1) + \epsilon_{\mathbf{b}'}$

3. $\supset A^i(\mathbf{b}) \cup \bigcup \left\{B_{\epsilon_{\mathbf{b}'}(c-1)+\epsilon_{\mathbf{b}'}}(\boldsymbol{s}_{\mathbf{b}'}) : \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b})\right\}$
4. $\supset A^i(\mathbf{b}) \cup \bigcup \left\{D_{\epsilon_{\mathbf{b}'}(c-1)}(B_{\epsilon_{\mathbf{b}'}}(\boldsymbol{s}_{\mathbf{b}'})) : \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b})\right\}$ (By A.40)
5. $\supset A^i(\mathbf{b}) \cup \bigcup \left\{D_{\epsilon_{\mathbf{b}'}(c-1)}(\mathbf{b}') : \mathbf{b}' \in \mathcal{Z}^i_{\blacktriangle}(\mathbf{b})\right\}$
6. $\supset A^i(\mathbf{b}) \cup D_{\epsilon_{\mathbf{b}'}(c-1)}\left(\bigcup \mathcal{Z}^i_{\blacktriangle}(\mathbf{b})\right)$
7. $\supset A^i(\mathbf{b}) \cup D_{\epsilon_{\mathbf{b}'}(c-1)}(\partial A^i(\mathbf{b}))$ (By A.37 and A.41)
8. $\supset D_{\epsilon_{\mathbf{b}'}(c-1)}(A^i(\mathbf{b}))$ (By A.39)
9. $\supset D_{a^i \epsilon_{\mathbf{b}}(c-1)}(A^i(\mathbf{b}))$ (By A.42)

□

Lemma A.44 (Extension Coverage Radius).

$$\forall i \in \mathbb{N},\ B_{R_i}(\boldsymbol{s}_{\mathbf{b}}) \subset A^i(\mathbf{b}), \text{ with } R_i = \epsilon_{\mathbf{b}} + \epsilon_{\mathbf{b}}(c-1)\tfrac{a^i-1}{a-1}$$

Proof: We will prove by induction.

1. $A^0(\mathbf{b}) = \bigcup_{l=0}^{0} \mathcal{Z}^l_{\blacktriangle}(\mathbf{b})$ (By A.33 and A.34)
2. $= \mathbf{b} = B_{\epsilon_{\mathbf{b}}}(\boldsymbol{s}_{\mathbf{b}})$

The base-case: let $R_0 = \epsilon_{\mathbf{b}}$.

3. $B_{R_0}(\boldsymbol{s}_{\mathbf{b}}) \subset A^0(\mathbf{b})$

The inductive-step: $R_{i+1} = R_i + a^i R_0(c-1)$. Assume that $B_{R_i}(\boldsymbol{s}_\mathbf{b}) \subset A^i(\mathbf{b})$.

4. $B_{R_i}(\boldsymbol{s}_\mathbf{b}) \subset A^i(\mathbf{b})$ (By Assumption)
5. $D_{a^i\epsilon_\mathbf{b}(c-1)}(B_{R_i}(\boldsymbol{s}_\mathbf{b})) \subset D_{a^i\epsilon_\mathbf{b}(c-1)}(A^i(\mathbf{b}))$ (By A.41)
6. $B_{R_i+a^i\epsilon_\mathbf{b}(c-1)}(\boldsymbol{s}_\mathbf{b}) \subset D_{a^i\epsilon_\mathbf{b}(c-1)}(A^i(\mathbf{b}))$ (By A.40)
7. $B_{R_i+a^i\epsilon_\mathbf{b}(c-1)}(\boldsymbol{s}_\mathbf{b}) \subset A^{i+1}(\mathbf{b})$ (By A.43)
8. $B_{R_{i+1}}(\boldsymbol{s}_\mathbf{b}) \subset A^{i+1}(\mathbf{b})$
9. $B_{R_i}(\boldsymbol{s}_\mathbf{b}) \subset A^i(\mathbf{b}) \implies B_{R_{i+1}}(\boldsymbol{s}_\mathbf{b}) \subset A^{i+1}(\mathbf{b})$ (By A.44.4 and A.44.8)
10. $\forall i \in \mathbb{N},\ B_{R_i}(\boldsymbol{s}_\mathbf{b}) \subset A^i(\mathbf{b})$ (By induction)

Now, for the derivation of R_i:

11. $R_i = R_{i-1} + a^{i-1}\epsilon_\mathbf{b}(c-1)$
12. $= \epsilon_\mathbf{b} + \sum_{n=0}^{i-1}\epsilon_\mathbf{b}(c-1)a^n$
13. $= \epsilon_\mathbf{b} + \epsilon_\mathbf{b}(c-1)\sum_{n=0}^{i-1}a^n$
14. $= \epsilon_\mathbf{b} + \epsilon_\mathbf{b}(c-1)\frac{a^i-1}{a-1}$

□

Now, for a given radius R, we can calculate how many generations it will take to match or exceed that radius, $i^\mathbf{b}_\blacktriangle(R)$:

$$\begin{aligned}
\epsilon_\mathbf{b} + \epsilon_\mathbf{b}(c-1)\tfrac{a^i-1}{a-1} &= R \\
\epsilon_\mathbf{b}(c-1)\tfrac{a^i-1}{a-1} &= R - \epsilon_\mathbf{b} \\
\tfrac{a^i-1}{a-1} &= \tfrac{R-\epsilon_\mathbf{b}}{\epsilon_\mathbf{b}(c-1)} \\
a^i - 1 &= \tfrac{R-\epsilon_\mathbf{b}}{\epsilon_\mathbf{b}(c-1)}(a-1) \\
a^i &= \tfrac{R-\epsilon_\mathbf{b}}{\epsilon_\mathbf{b}(c-1)}(a-1) + 1 \\
i &= \log_a\left(\tfrac{(R-\epsilon_\mathbf{b})(a-1)}{\epsilon_\mathbf{b}(c-1)} + 1\right)
\end{aligned}$$

The actual number of generations must be an integer as big or bigger than i, which we denote as:

$$i^\mathbf{b}_\blacktriangle(R) = \left\lceil \log_a\left(\tfrac{(R-\epsilon_\mathbf{b})(a-1)}{\epsilon_\mathbf{b}(c-1)} + 1\right)\right\rceil \tag{A.45}$$

By construction then, we have that:

Proposition A.46 (Radius Inequality). $R \leq R_{i^\mathbf{b}_\blacktriangle(R)}$

From this we can show that:

Lemma A.47 (Point Coverage).

$$\boldsymbol{s} \in S,\ \mathbf{b} \in \mathbb{O}_S \implies \boldsymbol{s} \in A^{i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s}-\boldsymbol{s}_{\mathbf{b}}\|)}(\mathbf{b})$$

Proof:

1. $\boldsymbol{s} \in B_{\|\boldsymbol{s}-\boldsymbol{s}_{\mathbf{b}}\|}(\boldsymbol{s}_{\mathbf{b}})$
2. $\in B_{\|\boldsymbol{s}-\boldsymbol{s}_{\mathbf{b}}\|}(\boldsymbol{s}_{\mathbf{b}}) \subset B_{R_{i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s}-\boldsymbol{s}_{\mathbf{b}}\|)}}(\boldsymbol{s}_{\mathbf{b}})$ (By A.46)
3. $\in B_{R_{i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s}-\boldsymbol{s}_{\mathbf{b}}\|)}}(\boldsymbol{s}_{\mathbf{b}}) \subset A^{i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s}-\boldsymbol{s}_{\mathbf{b}}\|)}(\mathbf{b})$ (By A.44)
4. $\in A^{i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s}-\boldsymbol{s}_{\mathbf{b}}\|)}(\mathbf{b})$

□

This means that for any starting hyperball $\mathbf{b}$, any arbitrary point $\boldsymbol{s}$ will be covered by $\mathbf{b}$'s i-family when $i = i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s} - \boldsymbol{s}_{\mathbf{b}}\|)$. When will every point (sub-goal) in a plan $\mathfrak{s}$ be covered by $\mathbf{b}$'s i-family? Well, each point $\boldsymbol{s} \in \mathfrak{s}$ will have a (potentially different) "coverage" generation, $i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s} - \boldsymbol{s}_{\mathbf{b}}\|)$. The maximum of these integers will correspond to the generation that covers *all* of the points in $\mathfrak{s}$.

$$i^{\mathbf{b}}_{\blacktriangle}(\mathfrak{s}) = \max(\{i^{\mathbf{b}}_{\blacktriangle}(\|\boldsymbol{s} - \boldsymbol{s}_{\mathbf{b}}\|) : \boldsymbol{s} \in \mathfrak{s}\}) \tag{A.48}$$

By construction, $\forall \boldsymbol{s} \in \mathfrak{s}$, $\boldsymbol{s} \in A^{i^{\mathbf{b}}_{\blacktriangle}(\mathfrak{s})}(\mathbf{b})$, so by definition:

Proposition A.49 (Hyperball Family Plan Coverage).

$$\forall \mathfrak{s} \in \mathfrak{S}^*_S \text{ and } \forall \mathbf{b} \in \mathbb{O}_S,\ \mathfrak{s} \Subset A^{i^{\mathbf{b}}_{\blacktriangle}(\mathfrak{s})}(\mathbf{b})$$

Consider again our arbitrary but fixed set of "seed" hyperballs, $\mathcal{B}_0$. We know by the previous proposition that for any plan $\mathfrak{s}$ and hyperball $\mathbf{b} \in \mathcal{B}_0$, $\mathfrak{s} \Subset A^{i^{\mathbf{b}}_{\blacktriangle}(\mathfrak{s})}$. In the course of $\mathcal{B}_0$ being mutated, for each hyperball $\mathbf{b} \in \mathcal{B}_0$ this will occur, per A.27, after $I^{i^{\mathbf{b}}_{\blacktriangle}(\mathfrak{s})}_{\blacktriangle}(\mathbf{b})$ applications of the extension function. The first time this occurs for *any* of the hyperballs, $\mathfrak{s}$ will be covered, so we only care about the *minimum* of these values:

$$I^{\star}_{\blacktriangle}(\mathfrak{s}) = \min(\{I^{i^{\mathbf{b}}_{\blacktriangle}(\mathfrak{s})}_{\blacktriangle}(\mathbf{b}) : \mathbf{b} \in \mathcal{B}_0\}) \tag{A.50}$$

From this definition, by the fact that $\forall \mathbf{b} \in \mathcal{B}_0, I^{i^{\mathbf{b}}_{\blacktriangle}(\mathfrak{s})}_{\blacktriangle}(\mathbf{b}) \leq I^{\star}_{\blacktriangle}(\mathfrak{s})$, we can see that:

Proposition A.51 (Universal Plan Coverage).

$$\forall \mathfrak{s} \in \mathfrak{S}^*_S, \exists \mathbf{b} \in \mathcal{B}_0 \text{ such that } \mathfrak{s} \Subset A^{I^{\star}_{\blacktriangle}(\mathfrak{s})}(\mathbf{b})$$

This of course only tells us how many applications of the *extension* mutator it takes to guarantee that $\mathfrak{s}$ is covered. Letting $k^{\star}_{\blacktriangle}(\mathfrak{s}) = k_{m_{\blacktriangle}}(I^{\star}_{\blacktriangle}(\mathfrak{s}))$, we can derive:

Theorem A.52 (Behavior Coverage by Extension). If $\hat{\mathring{m}}$ is a *fair* polymutator function, and $m_{\blacktriangle}$ is a *balanced* mutator function, then:

$$\mathfrak{s} \in \mathfrak{S}_S^* \implies \mathfrak{s} \Subset \hat{\mathring{m}}^{k_{\blacktriangle}^\star(\mathfrak{s})}(\mathcal{B}_0)$$

Proof:

1.	$\mathfrak{s} \Subset A^{i_{\blacktriangle}^{\mathbf{b}}(\mathfrak{s})}(\mathbf{b})$	(By A.51)
2.	$\Subset \hat{\mathcal{E}}_{\blacktriangle}^{I_{\blacktriangle}^\star(\mathfrak{s})}(\mathbf{b})$	(By A.34)
3.	$\Subset \hat{\mathcal{E}}_{\blacktriangle}^{I_{\blacktriangle}^\star(\mathfrak{s})}(\mathcal{B}_0)$	(By A.29)
4.	$\Subset m_{\blacktriangle}^{[I_{\blacktriangle}^\star(\mathfrak{s})]}(\mathcal{B}_0)$	(By A.31)
5.	$\Subset \hat{\mathring{m}}^{k_{\blacktriangle}^\star(\mathfrak{s})}(\mathcal{B}_0)$	(By A.11)

