Titre: Title:	Two-Object Search: Optimal Search of a Correlated Target
Auteur: Author:	Jérémie Bakambana
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Bakambana, J. (2025). Two-Object Search: Optimal Search of a Correlated Target [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/71420/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/71420/
Directeurs de recherche: Advisors:	Jérôme Le Ny
Programme: Program:	Génie électrique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Two-Object Search: Optimal Search of a Correlated Target

JÉRÉMIE BAKAMBANA
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie électrique

Décembre 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Two-Object Search: Optimal Search of a Correlated Target

présenté par **Jérémie BAKAMBANA**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Bowen YI, président
Jérôme LE NY, membre et directeur de recherche
Shuang GAO, membre

DEDICATION

À tous mes amis du labo,
vous me manquerez...

ACKNOWLEDGEMENTS

I express my gratitude to my research supervisor, Professor Jérôme Le Ny, for the opportunity to undertake this research under his guidance, which was helpful in shaping every stage of this work.

His support made this master's journey both possible and deeply rewarding. I am grateful for his mentorship and financial support, which were invaluable to the completion of this work.

My sincere appreciation also goes to Professors Bowen Yi and Shuang Gao for kindly accepting to be a part of the examination committee. Their thoughtful feedback and valuable perspectives have enriched this thesis in meaningful ways.

I am equally grateful to my fellow labmates at Polytechnique Montréal and GERAD, who have been a great support in cheering up my mood, sharing experiences, providing advice, and encouragement.

Finally, I wish to acknowledge my family and friends, who have always been there for me, keeping me in their prayers and supporting me mentally and financially during my journey.

RÉSUMÉ

Ce mémoire aborde le problème de la recherche optimale d'une cible statique cachée aléatoirement dans un environnement discret, et dont l'emplacement est corrélé avec un deuxième objet également statique. Nous proposons une approche basée sur des modèles pour résoudre ce problème de manière optimale. La corrélation entre objets est modélisée comme une distribution de probabilité conjointe, représentant la croyance du chercheur sur les emplacements possibles des objets. Nous utilisons l'inférence bayésienne pour mettre à jour cette croyance après chaque recherche infructueuse. Nous utilisons la programmation dynamique pour déterminer la politique optimale, qui tire parti du gain possible d'information que le deuxième objet peut fournir, s'il est détecté au cours de la recherche. Bien que cette approche basée sur la programmation dynamique fournisse un point de référence, elle est computationnellement difficile pour les problèmes à grande échelle en raison du fléau de la dimensionnalité. Cet écart entre l'optimalité théorique et la faisabilité pratique motive le développement et l'évaluation de méthodes approximatives. Nous étudions deux approches heuristiques distinctes : une méthode de planification en ligne de type k-steps (anticipation de k étapes) et une méthode hors ligne, sans modèle, utilisant l'algorithme Deep Q-Network. En comparant ces heuristiques à la solution optimale de la programmation dynamique et à une politique myope de base (gloutonne), cette étude fournit une analyse des compromis entre l'optimalité et l'efficacité computationnelle dans le problème de recherche de deux objets corrélationnels.

ABSTRACT

This thesis addresses the problem of optimal search for a static target randomly hidden in a discrete environment, and whose location is correlated with another static object. We propose a model-based framework to solve this problem optimally. The objects' correlation is modeled as a joint probability distribution, representing the searcher's belief about the objects' possible locations. We use Bayesian inference to update the searcher's belief after each unsuccessful search. We propose a solution based on dynamic programming to compute the theoretically optimal policy, which leverages the information gain from detecting the related object during the search. While the proposed dynamic programming method provides a benchmark, it is computationally intractable for large-scale problems due to the curse of dimensionality. This motivates the development and the use of scalable approximate methods for practical purposes. We investigate two distinct heuristic approaches: an online planning method based on $k-$step lookahead and an offline, model-free Deep Q-Network algorithm. By comparing performances of these heuristics against both the DP solution and a baseline myopic (greedy) policy, this study analyzes the trade-offs between optimality and computational efficiency in the two-object search problem with correlated objects.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ADP	Approximate Dynamic Programming
DP	Dynamic Programming
DQN	Deep Q-Network
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
NN	Neural Network
RL	Reinforcement Learning
SAR	Search and Rescue
STD	Standard Deviation
TSP	Traveling Salesman Problem