A.3 Precision by Refinement

For a given behavior $\mathfrak{s}$, imagine that we have a set of hyperballs, $\mathcal{B}_{\blacktriangle} = \hat{\mathring{m}}^{k_{\blacktriangle}^\star(\mathfrak{s})}(\mathcal{B}_0)$, so that $\mathfrak{s} \Subset \mathcal{B}_{\blacktriangle}$. Notice that $\mathfrak{s} \Subset \mathcal{B}$ does not imply that $\exists \mathfrak{b} \in \mathfrak{B}_{\mathcal{B}_{\blacktriangle}}^*$ such that $\mathfrak{s} \in [\![\mathfrak{b}]\!]$, and even if there *does* exist such a $\mathfrak{b} \in \mathfrak{B}_{\mathcal{B}_{\blacktriangle}}^*$, it does not imply that $r_{_K}(\mathfrak{b}) = 1$. However, we can *refine* hyperballs in $\mathcal{B}_{\blacktriangle}$ to fix this problem. Briefly, let's restate the definition of refinement:

Definition A.53 (Refinement). The refinement function `refine` takes in two arguments, $\mathbf{b}$ a hyperball and $a \in (0, 1)$ a scaling parameter, and returns a set of new "denser" hyperballs:

$$\texttt{refine}(\mathbf{b}, a) = \{\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, ...\mathbf{b}^{(k)}\}$$

with the following properties:

1. $\boldsymbol{s}(\mathbf{b}^{(1)}) = \boldsymbol{s}(\mathbf{b})$
2. $\forall \mathbf{b}' \in \texttt{refine}(\mathbf{b}, a), \epsilon(\mathbf{b}') = a \cdot \epsilon(\mathbf{b})$
3. $\mathbf{b} \subset \bigcup \texttt{refine}(\mathbf{b}, a)$

For the sake of convenience we will use $\mathcal{Z}_{\blacktriangledown}(\mathbf{b})$ for $\texttt{refine}(\mathbf{b}, a)$, with $m_{\blacktriangledown} = m_{s_{\blacktriangledown}, \mathcal{Z}_{\blacktriangledown}}$. Notice also that $\mathbf{b} \subset \bigcup \mathcal{Z}_{\blacktriangledown}(\mathbf{b})$. From this we can derive

Proposition A.54 (Refinement Generation Coverage).

$$\forall i \in \mathbb{N}, \bigcup \mathcal{Z}_{\blacktriangledown}^i(\mathcal{B}) \subset \bigcup \mathcal{Z}_{\blacktriangledown}^{i+1}(\mathcal{B})$$

Proof:

1.	$\mathcal{Z}_{\blacktriangledown}^{i+1}(\mathcal{B})$	$= \bigcup\{\mathcal{Z}_{\blacktriangledown}^{i+1}(\mathbf{b}) : \mathbf{b} \in \mathcal{B}\}$	(By A.29)
2.		$= \bigcup\{\bigcup\{\mathcal{Z}_{\blacktriangledown}(\mathbf{b}') : \mathbf{b}' \in \mathcal{Z}_{\blacktriangledown}^{i}(\mathbf{b})\} : \mathbf{b} \in \mathcal{B}\}$	(By A.19)
3.		$= \bigcup\{\mathcal{Z}_{\blacktriangledown}(\mathbf{b}) : \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i}(\mathcal{B})\}$	
4.	$\bigcup \mathcal{Z}_{\blacktriangledown}^{i+1}(\mathcal{B})$	$= \bigcup\bigcup\{\mathcal{Z}_{\blacktriangledown}(\mathbf{b}) : \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i}(\mathcal{B})\}$	
5.		$= \bigcup\{\bigcup\mathcal{Z}_{\blacktriangledown}(\mathbf{b}) : \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i}(\mathcal{B})\}$	
6.		$\supset \bigcup\{\mathbf{b} : \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i}(\mathcal{B})\}$	(By A.53.3)
7.		$\supset \bigcup \mathcal{Z}_{\blacktriangledown}^{i}(\mathcal{B})$	

□

From which it follows by induction that:

Proposition A.55 (Refinement Progenitor Coverage).

$$\forall i \in \mathbb{N},\ \bigcup \mathcal{B} \subset \bigcup \mathcal{Z}_{\blacktriangledown}^{i}(\mathcal{B})$$

We can also see from the definition of $\mathcal{Z}_{\blacktriangledown}$ that:

Proposition A.56 (Refinement Scaling).

$$\forall \mathbf{b}' \in \mathcal{Z}_{\blacktriangledown}^{i}(\mathbf{b}),\ \epsilon(\mathbf{b}') = a^i \cdot \epsilon(\mathbf{b})$$

Which is to say that after i generations, all hyperballs produced by refinement in the i-generation from $\mathbf{b}$ will have a radius a^i-times as big as that of $\mathbf{b}$. How many generations have to pass until the hyperballs produced have a radius as small or smaller than a given value, r?

$$\begin{aligned} \epsilon_{\mathbf{b}} \cdot a^i &= r \\ a^i &= \tfrac{r}{\epsilon_{\mathbf{b}}} \\ i &= \log_a\left(\tfrac{r}{\epsilon_{\mathbf{b}}}\right) \end{aligned}$$

The actual number of generations must be an integer as big or bigger than i, which we denote as:

$$i_{\blacktriangledown}^{\mathbf{b}}(r) = \left\lceil \log_a\left(\tfrac{r}{\epsilon_{\mathbf{b}}}\right)\right\rceil \tag{A.57}$$

For a set of hyperballs $\mathcal{B}$, each hyperball $\mathbf{b}$ in the set will require a different number of generations to reach a stage where all descendants in a generation have a radius smaller than r. Formally:

Lemma A.58. $\forall \mathbf{b}' \in \mathcal{Z}_{\blacktriangledown}^{i_{\blacktriangledown}^{\mathbf{b}}(r)}(\mathbf{b}),\ \epsilon(\mathbf{b}') \leq r$

Proof:

1. $\forall \mathbf{b}' \in \mathcal{Z}_{\blacktriangledown}^{i_{\blacktriangledown}^{\mathbf{b}}(r)}(\mathbf{b}),\ \epsilon(\mathbf{b}') = a^{i_{\blacktriangledown}^{\mathbf{b}}(r)} \cdot \epsilon(\mathbf{b})$ (By A.56)
2. $= a^{\left\lceil \log_a\left(\frac{r}{\epsilon_{\mathbf{b}}}\right)\right\rceil} \cdot \epsilon_{\mathbf{b}}$ (By Def. A.57)
3. $\leq a^{\log_a\left(\frac{r}{\epsilon_{\mathbf{b}}}\right)} \cdot \epsilon_{\mathbf{b}}$ (By $a < 1$)
4. $\leq \frac{r}{\epsilon_{\mathbf{b}}} \cdot \epsilon_{\mathbf{b}}$
5. $\leq r$

□

From this it also follows that

$$\forall n > i_{\blacktriangledown}^{\mathbf{b}}(r),\ \forall \mathbf{b}' \in \mathcal{Z}_{\blacktriangledown}^{i_{\blacktriangledown}^{\mathbf{b}}(r)}(\mathbf{b}),\ \epsilon(\mathbf{b}') \leq r \tag{A.59}$$

The number of generations after which they *all* have descendants of radius less than r is given by:

$$i_{\blacktriangledown}^{\mathcal{B}}(r) = \max(\{i_{\blacktriangledown}^{\mathbf{b}}(r) : \mathbf{b} \in \mathcal{B}\}) \tag{A.60}$$

Using this definition we can derive the following:

Lemma A.61 (Refinement Radius). $\forall \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i_{\blacktriangledown}^{\mathcal{B}}(r)}(\mathcal{B}),\ \epsilon(\mathbf{b}) \leq r$

Proof:

1. $i_{\blacktriangledown}^{\mathcal{B}}(r) = \max(\{i_{\blacktriangledown}^{\mathbf{b}}(r) : \mathbf{b} \in \mathcal{B}\})$ (By Def. A.60)
2. $\forall \mathbf{b} \in \mathcal{B},\ i_{\blacktriangledown}^{\mathbf{b}}(r) \leq i_{\blacktriangledown}^{\mathcal{B}}(r)$
3. $\forall \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i_{\blacktriangledown}^{\mathbf{b}}(r)}(\mathbf{b}),\ \epsilon(\mathbf{b}) \leq r$ (By A.58)
4. $\forall \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i_{\blacktriangledown}^{\mathcal{B}}(r)}(\mathcal{B}),\ \epsilon(\mathbf{b}) \leq r$ (By A.59)

□

Here, notation begins to get a little heavier, so we introduce the following definitions:

Definition A.62 (Refinement Index). $I_{\blacktriangledown}^{\star}(\mathcal{B}, r) = I_{\blacktriangledown}^{i_{\blacktriangledown}^{\mathcal{B}}(r)}(\mathcal{B})$

Definition A.63 (Polymutator Refinement Index). $k_{\blacktriangledown}^{\star}(\mathcal{B}, r) = k_{m_{\blacktriangledown}}(I_{\blacktriangledown}^{\star}(\mathcal{B}, r))$

By A.31 we have

Definition A.64. refined-hyperball-set $\mathcal{B}_{\blacktriangledown}(\mathcal{B}, r) = \mathring{m}^{k^\star_{\blacktriangledown}(\mathcal{B},r)}(\mathcal{B}_0)$

Lemma A.65 (Sub-Hyperballs).

$$\mathbf{b} = B_\epsilon(\boldsymbol{s}),\ \mathbf{b}' = B_{\epsilon'}(\boldsymbol{s}'),\ \epsilon' \leq \frac{\epsilon}{2},\ \boldsymbol{s} \in \mathbf{b}' \implies \mathbf{b}' \subset \mathbf{b}$$

Proof:

1.	$\|\boldsymbol{s} - \boldsymbol{s}'\| \leq \frac{\epsilon}{2}$	(Given)
2.	$\forall \boldsymbol{s}'' \in \mathbf{b}',\ \|\boldsymbol{s}'' - \boldsymbol{s}'\| \leq \frac{\epsilon}{2}$	
3.	$\|\boldsymbol{s} - \boldsymbol{s}'\| + \|\boldsymbol{s}'' - \boldsymbol{s}'\| \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$	
4.	$\|\boldsymbol{s}'' - \boldsymbol{s}\| \leq \|\boldsymbol{s} - \boldsymbol{s}'\| + \|\boldsymbol{s}'' - \boldsymbol{s}'\| \leq \epsilon$	(Triangle Ineq.)
5.	$\|\boldsymbol{s}'' - \boldsymbol{s}\| \leq \epsilon$	
6.	$\boldsymbol{s}'' \in \mathbf{b}' \implies \boldsymbol{s}'' \in \mathbf{b}$	
7.	$\mathbf{b}' \subset \mathbf{b}$	

□

Lemma A.66 (Arbitrary Refinement Precision).

$$\forall \boldsymbol{s} \in S, \forall r > 0\ ;\ \boldsymbol{s} \in \bigcup \mathcal{B} \implies \exists \mathbf{b} \in\ \mathcal{B}_{\blacktriangledown}(\mathcal{B}, \tfrac{r}{2}) \text{ s.t. } \boldsymbol{s} \in \mathbf{b}, \mathbf{b} \subset B_r(\boldsymbol{s})$$

Proof:

1.	$\boldsymbol{s} \in \bigcup \mathcal{B}$	(Given)
2.	$\in \bigcup \mathcal{B} \subset \bigcup \mathcal{Z}_{\blacktriangledown}^{i^{\mathcal{B}}_{\blacktriangledown}(\frac{r}{2})}(\mathcal{B})$	(By A.55)
3.	$\exists \mathbf{b} \in \mathcal{Z}_{\blacktriangledown}^{i^{\mathcal{B}}_{\blacktriangledown}(\frac{r}{2})}(\mathcal{B}) : \boldsymbol{s} \in \mathbf{b}$	
4.	$\epsilon(\mathbf{b}) \leq \frac{r}{2}$	(By A.61)
5.	$\mathbf{b} \subset B_r(\boldsymbol{s})$	(By A.65)
6.	$\mathcal{Z}_{\blacktriangledown}^{i^{\mathcal{B}}_{\blacktriangledown}(\frac{r}{2})}(\mathcal{B}) \subset m_{\blacktriangledown}^{[I^\star_{\blacktriangledown}(\mathcal{B},\frac{r}{2})]}(\mathcal{B}_0) = \mathring{m}^{k^\star_{\blacktriangledown}(\mathcal{B},\frac{r}{2})}(\mathcal{B}_0) = \mathcal{B}_{\blacktriangledown}(\mathcal{B}, \frac{r}{2})$	
7.	$\mathbf{b} \in \mathcal{B}_{\blacktriangledown}(\mathcal{B}, \frac{r}{2})$	

□

Lemma A.67 (Path Containment).

$$\forall \mathfrak{s} \in \mathfrak{S}^*_S, \forall r > 0\ ;\ \mathfrak{s} \in \mathcal{B} \implies \exists \mathfrak{b} \in \mathfrak{B}^*_{\mathcal{B}_{\blacktriangledown}(\mathcal{B},\frac{r}{2})} \text{ s.t. } \mathfrak{s} \in [\![\mathfrak{b}]\!], [\![\mathfrak{b}]\!] \subset \mathcal{N}_r(\mathfrak{s})$$

Proof:

1. $\mathfrak{s} \Subset \mathcal{B} \implies \forall \boldsymbol{s} \in \mathfrak{s}, \boldsymbol{s} \in \bigcup \mathcal{B}$
2. $\implies \forall \boldsymbol{s} \in \mathfrak{s}, \exists \mathbf{b} \in \mathcal{B}_{\blacktriangledown}(\mathcal{B}, \frac{r}{2}) : \boldsymbol{s} \in \mathbf{b}, \mathbf{b} \subset B_r(\boldsymbol{s})$ (By A.66)

Let $\mathbf{b}(\boldsymbol{s})$ be such a hyperball for each $\boldsymbol{s} \in \mathfrak{s}$, and let $\mathfrak{b} = (\mathbf{b}(\boldsymbol{s}))_{\boldsymbol{s} \in \mathfrak{s}}$.

3. $\forall \mathbf{b} \in \mathfrak{b}, \mathbf{b} \in \mathcal{B}_{\blacktriangledown}(\mathcal{B}, \frac{r}{2})$
4. $\mathfrak{b} \in \mathfrak{B}^*_{\mathcal{B}_{\blacktriangledown}(\mathcal{B}, \frac{r}{2})}$
5. $\mathfrak{s} \in [\![\mathfrak{b}]\!]$
6. $[\![\mathfrak{b}]\!] \subset \mathcal{N}_r(\mathfrak{s})$

□

A.4 Density and Countability

Now we can use the preceding definitions and facts to construct a special set of behaviors:

Definition A.68 (Polymutator Behavior Set). The *polymutator behavior set* is

$$\mathfrak{S}^*_{\mathring{m}} = \bigcup_{i \in \mathbb{N}} \mathfrak{S}^*_{\mathcal{S}(\mathring{m}^i(\mathcal{B}_0))}$$

I have analogized this set to the *rational numbers*, and the set of all behaviors, $\mathfrak{S}^*_S$, to the *real numbers*. Let me give a formal definition of a *rational set of behaviors*:

Definition A.69 (Rational Set of Behaviors). A set of behaviors $\mathfrak{S}$ is *rational* if and only if:

1. $|\mathfrak{S}| = \aleph_0$ (The set is countably infinite)
2. $\overline{\mathfrak{S}} = \mathfrak{S}^*_S$ (The closure of the set is equal to the set of all behaviors, that is to say, it is *dense* in the set of all behaviors)

It may be obvious, but for the sake of completeness, we will prove that the polymutator behavior set is a rational set of behaviors. First, we show that it is countably infinite.

Lemma A.70 (Countability of the Polymutator Behavior Set). $|\mathfrak{S}^*_{\mathring{m}}| = \aleph_0$

Proof:

1. $\left|\mathfrak{S}^*_{\mathring{m}}\right| = \left|\bigcup\{\mathfrak{S}^*_{\mathcal{S}(\mathring{m}^i(\mathcal{B}_0))} : i \in \mathbb{N}\}\right|$ (By A.68)

Let $n_i = |\mathcal{S}(\hat{m}^i(\mathcal{B}_0))|$, note that $n_i \in \mathbb{N}$.

2. $\left|\mathfrak{S}^*_{\mathcal{S}(\hat{m}^i(\mathcal{B}_0))}\right| = \sum_{j=2}^{n_i} \frac{n_i!}{(n_i-j)!}$
3. $\in \mathbb{N}$

A union of countably infinitely many countable sets is countably infinite, therefore:

4. $\left|\mathfrak{S}^*_{\hat{\diamond}_{\hat{m}}}\right| = \aleph_0$

□

Now we show that it is dense in $\mathfrak{S}^*_S$. To do this, we have to re-arrange our picture of $\mathfrak{S}^*_{\hat{\diamond}_{\hat{m}}}$, which we have constructed by interleaving going "out" (increasing coverage) with going "in" (increasing resolution). We begin by introducing a function that returns the index after which given plan is guaranteed to be covered by hyperballs of less than a given radius:

Definition A.71 (Hyperball Counting). $\psi(\mathcal{B}_0, \mathfrak{s}, r) = k^\star_\blacktriangledown(\hat{\diamond}^{k^\star_\blacktriangle(\mathfrak{s})}_{\hat{m}}(\mathcal{B}_0), r)$

From which we can define the set of corresponding paths:

Definition A.72 (Sufficient Path Set). $\mathfrak{B}^*_{\psi(\mathcal{B}_0,\mathfrak{s},r)} = \mathfrak{B}^*_{\hat{\diamond}^{\psi(\mathcal{B}_0,\mathfrak{s},r)}_{\hat{m}}(\mathcal{B}_0)}$

Theorem A.73 (Plan Possibility Index). If $\hat{m}$ is a *fair polymutator* function, and $m_\blacktriangle$ and $m_\blacktriangledown$ are *balanced mutator* functions, then $\forall \mathcal{B}_0 \in \mathbb{P}(\mathbb{O}_S), \forall \mathfrak{s} \in \mathfrak{S}^*_S, \forall r > 0$:

$$\exists \mathfrak{b} \in \mathfrak{B}^*_{\psi(\mathcal{B}_0,\mathfrak{s},r)} : \mathfrak{s} \in [\![\mathfrak{b}]\!], [\![\mathfrak{b}]\!] \subset \mathcal{N}_r(\mathfrak{s}), \psi(\mathcal{B}_0, \mathfrak{s}, r) \in \mathbb{N}$$

Proof:

1. $\mathfrak{s} \in \hat{\diamond}^{k^\star_\blacktriangle(\mathfrak{s})}_{\hat{m}}(\mathcal{B}_0)$ (By A.52)
2. $\exists \mathfrak{b} \in \mathfrak{B}^*_{\psi(\mathcal{B}_0,\mathfrak{s},r)} : \mathfrak{s} \in [\![\mathfrak{b}]\!], [\![\mathfrak{b}]\!] \subset \mathcal{N}_r(\mathfrak{s})$ (By A.67)
3. $\psi(\mathcal{B}_0, \mathfrak{s}, r) \in \mathbb{N}$ (By A.14, A.9, A.17, A.63)

□

This simply tells us at what point the set of paths contains a path that "hugs" a plan arbitrarily tightly. Framed in terms of paths, it isn't very helpful to us, but for every path $\mathfrak{b}$ there is a corresponding plan $\mathfrak{s}_\mathfrak{b}$, which is the plan of corresponding waypoints. We denote this set as $\mathfrak{S}^*_{\psi(\mathcal{B}_0,\mathfrak{s},r)} = \{\mathfrak{s}_\mathfrak{b} : \mathfrak{b} \in \mathfrak{B}^*_{\psi(\mathcal{B}_0,\mathfrak{s},r)}\}$. Using it, we can show that $\mathfrak{S}^*_{\hat{\diamond}_{\hat{m}}}$ is dense in $\mathfrak{S}^*_S$. To do this, we construct (a subset of) $\mathfrak{S}^*_{\hat{\diamond}_{\hat{m}}}$ in terms of "slices" of behavior space that have the same resolution, by introducing the following definitions:

Definition A.74 (Behavior Resolution).

$$\Omega(\mathfrak{s}, r) = \mathfrak{s}' \text{ such that } \mathfrak{s}' \in \mathfrak{S}^*_{\psi(\mathcal{B}_0, \mathfrak{s}, r)} : \mathfrak{s} \in \mathcal{N}_{\frac{r}{2}}(\mathfrak{s}'), \mathfrak{s}' \in \mathcal{N}_r(\mathfrak{s})$$

Definition A.75 (Behavior Resolution Slice).

$$\Omega[\mathfrak{S}^*_{\mathring{m}}, r] = \{\Omega(\mathfrak{s}, r) : \mathfrak{s} \in \mathfrak{S}^*_S\}$$

From this definition it should be clear that $\Omega[\mathfrak{S}^*_{\mathring{m}}, r] \subset \mathfrak{S}^*_{\mathring{m}}$. While this set is not obviously enough to reconstruct the entirety of $\mathfrak{S}^*_{\mathring{m}}$, we at least know that $\bigcup_{n \in \mathbb{N}} \Omega[\mathfrak{S}^*_{\mathring{m}}, \frac{1}{n}] \subset \mathfrak{S}^*_{\mathring{m}}$. We can use this set to cleanly show the density of $\mathfrak{S}^*_{\mathring{m}}$.

Lemma A.76 (Density of the Polymutator Behavior Set). $\overline{\mathfrak{S}^*_{\mathring{m}}} = \mathfrak{S}^*_S$

Proof:

1. $\Omega[\mathfrak{S}^*_{\mathring{m}}, r] = \{\Omega(\mathfrak{s}, r) : \mathfrak{s} \in \mathfrak{S}^*_S\}$ (Def. A.75)
2. $\Omega[\mathfrak{S}^*_{\mathring{m}}, r] = \{\Omega(\mathfrak{s}, r) : \mathfrak{s} \in \mathfrak{S}^*_S\} \subset \mathfrak{S}^*_{\mathring{m}} \subset \mathfrak{S}^*_S$
3. $\lim_{r \to 0} \Omega[\mathfrak{S}^*_{\mathring{m}}, r] = \left\{\lim_{r \to 0} \Omega(\mathfrak{s}, r) : \mathfrak{s} \in \mathfrak{S}^*_S\right\} \subset \overline{\mathfrak{S}^*_{\mathring{m}}} \subset \mathfrak{S}^*_S$
4. $= \{\mathfrak{s} : \mathfrak{s} \in \mathfrak{S}^*_S\} \subset \overline{\mathfrak{S}^*_{\mathring{m}}} \subset \mathfrak{S}^*_S$
5. $= \mathfrak{S}^*_S \subset \overline{\mathfrak{S}^*_{\mathring{m}}} \subset \mathfrak{S}^*_S$
6. $= \overline{\mathfrak{S}^*_{\mathring{m}}} = \mathfrak{S}^*_S$ (By def. of $\subset$)

□

This directly gives us our goal:

Theorem A.77 (The Polymutator Behavior Set is Rational). $\mathfrak{S}^*_{\mathring{m}}$ is a rational set of behaviors.

Proof:

1. $|\mathfrak{S}^*_{\mathring{m}}| = \aleph_0$ (By A.70)
2. $\overline{\mathfrak{S}^*_{\mathring{m}}} = \mathfrak{S}^*_S$ (By A.76)

□

Appendix B

Hebbian Learning

B.1 Definitions

1. $\boldsymbol{x} \triangleright \boldsymbol{y} := \frac{\boldsymbol{y}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2}$, the "bind" matrix between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
2. $\langle\!\langle \boldsymbol{x}, \boldsymbol{y} \rangle\!\rangle := \frac{\boldsymbol{x}^*\boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|}$, the cosine-similarity between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
3. $\boldsymbol{a} \odot \boldsymbol{b}$ is the Haddamard or element-wise product of $\boldsymbol{a}$ and $\boldsymbol{b}$

Suppose we have two vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$, and we want to associate $\boldsymbol{y}$ with $\boldsymbol{x}$ via some matrix $\mathbf{W}$ so that $\mathbf{W}\boldsymbol{x} = \boldsymbol{y}$. The matrix $\boldsymbol{x} \triangleright \boldsymbol{y}$ satisfies this requirement. To see this, first observe that:

$$(\boldsymbol{x} \triangleright \boldsymbol{y})\boldsymbol{z} = \frac{\boldsymbol{y}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2}\boldsymbol{z} = \boldsymbol{y}\frac{1}{\|\boldsymbol{x}\|}\frac{\boldsymbol{x}^*\boldsymbol{z}}{\|\boldsymbol{x}\|} = \boldsymbol{y}\frac{\|\boldsymbol{z}\|}{\|\boldsymbol{x}\|}\frac{\boldsymbol{x}^*\boldsymbol{z}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{z}\|} = \boldsymbol{y}\frac{\|\boldsymbol{z}\|}{\|\boldsymbol{x}\|}\langle\!\langle \boldsymbol{x}, \boldsymbol{z} \rangle\!\rangle \tag{B.1}$$

If $\boldsymbol{z} = \boldsymbol{x}$, we have $(\boldsymbol{x} \triangleright \boldsymbol{y})\boldsymbol{x} = \boldsymbol{y}\frac{\|\boldsymbol{x}\|}{\|\boldsymbol{x}\|}\langle\!\langle \boldsymbol{x}, \boldsymbol{x} \rangle\!\rangle = \boldsymbol{y} \cdot 1 \cdot 1 = \boldsymbol{y}$. From Equation B.1 it follows that:

$$[\textstyle\sum_{i=1}^{k} \boldsymbol{x}_i \triangleright \boldsymbol{y}_i]\boldsymbol{z} = \sum_{i=1}^{k}(\boldsymbol{x}_i \triangleright \boldsymbol{y}_i)\boldsymbol{z} = \sum_{i=1}^{k} \boldsymbol{y}_i \frac{\|\boldsymbol{z}\|}{\|\boldsymbol{x}_i\|}\langle\!\langle \boldsymbol{x}_i, \boldsymbol{z} \rangle\!\rangle \tag{B.2}$$

If all of $\boldsymbol{x}_i$ are mutually orthogonal ($j \neq \ell \implies \langle\!\langle \boldsymbol{x}_j, \boldsymbol{x}_\ell \rangle\!\rangle = 0$), then $[\sum_{i=1}^{k} \boldsymbol{x}_i \triangleright \boldsymbol{y}_i]\boldsymbol{x}_j = \boldsymbol{y}_j$ for all $j \in [1 \,..\, k]$. In this sense, $\mathbf{W} = \sum_{i=1}^{k} \boldsymbol{x}_i \triangleright \boldsymbol{y}_i$ acts like a dictionary, with $\boldsymbol{x}_i$ as "keys" and $\boldsymbol{y}_i$ as "values". From Equation B.2 we can see that the cosine-similarity of key-vectors controls the amount of interference between stored values.