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Searching for objects[1] under uncertainty is a fundamental challenge in various real-world scenarios, such as search and rescue operations in disaster zones, locating survivors or people in distress, or finding the wreckage of a crashed aircraft. However, uncertainty about the target's location and various factors such as imperfect sensors and a limited search budget often complicate the search and do not always guarantee its success. Thus, the common goal in the majority of search problems is to develop search strategies that maximize the chance of locating a target (or a set of targets) under environmental constraints.

The search is usually initiated with a prior probability distribution that serves as a belief of possible target locations. This probability guides the searcher's initial decision and is updated after every unsuccessful search. Thus, the necessity of an informative prior distribution [1] is evident, since it influences the search process. Constructing an efficient prior distribution requires domain knowledge and expertise. A good example of this process can be seen in the search for the wreckage of Air France Flight AF 447 [2]. The prior was created as a weighted mixture of three different scenarios. The first one was a uniform distribution assumed within a 40 NM radius centered by the aircraft's last known GPS position. The second distribution was based on statistical data from past airplane crashes involving high-altitude loss-of-control. The third distribution relied on reverse-drift analysis of the dead bodies found on the surface. This reverse-drift scenario needed knowledge about the ocean and atmosphere to simulate how recovered bodies moved backward in time while considering ocean currents and wind drift. By assigning subjective weights to these scenarios based on analyst confidence, a prior probability map was created. In this study, we assume that the searcher already possesses the prior distribution, and their only concern is to developing a robust search strategy.

We investigate a search problem with a single static target whose location is correlated with another static object present in the same search environment. We can explain this correlation with several real-world situations. For instance, in a search and rescue operation, a missing person in a forest might be found near a water source. Similarly, in wildlife, the presence of a particular predator potentially implies the presence of a specific prey in the surroundings.

By modeling this correlation as a joint probability distribution, we can propose a framework that leverages this informative relationship to optimize the search efficiency. This allows the

[1]The term objects is used broadly throughout this work to encompass living (e.g., a person), non-living (e.g., a vehicle), or virtual (e.g., a piece of data) entities.

searcher to exploit the valuable information gain that the discovery of the related object yields, by bringing more precision about the target's possible location. In this case, subsequent searches shift from referring to the joint belief distribution to referring to a simple distribution conditioned on the related object location.

We propose a search method based on dynamic programming (DP) [3, 4] and Bayesian search [5] to determine the optimal policy that maximizes the chance of a successful search, accounting for the correlation with the related object. We assume a discrete search space, a finite search horizon, and a sensory system can miss detecting the target, but does not declare a false detection.

While the DP formulation provides a theoretically optimal search policy, its practical application is severely limited by the curse of dimensionality. With a large number of search cells and a large time horizon, the number of possible search trajectories explodes, making the optimal solution computationally intractable. This motivates the second central focus of this study: the development and evaluation of scalable approximate methods. To address this tractability challenge, we investigate two distinct heuristic approaches. The first is an online planning method based on the $k-$step lookahead, and the second is a Reinforcement Learning (RL) approach using Deep Q-Network (DQN). By comparing these scalable heuristics against the optimal DP solution, and the myopic (greedy) policy as the baseline, this study provides a comprehensive analysis of the trade-offs between optimality and computational efficiency in the two-object search problem involving correlated objects.