B.2 Algebraic Properties

Left distribution over addition:

$$\begin{aligned} \boldsymbol{x} \triangleright (\boldsymbol{y} + \boldsymbol{z}) &= \frac{(\boldsymbol{y}+\boldsymbol{z})\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2} = \frac{\boldsymbol{y}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2} + \frac{\boldsymbol{z}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2} \\ &= (\boldsymbol{x} \triangleright \boldsymbol{y}) + (\boldsymbol{x} \triangleright \boldsymbol{z}) \end{aligned} \tag{B.3}$$

Right distribution over addition:

$$\begin{aligned}(\boldsymbol{x}+\boldsymbol{y}) \triangleright \boldsymbol{z} &= \frac{\boldsymbol{z}(\boldsymbol{x}+\boldsymbol{y})^*}{\|\boldsymbol{x}+\boldsymbol{y}\|^2} = \frac{\boldsymbol{z}\boldsymbol{x}^*}{\|\boldsymbol{x}+\boldsymbol{y}\|^2} + \frac{\boldsymbol{z}\boldsymbol{y}^*}{\|\boldsymbol{x}+\boldsymbol{y}\|^2} = \frac{\|\boldsymbol{x}\|^2}{\|\boldsymbol{x}+\boldsymbol{y}\|^2}\frac{\boldsymbol{z}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2} + \frac{\|\boldsymbol{y}\|^2}{\|\boldsymbol{x}+\boldsymbol{y}\|^2}\frac{\boldsymbol{z}\boldsymbol{y}^*}{\|\boldsymbol{y}\|^2} \\ &= \frac{\|\boldsymbol{x}\|^2}{\|\boldsymbol{x}+\boldsymbol{y}\|^2}(\boldsymbol{x} \triangleright \boldsymbol{z}) + \frac{\|\boldsymbol{y}\|^2}{\|\boldsymbol{x}+\boldsymbol{y}\|^2}(\boldsymbol{y} \triangleright \boldsymbol{z}) \end{aligned} \tag{B.4}$$

Left and right distribution over addition:

$$\begin{aligned}(\boldsymbol{w}+\boldsymbol{x}) \triangleright (\boldsymbol{y}+\boldsymbol{z}) &= [(\boldsymbol{w}+\boldsymbol{x}) \triangleright \boldsymbol{y}] + [(\boldsymbol{w}+\boldsymbol{x}) \triangleright \boldsymbol{z}] \\ &= \frac{\|\boldsymbol{w}\|^2}{\|\boldsymbol{w}+\boldsymbol{x}\|^2}(\boldsymbol{w} \triangleright \boldsymbol{y}) + \frac{\|\boldsymbol{x}\|^2}{\|\boldsymbol{w}+\boldsymbol{x}\|^2}(\boldsymbol{x} \triangleright \boldsymbol{y}) + \frac{\|\boldsymbol{w}\|^2}{\|\boldsymbol{w}+\boldsymbol{x}\|^2}(\boldsymbol{w} \triangleright \boldsymbol{z}) + \frac{\|\boldsymbol{x}\|^2}{\|\boldsymbol{w}+\boldsymbol{x}\|^2}(\boldsymbol{x} \triangleright \boldsymbol{z}) \\ &= \frac{\|\boldsymbol{w}\|^2}{\|\boldsymbol{w}+\boldsymbol{x}\|}[\boldsymbol{w} \triangleright (\boldsymbol{y}+\boldsymbol{z})] + \frac{\|\boldsymbol{x}\|^2}{\|\boldsymbol{w}+\boldsymbol{x}\|^2}[\boldsymbol{x} \triangleright (\boldsymbol{y}+\boldsymbol{z})] \end{aligned} \tag{B.5}$$

Left distribution over element-wise multiplication:

$$\begin{aligned}\boldsymbol{x} \triangleright (\boldsymbol{y} \odot \boldsymbol{z}) &= \frac{(\boldsymbol{y}\odot\boldsymbol{z})\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2} = \frac{\boldsymbol{y}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2} \odot \frac{\boldsymbol{z}1^*}{\|1\|^2}\|1\|^2 \\ &= \|1\|^2 (\boldsymbol{x} \triangleright \boldsymbol{y}) \odot (1 \triangleright \boldsymbol{z}) = \|1\|^2 (1 \triangleright \boldsymbol{y}) \odot (\boldsymbol{x} \triangleright \boldsymbol{z}) \end{aligned} \tag{B.6}$$

Right distribution over element-wise multiplication:

$$\begin{aligned}(\boldsymbol{x} \odot \boldsymbol{y}) \triangleright \boldsymbol{z} &= \frac{\boldsymbol{z}(\boldsymbol{x}\odot\boldsymbol{y})^*}{\|\boldsymbol{x}\odot\boldsymbol{y}\|^2} = \frac{\boldsymbol{z}(\boldsymbol{x}^*\odot\boldsymbol{y}^*)}{\|\boldsymbol{x}\odot\boldsymbol{y}\|^2} = \frac{\boldsymbol{z}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2}\frac{\|\boldsymbol{x}\|^2}{\|\boldsymbol{x}\odot\boldsymbol{y}\|^2} \odot \frac{1\boldsymbol{y}^*}{\|\boldsymbol{y}\|^2}\|\boldsymbol{y}\|^2 \\ &= \frac{\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2}{\|\boldsymbol{x}\odot\boldsymbol{y}\|^2}(\boldsymbol{x} \triangleright \boldsymbol{z}) \odot (\boldsymbol{y} \triangleright 1) = \frac{\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2}{\|\boldsymbol{x}\odot\boldsymbol{y}\|^2}(\boldsymbol{x} \triangleright 1) \odot (\boldsymbol{y} \triangleright \boldsymbol{z}) \end{aligned} \tag{B.7}$$

Appendix C

Harmonic Relational Keys (HaRKs)

C.1 Definitions

Definitions:

1. $\boldsymbol{x} \triangleright \boldsymbol{y} := \frac{\boldsymbol{y}\boldsymbol{x}^*}{\|\boldsymbol{x}\|^2}$, the "bind" matrix between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
2. $\langle\!\langle \boldsymbol{x}, \boldsymbol{y} \rangle\!\rangle := \frac{\boldsymbol{x}^*\boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|}$, the cosine-similarity between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
3. $\boldsymbol{a} \odot \boldsymbol{b}$ is the Haddamard or element-wise product of $\boldsymbol{a}$ and $\boldsymbol{b}$
4. $\boldsymbol{a} \oslash \boldsymbol{b}$ is the Haddamard or element-wise quotient of $\boldsymbol{a}$ and $\boldsymbol{b}$
5. $\mathcal{C} := \{z \in \mathbb{C} : |z| = 1\}$, the set of complex numbers with magnitude 1

Now, our goal is to construct a method of generating spatiotemporally-specific "key" vectors. Consider the function:

$$\boldsymbol{x}_{\boldsymbol{\delta}} = f_{\text{HaRK}}(\boldsymbol{x}, \boldsymbol{\delta}) = \boldsymbol{x} \odot e^{2\pi i \boldsymbol{\Gamma}\boldsymbol{\delta}} \tag{C.1}$$

where $\boldsymbol{\delta} \in \mathbb{R}^d$, $\boldsymbol{\Gamma} \in \mathbb{R}^{N \times d}$, and $\boldsymbol{x}, \boldsymbol{x}_{\boldsymbol{\delta}} \in \mathcal{C}^N$ (see above definition). In the future, we let $\boldsymbol{\gamma}_j^\top := \boldsymbol{\Gamma}_{j,}$. For our purposes, it can be helpful to think of $\boldsymbol{\delta}$ as existing in some low ($d \leq 100$) dimensional "physical" space. Then $\boldsymbol{\Gamma}$ represents a matrix of N frequencies oriented in d-dimensional space, where N is quite large (≥ 10000). It is easy to show that $f_{\text{HaRK}}(\boldsymbol{x}, 0) = \boldsymbol{x}$, and that $f_{\text{HaRK}}(f_{\text{HaRK}}(\boldsymbol{x}, \boldsymbol{\delta}_1), \boldsymbol{\delta}_2) = f_{\text{HaRK}}(\boldsymbol{x}, \boldsymbol{\delta}_1 + \boldsymbol{\delta}_2)$, meaning the structure of addition in $\mathbb{R}^d$ is preserved by f. This means that $\boldsymbol{x}_{\boldsymbol{\delta}}$ implicitly encodes $\boldsymbol{\delta}$, at least relative to some "origin" vector $\boldsymbol{x}_0$. Projecting from $\mathbb{R}^d$ to $\mathcal{C}^N$ in this way grants us (1) a simple model for grid cells, (2) many more than d-orthogonal keys, (3) uniform magnitude of key-vectors, and (4) a relative (rather than absolute) coordinate system, which is cognitively attractive.

C.2 Hebbian Learning with QPKM

Now, we investigate the consequences of using key-vectors generated by f when the keys are element-wise multiplied by some "modulatory" vectors $\boldsymbol{\mu}$ and $\boldsymbol{\eta}$, with scalar multipliers α and β. Consider when $\mathbf{W} = (\alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}) \triangleright \boldsymbol{y}$ and we query $\mathbf{W}$ with $(\beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}})$:

$$\begin{aligned}
\mathbf{W}(\beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}}) &= [(\alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}) \triangleright \boldsymbol{y}](\beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}}) \\
&= \boldsymbol{y}\frac{\|\beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}}\|}{\|\alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}\|}\langle\!\langle \alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}, \beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}} \rangle\!\rangle \\
&= \boldsymbol{y}\frac{\beta}{\alpha}\frac{\|\boldsymbol{\eta}\|}{\|\boldsymbol{\mu}\|}\frac{\langle \alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}, \beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}} \rangle}{\|\alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}\| \cdot \|\beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}}\|} \\
&= \boldsymbol{y}\frac{\beta}{\alpha}\frac{\|\boldsymbol{\eta}\|}{\|\boldsymbol{\mu}\|}\frac{\langle \boldsymbol{\mu} \odot \boldsymbol{x}, \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}} \rangle}{\|\boldsymbol{\mu}\| \cdot \|\boldsymbol{\eta}\|} \\
&= \boldsymbol{y}\frac{\beta}{\alpha}\frac{1}{\|\boldsymbol{\mu}\|^2}\langle \boldsymbol{\mu} \odot \boldsymbol{x}, \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}} \rangle \\
&= \boldsymbol{y}\frac{\beta}{\alpha}\frac{1}{\|\boldsymbol{\mu}\|^2}\langle \boldsymbol{\mu} \odot \boldsymbol{x}, \boldsymbol{\eta} \odot \boldsymbol{x} \odot e^{2\pi i \mathbf{\Gamma}\boldsymbol{\delta}} \rangle \\
&= \boldsymbol{y}\frac{\beta}{\alpha}\frac{1}{\|\boldsymbol{\mu}\|^2}\sum_{j=1}^{N}\overline{(\mu_j x_j)}(\eta_j x_j e^{2\pi i \mathbf{\Gamma}_{j,}\boldsymbol{\delta}}) \\
&= \boldsymbol{y}\frac{\beta}{\alpha}\frac{1}{\|\boldsymbol{\mu}\|^2}\sum_{j=1}^{N}\bar{\mu}_j \eta_j \overline{x}_j x_j e^{2\pi i \boldsymbol{\gamma}_j^\top \boldsymbol{\delta}} \\
&= \boldsymbol{y}\frac{\beta}{\alpha}\frac{1}{\|\boldsymbol{\mu}\|^2}\sum_{j=1}^{N}\bar{\mu}_j \eta_j e^{2\pi i \boldsymbol{\gamma}_j^\top \boldsymbol{\delta}}
\end{aligned} \tag{C.2}$$

Let $c(\boldsymbol{\delta}) = \frac{\beta}{\alpha}\frac{1}{\|\boldsymbol{\mu}\|^2}\sum_{j=1}^{N}\bar{\mu}_j \eta_j e^{2\pi i \boldsymbol{\gamma}_j^\top \boldsymbol{\delta}}$ be our "interference" function, so that we have:

$$[(\alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}) \triangleright \boldsymbol{y}](\beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}}) = c(\boldsymbol{\delta}) \cdot \boldsymbol{y} \tag{C.3}$$

Why do we interpret $c(\boldsymbol{\delta})$ as an "interference" function? Looking at the fourth line of the derivation in Equation C.2, we see that $c(\boldsymbol{\delta}) = \frac{\beta}{\alpha}\frac{\|\boldsymbol{\eta}\|}{\|\boldsymbol{\mu}\|}\langle\!\langle \boldsymbol{\mu} \odot \boldsymbol{x}, \boldsymbol{\eta} \odot \boldsymbol{x}_{\boldsymbol{\delta}} \rangle\!\rangle$, which is just a scaled cosine-similarity between $\boldsymbol{x}$ and $\boldsymbol{x}_{\boldsymbol{\delta}}$ (modulated by $\boldsymbol{\eta}$ and $\boldsymbol{\mu}$, respectively). The "shape" of $c(\boldsymbol{\delta})$ controls the "amount" of $\boldsymbol{y}$ that is retrieved by using the query vector $\boldsymbol{x}_{\boldsymbol{\delta}}$. In general, we will want $c(0) = 1$ so that when we query with $\boldsymbol{x}$ (the original storage vector), we get back $\boldsymbol{y}$, and additionally, as $\|\boldsymbol{\delta}\| \to \infty$, we would like $c(\boldsymbol{\delta}) = 0$, so that values stores "far-away" from each other have minimal interference (though there are other possibilities). By carefully choosing α, $\boldsymbol{\mu}$, β, $\boldsymbol{\eta}$, and $\mathbf{\Gamma}$, we can control the shape of $c(\boldsymbol{\delta})$. By doing this, will be able to approximately control the amount of interference between keys as a function of distance $\|\boldsymbol{\delta}\|$ between them.

First, we want to enforce that $c(0) = 1$, so that we have $[(\alpha \cdot \boldsymbol{\mu} \odot \boldsymbol{x}) \triangleright \boldsymbol{y}](\beta \cdot \boldsymbol{\eta} \odot \boldsymbol{x}_0) =$

$\boldsymbol{y}$.

$$c(0) = \frac{\beta}{\alpha}\frac{1}{\|\boldsymbol{\mu}\|^2}\sum_{j=1}^{N}\bar{\mu}_j\eta_j e^{2\pi i\boldsymbol{\gamma}_j^\top 0} = \frac{\beta}{\alpha}\frac{\sum_{j=1}^{N}\bar{\mu}_j\eta_j}{\|\boldsymbol{\mu}\|^2} = 1 \implies \frac{\alpha}{\beta} = \frac{\sum_{j=1}^{N}\bar{\mu}_j\eta_j}{\|\boldsymbol{\mu}\|^2} \tag{C.4}$$

For the sake of simplicity, we split the problem into two situations: modulation on retrieval, or modulation on storage.

If we only perform modulation on retrieval, then we set $\alpha = 1$ and $\boldsymbol{\mu} = 1$, yielding:

$$\frac{1}{\beta} = \frac{\sum_{j=1}^{N}\eta_j}{\|1\|^2} = \frac{\sum_{j=1}^{N}\eta_j}{N} \implies \beta = \frac{N}{\sum_{j=1}^{N}\eta_j} \implies c(\boldsymbol{\delta}) = \frac{1}{\sum_{j=1}^{N}\eta_j}\sum_{j=1}^{N}\eta_j e^{2\pi i\boldsymbol{\gamma}_j^\top\boldsymbol{\delta}}$$

If we only perform modulation on storage, then we set $\beta = 1$ and $\boldsymbol{\eta} = 1$, yielding:

$$\alpha = \frac{\sum_{j=1}^{N}\bar{\mu}_j}{\|\boldsymbol{\mu}\|^2} \implies c(\boldsymbol{\delta}) = \frac{\|\boldsymbol{\mu}\|^2}{\sum_{j=1}^{N}\bar{\mu}_j}\frac{1}{\|\boldsymbol{\mu}\|^2}\sum_{j=1}^{N}\bar{\mu}_j e^{2\pi i\boldsymbol{\gamma}_j^\top\boldsymbol{\delta}} = \frac{1}{\sum_{j=1}^{N}\bar{\mu}_j}\sum_{j=1}^{N}\bar{\mu}_j e^{2\pi i\boldsymbol{\gamma}_j^\top\boldsymbol{\delta}}$$

The key thing to notice is that either way, $c(\boldsymbol{\delta}) = \frac{1}{\sum_{j=1}^{N}g_j}\sum_{j=1}^{N}g_j e^{2\pi i\boldsymbol{\gamma}_j^\top\boldsymbol{\delta}}$, which happens to have the form of a Fourier decomposition. There is a caveat, however, which is that the Fourier decomposition implicitly assumes a uniform "density" of frequencies $\boldsymbol{\gamma}_j$: for both biological and representational reasons, we prefer to use a non-uniform density of frequencies.

C.3 Derivation of g_j

We will use the following statement, an expression of the "law of the unconscious statistician":

$$\begin{aligned}\left[\lim_{k\to\infty}\frac{1}{k}\sum_{j=1}^{k}h(x_j)\right]_{p_x(x_j)} &= \left[\lim_{k\to\infty}\frac{1}{k}\sum_{j=1}^{k}h(x_j), x_j \sim p_x(x)\right] \\ &= E_{p_x(x)}[h(x)] \\ &= \int_\Omega h(x)p_x(x)dx\end{aligned} \tag{C.5}$$

As well as these definitions for the Fourier and inverse Fourier transform.

$$\hat{f}(\boldsymbol{\xi}) = \mathcal{F}_{\boldsymbol{x}}[f(\boldsymbol{x})](\boldsymbol{\xi}) = \int_{\mathbb{R}^d} f(\boldsymbol{x})e^{-2\pi i\boldsymbol{x}^\top\boldsymbol{\xi}}d\boldsymbol{x} \qquad \text{(the Fourier transform)}$$

$$f(\boldsymbol{x}) = \mathcal{F}_{\boldsymbol{\xi}}^{-1}[\hat{f}(\boldsymbol{\xi})](\boldsymbol{x}) = \int_{\mathbb{R}^d} \hat{f}(\boldsymbol{\xi})e^{2\pi i\boldsymbol{\xi}^\top\boldsymbol{x}}d\boldsymbol{\xi} \qquad \text{(the inverse Fourier transform)}$$

Suppose that we distribute $\boldsymbol{\gamma}_j$ according to $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$. We will make g_j be a function of $\boldsymbol{\gamma}_j$, so $g_j = g(\boldsymbol{\gamma}_j)$. This means that $c(\boldsymbol{\delta}) = \frac{1}{\sum_{j=1}^{N}g(\boldsymbol{\gamma}_j)}\sum_{j=1}^{N}g(\boldsymbol{\gamma}_j)e^{2\pi i\boldsymbol{\gamma}_j^\top\boldsymbol{\delta}}, \boldsymbol{\gamma}_j \sim p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$

Now, consider $c_\infty(\boldsymbol{\delta}) = \lim_{N\to\infty} c(\boldsymbol{\delta})$, the function we want to approximate with $c(\boldsymbol{\delta})$:

$$c_\infty(\boldsymbol{\delta}) = \lim_{N\to\infty} c(\boldsymbol{\delta}) = \left[\lim_{N\to\infty} \frac{1}{\sum_{j=1}^N g(\boldsymbol{\gamma}_j)} \sum_{j=1}^N g(\boldsymbol{\gamma}_j) e^{2\pi i \boldsymbol{\gamma}_j^\top \boldsymbol{\delta}}\right]_{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}_j)}$$

$$c_\infty(\boldsymbol{\delta}) = \left[\lim_{N\to\infty} \frac{N}{\sum_{j=1}^N g(\boldsymbol{\gamma}_j)} \frac{1}{N} \sum_{j=1}^N g(\boldsymbol{\gamma}_j) e^{2\pi i \boldsymbol{\gamma}_j^\top \boldsymbol{\delta}}\right]_{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}_j)}$$

$$= \left[\lim_{N\to\infty} \frac{N}{\sum_{j=1}^N g(\boldsymbol{\gamma}_j)}\right]_{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}_j)} \cdot \left[\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^N g(\boldsymbol{\gamma}_j) e^{2\pi i \boldsymbol{\gamma}_j^\top \boldsymbol{\delta}}\right]_{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}_j)}$$

We let $C = \left[\lim_{N\to\infty} \frac{N}{\sum_{j=1}^N g(\boldsymbol{\gamma}_j))}\right]_{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}_j)}$ and substitute it in...

$$= C \cdot \left[\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^N g(\boldsymbol{\gamma}_j) e^{2\pi i \boldsymbol{\gamma}_j^\top \boldsymbol{\delta}}\right]_{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}_j)}$$

We apply the "law of the unconscious statistician" (Equation C.5)...

$$= C \cdot \int_{\mathbb{R}^d} g(\boldsymbol{\gamma}) e^{2\pi i \boldsymbol{\gamma}^\top \boldsymbol{\delta}} p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) d\boldsymbol{\gamma} = C \cdot \int_{\mathbb{R}^d} [g(\boldsymbol{\gamma}) p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})] e^{2\pi i \boldsymbol{\gamma}^\top \boldsymbol{\delta}} d\boldsymbol{\gamma}$$

Finding the definition of the *inverse* Fourier transform, we substitute it in...

$$c_\infty(\boldsymbol{\delta}) = C \cdot \mathcal{F}_{\boldsymbol{\gamma}}^{-1}[g(\boldsymbol{\gamma}) p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})](\boldsymbol{\delta})$$

We divide each side by C and take the Fourier transform of both sides...

$$\tfrac{1}{C} \mathcal{F}_{\boldsymbol{\delta}}[c_\infty(\boldsymbol{\delta})](\boldsymbol{\gamma}) = g(\boldsymbol{\gamma}) p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$$

Letting $\hat{c}_\infty(\boldsymbol{\gamma}) = \mathcal{F}_{\boldsymbol{\delta}}[c_\infty(\boldsymbol{\delta})](\boldsymbol{\gamma})$ be the Fourier transform of $c_\infty(\boldsymbol{\delta})$...

$$\frac{1}{C} \cdot \frac{\hat{c}_\infty(\boldsymbol{\gamma})}{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})} = g(\boldsymbol{\gamma})$$

It turns out C is a free parameter which will be normalized-out anyway, so we drop it, yielding...

$$g(\boldsymbol{\gamma}) = \frac{\hat{c}_\infty(\boldsymbol{\gamma})}{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})} \tag{C.6}$$

So, the modulation "gains" $g(\boldsymbol{\gamma}_j)$ just have to be the Fourier transform of the target cosine-similarity function evaluated at $\boldsymbol{\gamma}_j$ divided by the density of $\boldsymbol{\gamma}_j$, determined by our choice of $\boldsymbol{\Gamma}$.

C.4 Derivation of $c(\boldsymbol{\delta})$

We are primarily interested in two cases:

1. $c_\sigma(\boldsymbol{\delta}) = e^{-\left\|\frac{\delta}{\sigma}\right\|^2}$, an isometric Gaussian with standard deviation σ
2. $c_{\boldsymbol{\sigma}}(\boldsymbol{\delta}) = e^{-\|\boldsymbol{\delta} \oslash \boldsymbol{\sigma}\|^2}$, a non-isometric diagonal Gaussian with standard-deviations $\boldsymbol{\sigma}$

According to Abramowitz and Stegun (1972, p. 302, equation 7.4.6), the Fourier transform of a Gaussian given by $e^{-x^2\sigma^{-2}}$ is:

$$\mathcal{F}_x\left[e^{-ax^2}\right](\xi) = \pi^{\frac{1}{2}} a^{-\frac{1}{2}} e^{-\pi^2\xi^2 a^{-1}} \tag{C.7}$$

In our case, $a = \sigma^{-2}$, so we substitute and get:

$$\mathcal{F}_x\left[e^{-x^2\sigma^{-2}}\right](\xi) = \pi^{\frac{1}{2}}(\sigma^{-2})^{-\frac{1}{2}} e^{-\pi^2\xi^2(\sigma^{-2})^{-1}} = \sqrt{\pi}\sigma e^{-\pi^2\sigma^2\xi^2} \tag{C.8}$$

What is the Fourier transform for a Gaussian given by $e^{-\|\boldsymbol{x} \oslash \boldsymbol{\sigma}\|^2}$, where $\boldsymbol{\sigma}$ is the standard deviation along each axis? Note that:

$$e^{-\|\boldsymbol{x} \oslash \boldsymbol{\sigma}\|^2} = e^{-\sum_{j=1}^d x_j^2 \sigma_j^{-2}} = \prod_{j=1}^d e^{-x_j^2\sigma_j^{-2}}$$

According to [https://see.stanford.edu/materials/lsoftaee261/chap8.pdf], this means that:

$$\begin{aligned}\mathcal{F}_{\boldsymbol{x}}\left[e^{-\|\boldsymbol{x} \oslash \boldsymbol{\sigma}\|^2}\right](\boldsymbol{\xi}) &= \prod_{j=1}^d \mathcal{F}_x\left[e^{-x_j^2\sigma_j^{-2}}\right](\boldsymbol{\xi}) = \prod_{j=1}^d \sqrt{\pi}\sigma_j e^{-\pi^2\sigma_j^2\xi_j^2} \\ &= \pi^{\frac{d}{2}}\left(\prod_{j=1}^d \sigma_j\right) e^{-\sum_{j=1}^d \pi^2\sigma_j^2\xi_j^2} = \pi^{\frac{d}{2}}\left(\prod_{j=1}^d \sigma_j\right) e^{-\pi^2\|\boldsymbol{\sigma} \odot \boldsymbol{\xi}\|^2}\end{aligned} \tag{C.9}$$

From this, we directly get the expression for the Fourier transform of the non-isometric diagonal Gaussian with standard-deviation vector $\boldsymbol{\sigma}$:

$$\hat{c}_{\boldsymbol{\sigma}}(\boldsymbol{\gamma}) = \pi^{\frac{d}{2}}\left(\prod_{j=1}^d \sigma_j\right) e^{-\pi^2\|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2} \tag{C.10}$$

Notice that $c_{\boldsymbol{\sigma}}(\boldsymbol{\delta}) = c_\sigma(\boldsymbol{\delta})$ when every $\sigma_j = \sigma$. We can use this result to get the Fourier transform of the isometric Gaussian with standard-deviation σ:

$$\hat{c}_\sigma(\boldsymbol{\gamma}) = \pi^{\frac{d}{2}}\sigma^d e^{-\pi^2\sigma^2\|\boldsymbol{\gamma}\|^2} \tag{C.11}$$

C.5 Derivation of $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$

Now we have to address the question of choosing $\boldsymbol{\Gamma}$, which we do by sampling individual $\boldsymbol{\gamma}$ from a chosen distribution, $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$. For the sake of simplicity, we choose $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$ to be symmetric, meaning that $\forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$, $\|\boldsymbol{a}\| = \|\boldsymbol{b}\| \implies p_{\boldsymbol{\gamma}}(\boldsymbol{a}) = p_{\boldsymbol{\gamma}}(\boldsymbol{b})$. This implies that there exists some function $h(r)$ s.t. $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) = h(\|\boldsymbol{\gamma}\|)$. In order to sample

individual $\boldsymbol{\gamma}$, we will actually specify a distribution over frequency *magnitudes* $q_r(\|\boldsymbol{\gamma}\|)$, then assign a spherically uniform orientation to each frequency.