CHAPTER 2 LITERATURE REVIEW

Finding objects of interest within a complex environment is a significant challenge in various real-world scenarios. These include and are not limited to search and rescue (SAR) operations, where time is crucial for the survival of victims; the search for missing aircraft; the deployment of military assets; and the detection of invasive threats. This problem is relevant across to diverse fields such as robotics, autonomous systems, computer vision, artificial intelligence, tracking, surveillance, and cybersecurity.

This chapter establishes a foundation for our research by reviewing the key literature in optimal search theory. We start by tracing the historical development of the field, from its foundational principles to the modern classification of search problems. Then, we outline essential mathematical tools, such as Bayesian inference and DP, that are relevant for developing solution methods. This background provides the necessary context for our primary focus: the benefits and challenges in correlated multi-object search.

2.1 Foundational Concepts

Optimal search theory originated from World War II with the Anti-Submarine Warfare Operations Research Group of the U.S. Navy. A team led by Bernard Osgood Koopman pioneered research in the field by developing strategies to locate enemy submarines [6]. His work, published in the book *Search and Screening* [7], was initially classified but later expanded upon in the following years and serves as the foundation for many modern search techniques. Koopman developed methods to compute detection probabilities, identify the optimal search effort allocation, and analyze different search patterns.

Following the success of the Anti-Submarine Warfare Operations, optimal search theory has been used extensively by the U.S. Navy in several search operations, including finding the H-bomb lost in the ocean near Palomares, Spain, in 1966, and the submarine USS Scorpion lost in 1968 [8]. The U.S. Coast Guard popularized the theory by extensively using it in its SAR operations. In 1974, the U.S. Coast Guard deployed the first Computer-Assisted Search Planning (CASP) program, incorporating probability theory to assist in its SAR operations [9]. CASP was effective in finding the lost sailing vessel S/V Spirit and its crew members in 1976.

In the work [10], Koopman formulated the basic search problem for a stationary target

with an exponential detection function. L.D. Stone [8] provided a general description of the basic problem for a single stationary target and established the relationship between locally optimal and uniformly optimal plans. He also discussed notions such as the search allocation vector and the whereabouts search [11, 12]. He also detailed the ingredients for effective search planning, such as **the target's prior location distribution**, **sensor capabilities**, **the detection function**, **the search plan strategy**, and **the target's location belief update**. Over the years, the theory has been extended to a wide range of fields such as robotics and artificial intelligence [13–17].

2.2 Basic Search Problem

In its initial development, search theory was concerned with locating a target *randomly* hidden in a defined search environment. This can be a physical or a virtual entity that the searcher aims to find. Depending on the scenario, the target can be static, like a misplaced item in an office, or mobile, such as a survivor drifting at sea or a wreckage of a crashed airplane being moved by ocean currents. Stone [8] covered basic search problems and formalized different approaches for detecting a lost target, depending on the search scenario and objective. These approaches involve **Bayesian inference** for updating the belief about a target's presence based on collected information, and they serve as the foundation of modern optimal search methods.

Each search mission starts with a specific goal or objective. Typically, the main goal is to **maximize the chance of finding the target** with a limited budget. In these scenarios, early detections and later detections are considered equal successes. This is like looking for a misplaced item at home.

Sometimes, the objective could be to **locate the target as early as possible**. This is crucial in time-sensitive scenarios like SAR operations, where every second is critical for victims' survival. Similarly, the goal could be to **minimize the search budget**, such as saving a robot's battery or reducing fuel consumption. In these cases, the strategy isn't just about finding the target, but about finding it with the least amount of effort and resources spent.