In order to relate $h(\|\boldsymbol{\gamma}\|)$ and $q_r(\|\boldsymbol{\gamma}\|)$, notice that the density of frequency *magnitudes* must incorporate frequencies of all *orientations*. What this means, geometrically, is that the density $h(r)$ of any individual frequency with magnitude r must be equal to to the density of that magnitude, $q_r(r)$, *divided* by the area of the hypersphere with radius r (the area is proportional to r^{d-1}). We can use this to get the exact relationship between $p_{\boldsymbol{\gamma}}$ and $q_r(r)$. Ignoring constants, this is:

$$\begin{aligned} h(\|\boldsymbol{\gamma}\|) &\propto \frac{q_r(\|\boldsymbol{\gamma}\|)}{\|\boldsymbol{\gamma}\|^{d-1}} \\ p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) = h(\|\boldsymbol{\gamma}\|) &\propto q_r(\|\boldsymbol{\gamma}\|) \cdot \|\boldsymbol{\gamma}\|^{1-d} \\ p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) &= \frac{1}{\int_\Omega q(\|\boldsymbol{\gamma}\|) \cdot \|\boldsymbol{\gamma}\|^{1-d} d\boldsymbol{\gamma}} \cdot q_r(\|\boldsymbol{\gamma}\|) \cdot \|\boldsymbol{\gamma}\|^{1-d} \end{aligned}$$

Letting $r = \|\boldsymbol{\gamma}\|$, and $\gamma_{\min}$ and $\gamma_{\max}$ the minimum and maximum of $\|\boldsymbol{\gamma}\|$...

$$\begin{aligned} p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) &= \left[\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})} \int_{\gamma_{\min}}^{\gamma_{\max}} q_r(r) \cdot r^{1-d} \cdot r^{d-1} dr \right]^{-1} \cdot q_r(\|\boldsymbol{\gamma}\|) \cdot \|\boldsymbol{\gamma}\|^{1-d} \\ p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) &= \left[\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})} \int_{\gamma_{\min}}^{\gamma_{\max}} q_r(r) dr \right]^{-1} \cdot q_r(\|\boldsymbol{\gamma}\|) \cdot \|\boldsymbol{\gamma}\|^{1-d} \end{aligned}$$

Letting $Q(r)$ be the antiderivative of $q_r(r)$...

$$\begin{aligned} p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) &= \left[\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})} [Q(\gamma_{\max}) - Q(\gamma_{\min})] \right]^{-1} \cdot q_r(\|\boldsymbol{\gamma}\|) \cdot \|\boldsymbol{\gamma}\|^{1-d} \\ p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) &= \frac{\Gamma(\frac{d}{2})}{2\pi^{\frac{d}{2}}} \cdot \frac{q_r(\|\boldsymbol{\gamma}\|) \cdot \|\boldsymbol{\gamma}\|^{1-d}}{Q(\gamma_{\max}) - Q(\gamma_{\min})} \end{aligned} \tag{C.12}$$

This gives us our analytical density over frequencies $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$ given a chosen density over frequency magnitudes $q_r(\|\boldsymbol{\gamma}\|)$.

C.6 Numerical calculation of $g(\boldsymbol{\gamma})$

It turns out that $q_r(\|\boldsymbol{\gamma}\|) = \|\boldsymbol{\gamma}\|^{-1}$ is a nice choice, giving us some scale-free and dimensionality-invariant properties, as well as some modicum of agreement with the observed distribution of scales in grid-cell responses.

$q_r(\|\boldsymbol{\gamma}\|) = \|\boldsymbol{\gamma}\|^{-1}$ implies the antiderivative of $q_r(\|\boldsymbol{\gamma}\|)$ is $Q(\|\boldsymbol{\gamma}\|) = \ln(\|\boldsymbol{\gamma}\|)$. Recalling from Equation C.6 that $g(\boldsymbol{\gamma}) = \frac{\hat{c}_\infty(\boldsymbol{\gamma})}{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})}$ and from Equation C.10 that $\hat{c}_{\boldsymbol{\sigma}}(\boldsymbol{\gamma}) = \pi^{\frac{d}{2}} \left(\prod_{j=1}^{d} \sigma_j \right) e^{-\pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2}$, we now have all the information we need to determine $g(\boldsymbol{\gamma})$

as a function of $\boldsymbol{\sigma}$:

$$\begin{aligned}
g(\boldsymbol{\gamma}) = \frac{\hat{c}_{\boldsymbol{\sigma}}(\boldsymbol{\gamma})}{p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})} &= \frac{\pi^{\frac{d}{2}} \left(\prod_{j=1}^{d} \sigma_j\right) e^{-\pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2}}{\|\boldsymbol{\gamma}\|^{-1} \cdot \|\boldsymbol{\gamma}\|^{1-d}} \\
&= \|\boldsymbol{\gamma}\|^{d} \pi^{\frac{d}{2}} \left(\prod_{j=1}^{d} \sigma_j\right) e^{-\pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2} \\
&= e^{\ln\left(\|\boldsymbol{\gamma}\|^{d} \pi^{\frac{d}{2}} \left(\prod_{j=1}^{d} \sigma_j\right) e^{-\pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2}\right)} \\
&= e^{\ln(\|\boldsymbol{\gamma}\|^{d}) + \ln(\pi^{\frac{d}{2}}) + \ln\left(\prod_{j=1}^{d} \sigma_j\right) - \pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2} \\
&= e^{d\ln(\|\boldsymbol{\gamma}\|) + \frac{d}{2}\ln(\pi) + \sum_{j=1}^{d} \ln(\sigma_j) - \pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2} \\
&= \left(e^{\frac{d}{2}\ln(\pi) + \sum_{j=1}^{d} \ln(\sigma_j)}\right) \left(e^{d\ln(\|\boldsymbol{\gamma}\|) - \pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2}\right)
\end{aligned}$$

Note that with respect to $\boldsymbol{\gamma}$, the first term is a constant which will be normalized out later by α and β, so we drop it, leaving...

$$= e^{d\ln(\|\boldsymbol{\gamma}\|) - \pi^2 \|\boldsymbol{\sigma} \odot \boldsymbol{\gamma}\|^2} \tag{C.13}$$

Calculation of this term can become numerically unstable on a computer. To correct this, we find the maximum value of the expression, and divide it out, re-arranging so the correction happens inside the exponential. To do this, we first have to realize that the relevant σ is the minimum component of $\boldsymbol{\sigma}$, $\sigma_{\min}$. For clarity, we replace $\|\boldsymbol{\gamma}\|$ with γ, and find where the derivative of Equation C.13 is equal to 0 to find its maximum:

$$\begin{aligned}
0 &= \frac{d}{d\gamma}(e^{d\ln(\|\boldsymbol{\gamma}\|) - \pi^2 \sigma_{\min}^2 \gamma^2}) = \left(\frac{d}{\gamma} - 2\gamma\pi^2\sigma_{\min}^2\right) e^{d\ln(\|\boldsymbol{\gamma}\|) - \pi^2 \sigma_{\min}^2 \gamma^2} \\
0 &= d - 2\gamma^2\pi^2\sigma_{\min}^2 \\
\gamma_{\max} &= \frac{\sqrt{d}}{\sqrt{2}\pi\sigma_{\min}}
\end{aligned}$$

Now we plug this back into Equation C.13...

$$\begin{aligned}
e^{d\ln(\gamma_{\max}) - \pi^2 \sigma_{\min}^2 \gamma_{\max}^2} &= e^{d\ln\left(\frac{\sqrt{d}}{\sqrt{2}\pi\sigma_{\min}}\right) - \pi^2\sigma_{\min}^2\left(\frac{\sqrt{d}}{\sqrt{2}\pi\sigma_{\min}}\right)^2} \\
&= e^{d\ln\left(\frac{\sqrt{d}}{\sqrt{2}\pi\sigma_{\min}}\right) - \pi^2\sigma_{\min}^2 \frac{d}{2\pi^2\sigma_{\min}^2}} \\
&= e^{d\ln\left(\frac{\sqrt{d}}{\sqrt{2}\pi\sigma_{\min}}\right) - \frac{d}{2}} = e^{d\left(\ln\left(\frac{\sqrt{d}}{\sqrt{2}\pi\sigma_{\min}}\right) - \frac{1}{2}\right)} = e^{d\left(\ln\left(\sqrt{\frac{d}{2\pi^2\sigma_{\min}^2}}\right) - \frac{1}{2}\right)} \\
&= e^{d\left(\frac{1}{2}\ln\left(\frac{d}{2\pi^2\sigma_{\min}^2}\right) - \frac{1}{2}\right)} = e^{\frac{d}{2}\left(\ln\left(\frac{d}{2\pi^2\sigma_{\min}^2}\right) - 1\right)}
\end{aligned} \tag{C.14}$$

Now, to ensure numerical stability, we divide Equation C.13 by the term derived in

Equation C.14 so that the maximum value is fixed at 1:

$$g(\boldsymbol{\gamma}) = \frac{e^{d\ln(\|\boldsymbol{\gamma}\|)-\pi^2\|\boldsymbol{\sigma}\odot\boldsymbol{\gamma}\|^2}}{e^{\frac{d}{2}\left(\ln\left(\frac{d}{2\pi^2\sigma_{\min}^2}\right)-1\right)}} = e^{\left[d\ln(\|\boldsymbol{\gamma}\|)-\pi^2\|\boldsymbol{\sigma}\odot\boldsymbol{\gamma}\|^2-\frac{d}{2}\left(\ln\left(\frac{d}{2\pi^2\sigma_{\min}^2}\right)-1\right)\right]} \tag{C.15}$$

This is a numerically stable expression which we can use to compute $g(\boldsymbol{\gamma})$ even for very large d, meaning that we can apply this system to high-dimensional sensory or action spaces, not just low-dimensional "physical" spaces.