Several key concepts help formalize search scenarios as solvable problems. Among these we have **the search space**, which represents the environment where the search occurs. This environment can be a continuous area, such as a patch of ocean, or, more commonly for computational methods, it can be *discretized* into a finite number (let's say $n > 1$) of cells,

such as a *search grid* as shown in **Figure 2.1**. The structure of the search space, including obstacles or varying terrain, impacts the search process. Within this defined search space, the precise location of the target is unknown. However, at the beginning of the search, the searcher usually possesses an initial probability distribution, known as **the target's prior distribution**, denoted by p_0. This prior distribution creates a probability map over the search environment, reflecting the searcher's belief of the target's presence across that search environment. In discrete search scenarios, the prior distribution is a vector of the form $p_0 = [p_{0,1}, \ldots, p_{0,n}]$. Let define the set $[n] = \{1, \ldots, n\}$. We then have $p_{0,i}$, with $i \in [n]$, the probability that the target might be located in cell i. For example, based on the scenario illustrated in Figure 2.1, we could have

$$p_0 = [0.10, 0.12, 0.08, 0.15, 0.11, 0.09, 0.13, 0.10, 0.12].$$

The prior distribution is critical, as it allows the searcher to formulate an optimal search strategy. As the search progresses and new information is gathered, this distribution can be refined into a **posterior distribution** (as shown in Figure 2.2) to guide subsequent decisions.

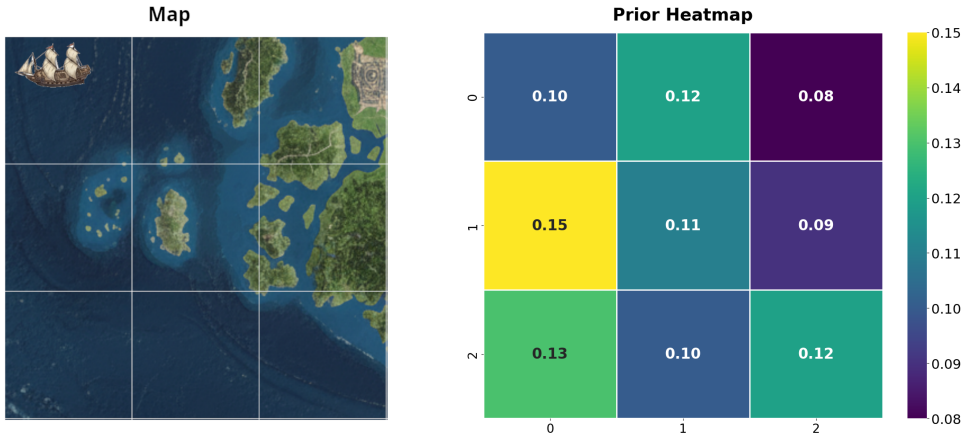

Figure 2.1 Illustration of a discretized search environment. The left panel shows a conceptual map divided into 3×3 search cells, with a search vessel (the target) positioned in the top-left cell. The right panel displays an example heatmap representing the searcher's belief (probability distribution) over the possible locations of an object across these cells. Higher probability values (yellow) indicate a greater likelihood of the object being in that cell.

The process of searching consumes resources, collectively referred to as **search effort** or a search budget. This can be for example time, energy, or financial cost allotted for the search. In practice, the searcher often has a limited budget in hand, and therefore aims to *strategically* allocate that budget across the search environment. In the case of discrete search, this refers to deciding the quantity of the search budget each cell should receive. However, the