C.7 Picking $\gamma_{\min}$ and $\gamma_{\max}$

As Equation C.15 is a Gaussian, there will be values of $\boldsymbol{\gamma}$ which are negligible. We can use Equation C.15 to find the minimum and maximum "useful" $\|\boldsymbol{\gamma}\|$ for a given choice of σ, $\gamma_{\min}^{\sigma}$ and $\gamma_{\max}^{\sigma}$, respectively (we now restrict ourselves to the isometric case for the sake of simplicity). We can use this fact to go from a range of scales, σ, that we would like to represent, to a range of frequency magnitudes $\|\boldsymbol{\gamma}\|$ that our system will need to represent those scales. Note that we are here assuming the isometric Gaussian case. Let ϵ be our "minimum" non-zero gain, then setting ϵ equal to Equation C.6 yields:

$$\begin{aligned}
\epsilon &= e^{\left[d\ln(\gamma)-(\pi\sigma\gamma)^2-\frac{d}{2}\left(\ln\left(\frac{d}{2\pi^2\sigma^2}\right)-1\right)\right]} \\
\epsilon &= e^{\left[d\ln(\gamma)-(\pi\sigma\gamma)^2-\frac{d}{2}\ln\left(\frac{d}{2\pi^2\sigma^2}\right)+\frac{d}{2}\right]} \\
\epsilon &= e^{d\ln(\gamma)}e^{-(\pi\sigma\gamma)^2}e^{-\frac{d}{2}\ln\left(\frac{d}{2\pi^2\sigma^2}\right)}e^{\frac{d}{2}} \\
\epsilon &= e^{\ln(\gamma^d)}e^{-(\pi\sigma\gamma)^2}e^{\ln\left(\left(\frac{2\pi^2\sigma^2}{d}\right)^{\frac{d}{2}}\right)}e^{\frac{d}{2}} \\
\epsilon &= \gamma^d e^{-(\pi\sigma\gamma)^2}\left(\frac{2\pi^2\sigma^2}{d}\right)^{\frac{d}{2}}e^{\frac{d}{2}} \\
\epsilon &= \gamma^d e^{-(\pi\sigma\gamma)^2}\left(\frac{2e\pi^2\sigma^2}{d}\right)^{\frac{d}{2}} \\
\epsilon\left(\frac{d}{2e\pi^2\sigma^2}\right)^{\frac{d}{2}} &= \gamma^d e^{-(\pi\sigma\gamma)^2} \\
\left(\epsilon\left(\frac{d}{2e\pi^2\sigma^2}\right)^{\frac{d}{2}}\right)^{\frac{2}{d}} &= \left(\gamma^d e^{-(\pi\sigma\gamma)^2}\right)^{\frac{2}{d}} \\
\epsilon^{\frac{2}{d}}\frac{d}{2e\pi^2\sigma^2} &= \gamma^2 e^{-\frac{2}{d}\pi^2\sigma^2\gamma^2} \\
-\frac{2}{d}\pi^2\sigma^2\epsilon^{\frac{2}{d}}\frac{d}{2e\pi^2\sigma^2} &= -\frac{2}{d}\pi^2\sigma^2\gamma^2 e^{-\frac{2}{d}\pi^2\sigma^2\gamma^2} \\
-\frac{\epsilon^{\frac{2}{d}}}{e} &= -\frac{2}{d}\pi^2\sigma^2\gamma^2 e^{-\frac{2}{d}\pi^2\sigma^2\gamma^2}
\end{aligned}$$

We can solve for γ using the Lambert W function, the inverse of $f(x) = x \cdot e^x$...

$$\begin{aligned} -\tfrac{2}{d}\pi^2\sigma^2\gamma^2 &= W_k\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right) \\ \gamma^2 &= -\frac{d}{2\pi^2\sigma^2}W_k\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right) \\ \gamma &= \sqrt{-\frac{d}{2\pi^2\sigma^2}W_k\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} \end{aligned} \tag{C.16}$$

It turns out that the minimum and maximum values correspond to the 0 and 1-branches of the Lambert W function:

$$\gamma^\sigma_{\min} = \sqrt{-\frac{d}{2\pi^2\sigma^2}W_0\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} \qquad \gamma^\sigma_{\max} = \sqrt{-\frac{d}{2\pi^2\sigma^2}W_{-1}\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} \tag{C.17}$$

Thus, the frequency "width" is:

$$\begin{aligned} \gamma^\sigma_{\text{width}} &= \gamma^\sigma_{\max} - \gamma^\sigma_{\min} \\ &= \sqrt{-\frac{d}{2\pi^2\sigma^2}W_{-1}\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} - \sqrt{-\frac{d}{2\pi^2\sigma^2}W_0\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} \\ &= \sqrt{\frac{d}{2\pi^2\sigma^2}}\sqrt{-W_{-1}\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} - \sqrt{\frac{d}{2\pi^2\sigma^2}}\sqrt{-W_0\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} \\ &= \sqrt{\frac{d}{2\pi^2\sigma^2}}\left(\sqrt{-W_{-1}\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} - \sqrt{-W_0\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)}\right) \\ &= \frac{\sqrt{d}}{\sqrt{2}\pi\sigma}\left(\sqrt{-W_{-1}\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)} - \sqrt{-W_0\left(-\tfrac{\epsilon^{\frac{2}{d}}}{e}\right)}\right) \end{aligned} \tag{C.18}$$

In general, we want to represent more than one "scale", σ. Assuming we have already picked $\gamma_{\min}$ and $\gamma_{\max}$ to represent a range of scales that includes σ, what fraction of neurons will be more than ϵ-active for the σ-scale? We use the fact that we chose $p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma}) \propto \frac{1}{\|\boldsymbol{\gamma}\|}$ to simplify things, by noting that when $\boldsymbol{\gamma} \sim p_{\boldsymbol{\gamma}}(\boldsymbol{\gamma})$, then $\ln(\|\boldsymbol{\gamma}\|)$ is uniform between $\ln(\gamma_{\min})$ and $\ln(\gamma_{\max})$. This means the probability mass between $\gamma^\sigma_{\min}$ and $\gamma^\sigma_{\max}$

is given by:

$$\begin{aligned}
\mathrm{P}(\gamma_{\min}^{\sigma} < \|\boldsymbol{\gamma}\| < \gamma_{\max}^{\sigma}) &= \frac{\ln(\gamma_{\max}^{\sigma}) - \ln(\gamma_{\min}^{\sigma})}{\ln(\gamma_{\max}) - \ln(\gamma_{\min})} \\
&= \frac{\ln\left(\frac{\gamma_{\max}^{\sigma}}{\gamma_{\min}^{\sigma}}\right)}{\ln\left(\frac{\gamma_{\max}}{\gamma_{\min}}\right)} \\
&= \ln\left(\frac{\gamma_{\max}}{\gamma_{\min}}\right)^{-1} \ln\left(\frac{\gamma_{\max}^{\sigma}}{\gamma_{\min}^{\sigma}}\right) \\
&= \ln\left(\frac{\gamma_{\max}}{\gamma_{\min}}\right)^{-1} \ln\left(\frac{\sqrt{-\frac{d}{2\pi^2\sigma^2} W_{-1}\left(-\frac{\epsilon^{\frac{2}{d}}}{e}\right)}}{\sqrt{-\frac{d}{2\pi^2\sigma^2} W_{0}\left(-\frac{\epsilon^{\frac{2}{d}}}{e}\right)}}\right) \\
&= \frac{1}{2}\ln\left(\frac{\gamma_{\max}}{\gamma_{\min}}\right)^{-1} \ln\left(\frac{W_{-1}\left(-\frac{\epsilon^{\frac{2}{d}}}{e}\right)}{W_{0}\left(-\frac{\epsilon^{\frac{2}{d}}}{e}\right)}\right) \\
&= \frac{1}{2}\ln\left(\frac{\gamma_{\max}}{\gamma_{\min}}\right)^{-1} \left[W_{0}\left(-\frac{\epsilon^{\frac{2}{d}}}{e}\right) - W_{-1}\left(-\frac{\epsilon^{\frac{2}{d}}}{e}\right)\right]
\end{aligned}$$

When $\epsilon = 0.001$, this can be extremely closely approximated by...

$$\approx 5.257 \cdot \ln\left(\frac{\gamma_{\max}}{\gamma_{\min}}\right)^{-1} \left(\frac{\sqrt{d + 1.83}}{d}\right) \tag{C.19}$$

From which it's evident that as d increases, the overall number of neurons that will be "active" at a given σ shrinks. In higher dimensions, a smaller range of scales can be represented using the same number of neurons. Another important point, though, is that our choice of density for $\boldsymbol{\gamma}$ has resulted in our representational power being scale-invariant, in the sense that the number of "active" neurons doesn't depend on σ.

Appendix D

Detailed Methods

We cover some technical details that important for understanding the operation of the ARMS algorithm, as well as its neural implementation.

D.1 Extension and Prior Knowledge

Extension adds new vertices around an existing vertex so as to increase the coverage of $\mathcal{G}$. During extension, the agent randomly samples points $\boldsymbol{p}_j$ around the exterior of the field of the vertex $\mathbf{v}$. If the agent can estimate the reachability r_j or be provided with it, then it can store that information using HaRKs in a matrix $\mathbf{R} = \sum_j \boldsymbol{x}_{\boldsymbol{p}_j} \triangleright r_j$, along with a normalization matrix $\mathbf{C} = \sum_j \boldsymbol{x}_{\boldsymbol{p}_j} \triangleright 1$. If the agent is Naive, then $r_j = 1$ regardless of the actual reachability (we could also call the Naive agent "optimistic").

Then, the agent tries to add new fields. Suppose the size of the field $F(\mathbf{v})$ is σ, then the agent starts by trying to add new fields with a size of $\sigma' = \sqrt{2}\sigma$. It does this by checking the value $\hat{r}_j = \frac{\mathbf{R}(\boldsymbol{x}_{\boldsymbol{p}_j} \odot \boldsymbol{\eta}_{\sigma'})}{\mathbf{C}(\boldsymbol{x}_{\boldsymbol{p}_j} \odot \boldsymbol{\eta}_{\sigma'})}$, where $\hat{r}_j$ is the local "average" of reachability, with "local" defined by the scale of σ'. All $\boldsymbol{p}_j$ are evaluated and ordered from highest to lowest $\hat{r}_j$, and the points with the highest $\hat{r}_j$ (above a certain threshold) are added as new vertex with fields of size σ' (new vertices fields that are too redundant with older vertices are not added).

After doing this check for $\sigma' = \sqrt{2}\sigma$, the agent does this check for $\sigma' = \sigma$ and $\sigma' = \frac{\sqrt{2}}{2}\sigma$. For the last check, it doesn't matter if $\tilde{r}_j$ is above the pre-defined threshold, the space surrounding $F(\mathbf{v})$ is filled with new vertices of field-size $\frac{\sqrt{2}}{2}\sigma$. The "Astute" agent uses a direct collision detection calculation to determine r_j, while the "Misled" agent uses $1 - \hat{r}_j$ instead of $\hat{r}_j$, effectively inverting the judgments of the "Astute" agent.

D.2 ARMS α' Regulation

Internally, information about inhibition dynamics is stored in a tuple

$$\Lambda = (\alpha', \alpha'_0, \Delta^+, \Delta^-, \kappa) \tag{D.1}$$

with three main functions that modify Λ, f_1^Λ, f_2^Λ, and f_3^Λ. Referring back to the ARMS flowchart in Fig. 9.1, there are three points where Λ is updated, those being steps [p] (where f_1^Λ is called on the edge that just failed), [e] (where f_2^Λ is called after a path has been found), and [r] (where f_3^Λ is called on the edge with the highest α that is inhibited after a pathing failure).

Function 1: Stochastic α update.

function $d\alpha'(\alpha',\ \Delta)$
  $n \sim U([0,2])$
  $\alpha' \leftarrow \alpha' + n \cdot \Delta$
  if $\alpha' < 0$ **then**
    $\alpha' \leftarrow 0$
  return α'

Function 2: Traversal-failure response.

function $f_1^\Lambda(\Lambda,\ \mathbf{e})$
  $(\alpha'_0, \alpha', \Delta^+, \Delta^-, \kappa) \leftarrow \Lambda$
  $\Delta\alpha' \leftarrow \alpha(\mathbf{e}) - \alpha'$
  $\Delta^+ \leftarrow (1-\beta_r)\cdot\Delta^+ + \beta_r \cdot \lambda_\tau^+ \cdot \max(0, \Delta\alpha')$
  $\alpha' \leftarrow d\alpha'(\alpha', \Delta^+)$
  $\Lambda \leftarrow (\alpha'_0, \alpha', \Delta^+, \Delta^-, \kappa)$
  return Λ

Function 3: Pathing-success response.

function $f_2^\Lambda(\Lambda)$
  $(\alpha'_0, \alpha', \Delta^+, \Delta^-, \kappa) \leftarrow \Lambda$
  if $\kappa > 0$ **then**
    $\Delta\alpha' \leftarrow \alpha'_0 - \alpha'$
    $\Delta^- \leftarrow (1-\beta_r)\Delta^- + \beta_r \cdot \lambda_\tau^- \cdot \Delta\alpha'$
    $\kappa \leftarrow 0$
  $\Lambda \leftarrow (\alpha'_0, \alpha', \Delta^+, \Delta^-, \kappa)$
  return Λ

Function 4: Pathing-failure response.

function $f_3^\Lambda(\Lambda,\ \mathbf{e})$
  $(\alpha'_0, \alpha', \Delta^+, \Delta^-, \kappa) \leftarrow \Lambda$
  if $\kappa = 0$ **then**
    $\alpha'_0 \leftarrow \alpha'$
  $\kappa \leftarrow \kappa + 1$
  $\alpha' \leftarrow \min(\alpha', \alpha(\mathbf{e}))$
  $\alpha' \leftarrow d\alpha'(\alpha', -\Delta^-)$
  $\Lambda \leftarrow (\alpha'_0, \alpha', \Delta^+, \Delta^-, \kappa)$
  return Λ