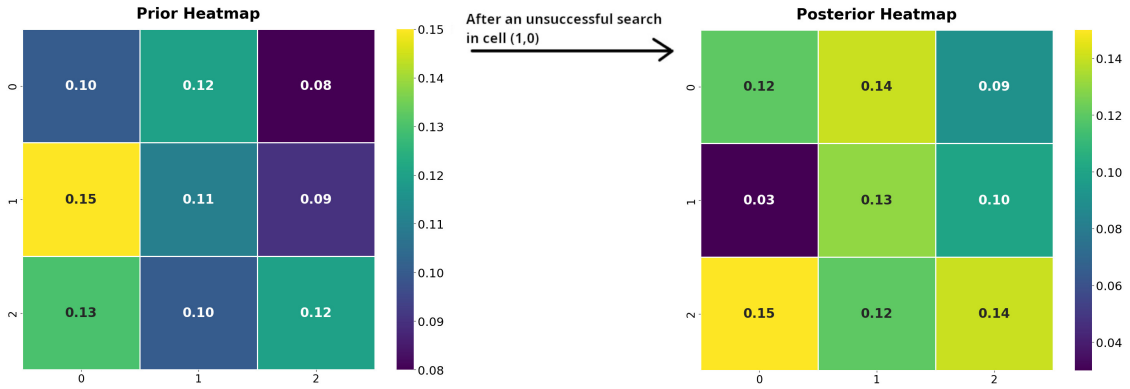

Figure 2.2 Illustration of a prior belief updated to a posterior belief. The left panel shows the searcher's prior belief (probability distribution) over a 3×3 grid. An unsuccessful search is then conducted in cell $(1, 0)$. The right panel displays the updated posterior belief, where the probability mass has been shifted away from the unsuccessfully searched cell $(1, 0)$.

effectiveness of the effort applied to a given cell is directly linked to the **sensor capability** to detect the target when it is present in that cell. Sensors can be categorized as *perfect* sensors, which detect a target with certainty (with a detection probability of 1) when the target is within its detection range, or *imperfect* sensors. Imperfect sensors can be subject to *false positives* (incorrectly indicating a detection) and, more commonly, *false negatives* (failing to detect a target that is present within the detection range). For these imperfect sensors, the levels of imperfection are measured with probabilities of declaring either a false positive or a false negative detection. These elements—the target's prior distribution, the search effort, and the sensor's reliability— are fundamental for developing an efficient search strategy to optimize the chance of a successful search. In the remainder of this work, we consider only discrete search scenarios.

2.3 Effort Allocation Vector

An allocation vector, $z = [z_1, \ldots, z_n]$, where $z_i \in \mathbb{N}$, represents a specific distribution of a total search effort K across the n cells of the search environment. This means $\sum_{i=1}^{n} z_i = K$. We define $Z = \left\{ z \in \mathbb{N}^n \mid \sum_{i=1}^{n} z_i = K \right\}$ as the set of all possible effort allocation vectors. For each vector $z \in Z$, we denote $P[z]$ as the probability of locating the target with that allocation and $C[z]$ as the associated cost. The cost is constrained by $C[z] \leq K$, where K represents the total search effort (or search budget). Based on this framework, Stone [12] formally defines the basic search problem as finding an optimal allocation z^* that maximizes the overall detection probability while adhering to the cost constraint:

$$P[z^*] = \max_z \{P[z] : z \in Z \text{ and } C[z] \leq K\}. \tag{2.1}$$

To define $P[z]$ and $C[z]$, we first consider the following key assumption [8]:

Assumption 1 *Searching a cell with the sensor cannot lead to a false positive detection.*

Assumption 1 is important to simplifying the search problem for two main reasons. First, it allows the search to end with certainty as soon as a detection occurs. Secondly, it simplifies the problem structure. Without this assumption, a detection would not be definitive; it would only increase the belief that the target might be in cell $i \in [n]$, and the search would likely continue to gather more certainty. This would result in a more complex Partially Observable Markov Decision Process (POMDP), making it significantly harder to solve. This assumption can be justified, for instance, by scenarios where any potential detection is confirmed by a reliable second source (like a human supervisor) before being declared.

We also make the following assumption [8]:

Assumption 2 *The miss-detection rate for a specific cell i is assumed to be stationary and independent across repeated search attempts.*

If $z_i \in \mathbb{N}$ is the number of time a cell i has been searched, and $\gamma_i \in [0, 1]$ is the probability of failing to detect the target in a single and independent search in cell i, Assumption 2 means that the probability of failing to detect the target given its presence in cell i remains constant and invariant in time for every look. This means that the cumulative probability of non-detection after z_i independent attempts is geometrically distributed as $\gamma_i^{z_i}$. This implies that the overall miss-detection in a cell containing the target decreases as the number of looks in that cell increases.