D.3 Derivation of Δq

While "refinement" is an operation that occurs to an individual *vertex*, the relevant information about when to refine a vertex actually exists on an *edge*, as we are ultimately interested in increasing the resolution of the corresponding $\boldsymbol{\epsilon}$-tubes. Henceforth, when we refer to "refinement", we will be talking about an operation that occurs *on an edge*, with the decision to "refine an edge" resulting in the refinement of *one of its vertices*. To understand when our agent should refine an edge, we need to understand what exactly the agent gains through refinement. Technically, the overall utility of an edge is related to how helpful the edge is for getting the agent to the rest of the environment... a global measure that depends on all the paths that pass through that edge, and their reliability. We can quantify this as:

$$q(\mathbf{e}) = \sum_{\mathfrak{p} \in \mathfrak{P}_{\mathcal{G}}(\mathbf{e})} p(\mathfrak{p}) \cdot \alpha(\mathfrak{p}) \tag{D.2}$$

where $\mathfrak{P}_{\mathcal{G}}(\mathbf{e})$ is the set of all paths containing $\mathbf{e}$, $p(\mathfrak{p})$ is the probability that $\mathfrak{p}$ is sampled, and $\alpha(\mathfrak{p})$ is a generalization of potential affordance to paths, with $\alpha(\mathfrak{p}) = r_{K}(\mathfrak{p}) \cdot M(\mathfrak{p}[0]) \cdot M(\mathfrak{p}[\text{-}1])$. To make $q(\mathbf{e})$ tractable to calculate, we have to make some assumptions. First, we assume that $p(\mathfrak{p}) \propto r_{K}(\mathbf{e})$, with an unknown proportionality constant c_p. Second, we assume that $r_{K}(\mathfrak{p}) \propto r_{K}(\mathbf{e})$, with a proportionality constant of c_r. Third, we assume that $M(\mathfrak{p}[0])$ and $M(\mathfrak{p}[\text{-}1])$ are independent of each other and of $\mathbf{e}$, so we factor them out into their average across the whole graph, denoted c_F. Fourth, we assume that the number of paths including $\mathbf{e}$ is proportional to the number of traversals over that edge, $n(\mathbf{e})$, giving us:

$$\begin{aligned} q(\mathbf{e}) &= \sum_{\mathfrak{p} \in \mathfrak{P}_{\mathcal{G}}(\mathbf{e})} p(\mathfrak{p}) \cdot \alpha(\mathfrak{p}) = \sum_{\mathfrak{p} \in \mathfrak{P}_{\mathcal{G}}(\mathbf{e})} c_p r_{K}(\mathbf{e}) \cdot r_{K}(\mathfrak{p}) \cdot M(\mathfrak{p}[0]) \cdot M(\mathfrak{p}[\text{-}1]) \\ &= \sum_{\mathfrak{p} \in \mathfrak{P}_{\mathcal{G}}(\mathbf{e})} c_p r_{K}(\mathbf{e}) \cdot c_r r_{K}(\mathbf{e}) \cdot c_F = c_p c_r c_F \cdot \sum_{\mathfrak{p} \in \mathfrak{P}_{\mathcal{G}}(\mathbf{e})} r_{K}(\mathbf{e})^2 = c_q \cdot n(\mathbf{e}) \cdot r_{K}(\mathbf{e})^2 \end{aligned} \tag{D.3}$$

where $c_q = c_p c_r c_F$. Assuming that refining the edge will split it into k smaller edges $\{\mathbf{e}'_1, ...\mathbf{e}'_k\}$ with disjoint fields, the total "utility" of these refined edges would be:

$$q_{\text{ref}}(\mathbf{e}) = \sum_{i=1}^{k} q(\mathbf{e}'_i) = \sum_{i=1}^{k} c_q n(\mathbf{e}'_i) r_{K}(\mathbf{e}'_i)^2 \tag{D.4}$$

We assume that each "child"-edge $\mathbf{e}'_i$ is in approximately $\frac{1}{k}$th as many paths as $\mathbf{e}$, and that $r_{K}(\mathbf{e}'_i) \sim \text{B}(c \cdot r_{K}(\mathbf{e}), c \cdot (1 - r_{K}(\mathbf{e})))$, the beta distribution re-parameterized with $r_{K}(\mathbf{e})$ the average of the distribution, c a parameter encoding the intrinsic properties of the statespace. Taking $k \to \infty$ for analytical tractability, we get:

$$\begin{aligned} q_{\text{ref}}(\mathbf{e}) &= \sum_{i=1}^{k} c_q n(\mathbf{e}'_i) r_{K}(\mathbf{e}'_i)^2 = \sum_{i=1}^{k} c_q \frac{n(\mathbf{e})}{k} r_{K}(\mathbf{e}'_i)^2 = c_q \cdot n(\mathbf{e}) \cdot \frac{1}{k} \sum_{i=1}^{k} r_{K}(\mathbf{e}'_i)^2 \\ &= c_q \cdot n(\mathbf{e}) \cdot \mathbb{E}\left[r_{K}(\mathbf{e}'_i)^2\right] = c_q \cdot n(\mathbf{e}) \cdot \left(\frac{r_{K}(\mathbf{e})(1 - r_{K}(\mathbf{e}))}{c + 1} + r_{K}(\mathbf{e})^2 \right) \end{aligned} \tag{D.5}$$

The difference between these two terms, $q(\mathbf{e})$ and $q_{\text{ref}}(\mathbf{e})$, gives the expected utility of refining an edge:

$$\begin{aligned}\Delta\tilde{q}(\mathbf{e}) &= q_{\text{ref}}(\mathbf{e}) - q(\mathbf{e}) \\ &= \left[c_q \cdot n(\mathbf{e}) \cdot \left(\frac{r_{_K}(\mathbf{e})(1-r_{_K}(\mathbf{e}))}{c+1} + r_{_K}(\mathbf{e})^2\right)\right] - \left[c_q \cdot n(\mathbf{e}) \cdot r_{_K}(\mathbf{e})^2\right] \\ &= c_q \cdot n(\mathbf{e}) \cdot \left(\frac{r_{_K}(\mathbf{e})(1-r_{_K}(\mathbf{e}))}{c+1} + r_{_K}(\mathbf{e})^2 - r_{_K}(\mathbf{e})^2\right) \\ &= c_{\Delta\tilde{q}} n(\mathbf{e}) r_{_K}(\mathbf{e})(1-r_{_K}(\mathbf{e})) \end{aligned} \tag{D.6}$$

where $c_{\Delta\tilde{q}}$ folds the unknown value $c+1$ in with the also unknown c_q. One small caveat: our agent has to estimate $r_{_K}(\mathbf{e})$ as $\tilde{r}_{_K}(\mathbf{e})$, but pseudo-counts in this case are unhelpful... to avoid them, we re-arrange terms:

$$\Delta\tilde{q}(\mathbf{e}) = c_{\Delta\tilde{q}} n(\mathbf{e}) \frac{n_s(\mathbf{e})}{n(\mathbf{e})} \frac{n_f(\mathbf{e})}{n(\mathbf{e})} = c_{\Delta\tilde{q}} \frac{n_s(\mathbf{e}) n_f(\mathbf{e})}{n(\mathbf{e})} \approx c_{\Delta\tilde{q}} \frac{n_s(\mathbf{e}) n_f(\mathbf{e})}{n(\mathbf{e}) + \epsilon} \tag{D.7}$$

with ϵ some constant for numerical stability (we arbitrarily pick $\epsilon = 1$).

This score, an indicator of how useful it would be to refine an edge, is actually quite intuitive. An edge that is never used (or has never been used) is unlikely to benefit from refinement. If an edge is completely reliable *or* completely unreliable, then $\Delta\tilde{q}(\mathbf{e}) = 0$ and it is useless to refine the edge. If the edge is of intermediate reliability, then $\Delta\tilde{q}(\mathbf{e}) > 0$, because refining the edge creates the possibility that some child-edges will be *more* reliable than the parent, while others less: these lesser children can be filtered out (inhibited), allowing our agent to separate the wheat from the chafe, so to speak.

D.4 Unobserved vertices and Vertex deletion

A state that has never been entered by the agent has a special status in the ARMS system. Any edge $\mathbf{e}_{i,j} = (\mathbf{v}_i, \mathbf{v}_j)$ for which $\mathbf{v}_i$ has not been visited is considered a *ghost edge*. Such an edge is considered to have zero epistemic value, and is inhibited until $\mathbf{v}_i$ is visited. Not maintaining this "ghost/non-ghost" distinction can cause the agent to fixate on impossible-to-reach goals.

Once the agent has reached the maximum number of vertices that it's memory-matrix can allow, to keep progressing it needs a way to delete useless vertices. To determine this, we developed an ad-hoc score, which incorporates the fraction of edges that a vertex $\mathbf{v}$ has which are "ghosts", $F_{\text{ghost}}(\mathbf{v})$. This score is computed as:

$$g(\mathbf{v}) = -\frac{(1 - F_{\text{ghost}}(\mathbf{v}))(n_{\text{transits}}(\mathbf{v}) + 1)(n_{\text{visits}}(\mathbf{v}) + 1)(M(\mathbf{v}) + 1)}{\sqrt{T_{\text{last}}(\mathbf{v})} + 1} \tag{D.8}$$

where $n_{\text{transits}}(\mathbf{v})$ is the number of times the agent successfully entered and then subsequently left $\mathbf{v}$, $n_{\text{visits}}(\mathbf{v})$ is the number of times $\mathbf{v}$ has been visited, and $T_{\text{last}}(\mathbf{v})$ is the time since $\mathbf{v}$ was last visited. When the agent wants to add a new vertex but no space

is left, it deletes the vertex which has the maximum g.

D.5 Pathfinding Algorithm

The $\mathcal{P}_{\circledcirc}$ pathfinding algorithm is based on recursive problem decomposition via wave-based midpoint finding. The algorithm can be described by the following functions:

Function 5: Find midway vertices.

```
function FIND_MID(m_s, m_g, G)
    f_s ← m_s, f_g ← m_g
    while True do
        f_s ← f_prop(G, f_s)
        if f_s ⊙ f_g ≠ 0 then
            return f_s ⊙ f_g
        f_g ← f_prop(G, f_g)
        if f_s ⊙ f_g ≠ 0 then
            return f_s ⊙ f_g
        if f_s = 0 OR f_g = 0 then
            return 0
```

Function 6: Find path to goal.

```
function P_◎(m'_s, m'_g, G)
    stack ← [], p ← []
    m_s ← m'_s, m_g ← m'_g
    while True do
        m ← FIND_MID(m_s, m_g, G)
        if m = 0 then
            return None
        if m ⊙ m_g = 0 then
            stack.push(m_g)
            m_g ← m
        else
            p ← RANDOM(m ⊙ m_s)
            p.append(p)
            m_s ← p
            m_g ← stack.pop()
            if p ⊙ m'_g ≠ 0 then
                return p
```

The `RANDOM` function returns a random one-hot vector from a multi-hot vector, and corresponds to random selection of a vertex from a set of vertices.

D.6 Maze Generation

We use two different kinds of random mazes, the first are mazes laid out on a rectilinear lattice, and the second are randomly distributed within the boundaries of a specific polygon. In both cases, we generate a set of "node" points (on a grid for rectilinear mazes, and randomly but evenly distributed using Lloyd's algorithm [163] for polygonal mazes) for which we compute the Delaunay triangulation, forming a planar graph. The topology of the maze is determined by randomly selecting a spanning tree of this graph. The interior of the maze is just determined by the geometric union of many "hallway" polygons corresponding to edges of this graph, with a small amount of "smoothing" to reduce the complexity of the polygon to speed up collision-detection.

D.7 Estimating Graph Reliability $R(\mathcal{G})$

In the text, we say that the reliability of a segraph $R(\mathcal{G})$ measures the ability of the agent to go from any point in a statespace to any other point in the statespace. Technically, this is not true for two reasons. First of all, we restrict ourselves to subset of statespace that is physically *reachable* by the agent, meaning we only include points that are inside of the maze. Second of all, there are a continuum of points within the maze, so measuring the reliability for all points is computationally impossible. We compromise by selecting "special" points in the maze, which correspond to the "node" points used for constructing the maze in the first place.

So, consider two node points from the maze, p_1 and p_2. We map these points to vertices of $\mathcal{G}$, $\mathbf{v}_1$ and $\mathbf{v}_2$, respectively. If either of these points isn't inside a vertex-field of $\mathcal{G}$, then the reliability of $\mathcal{G}$ for going between p_1 and p_2 is marked as 0. Otherwise, $\mathcal{A}$ uses the $\mathcal{P}_{\circledcirc}$ algorithm to find a path across the graph, using a value of α' that is a decaying running average of the actual α' used by the agent up until the point of measurement. If no path can be found, the agent uses its regular mechanisms for lowering the threshold until a path can be found. We then simulate the agent following the path. If the sampled trajectory hits a wall, we use the normal mechanism for handling traversal failure of inhibiting the failed edge and raising α', then looking for a new path. If at any point no path can be found, the reliability of the graph for going from p_1 to p_2 is marked as 0.

The "true" length of the path between p_1 and p_2 is determined using the graph used to initially generate the maze, denoted $d(p_1, p_2)$. If the reliability of the path is $r(p_1, p_2) \in \{0, 1\}$, and the set of node-points for the maze is P, then the formula for graph reliability is given as:

$$R(\mathcal{G}) = \frac{1}{\sum_{p_1,p_2 \in P} d(p_1, p_2)} \sum_{p_1,p_2 \in P} d(p_1, p_2) \cdot r(p_1, p_2) \tag{D.9}$$