Consequently, the probability of finding the target after z_i independent attempts is the complement of failing z_i times independently, which is $1-\gamma_i^{z_i}$. By weighting this detection probability by the prior belief $p_{0,i}$ and summing over all cells, Stone [8] defined the overall detection probability for a given search plan as:

$$P[z] = \sum_{i=1}^{n} p_{0,i} \left(1 - \gamma_i^{z_i}\right). \tag{2.2}$$

Similarly, if we define $c(i, z_i)$ as the cost of applying effort z_i to cell i, the total cost for the allocation vector z is the sum of the individual costs:

$$C[z] = \sum_{i=1}^{n} c(i, z_i). \tag{2.3}$$

Therefore, the basic search problem, under Assumption 1, concerns of finding the optimal integer vector z^* that maximizes (2.2) subject to the constraint that its total cost, calculated via (2.3), does not exceed the budget K.

2.4 Types of Search Problems

Depending on the objective and the number of targets [18–20], or the number of searchers [21], we can identify different categories of search problems. Benkoski et al. [6] suggested dividing them into two main categories: *One-Sided Search Problems*, where only the searcher creates strategies to find the target, and *Two-sided Search Problems*, where the target is also an active agent using its own strategy, either to escape from the searcher or to cooperate in being found.

2.4.1 One-Sided Search Problems

In this category, the target does not use any particular strategy to respond to the searcher's actions. The target can be stationary [11, 20] or moving [12]. The latter case leads to a search and tracking problem [22]. The most common goals in this category usually involve maximizing the detection probability or minimizing the search cost (e.g., time or resources). This category applies to many real-world scenarios, such as search scenarios with non-mobile objects in SAR and Unmanned Aerial Vehicle (UAV) missions [23, 24], police searches for evidence, detection of Improvised Explosive Devices (IEDs) [25], and visual search in real scenes [26].

2.4.2 Two-sided Search Problems

In this category we distinguish two specific scenarios: *cooperative* and *non-cooperative* searches [27]. An example of the first one is the *rendezvous search*, where both entities cooperate to locate each other. In the *non-cooperative search*, the target knows about the searcher's intentions and develops strategies to escape or to remain hidden. Search problems like these are typically modeled with game theory [28]. These scenarios often require more complex search

strategies and dynamic decision-making. We count many subcategories from this, such as the *Mobile Evader* and the *Ambush Game* [29–31]. In the latter, for example, the hider tries to move between specific areas without being detected, while the searcher optimizes their resource allocation across regions to increase the chance of an ambush. These types of search problems find many real-world applications, such as in military and law enforcement operations or predator-prey studies. Moreover, these types of searches can involve one or multiple searchers against one or multiple hiders.

Note: Although this work investigates a Two-Object search problem involving a target and a correlated object, both objects are assumed static and non-reactive. Thus, the research presented in work falls under the category of **One-Sided Search Problems**.

2.5 Bayesian Search

From the success of CASP, developments in Bayesian search theory became key to optimal search problems. This method uses *Bayesian inference* to update beliefs based on information gathered, thus improving the efficiency of future search decisions [5, 22, 32, 33]. For instance, this could be dynamically updating the prior belief p_0 to a **posterior belief** p_t leveraging all collected information after t search attempts.

Under Assumption 1, and assuming uniform costs across all search cells, Stone [8] developed a uniformly optimal search plan that uses Bayesian inference to minimize the time to detect a static target. Loxley et al. [4] applied Bayesian inference in the context of entropy maximization to leverage informative measurements collected during the search in decision-making. Stone et al. [2] used Bayesian inference to produce the posterior distribution of the Flight AF 447. Their Bayesian model enabled discovery undersea in one week of the wreckage, after two years of failure with prior search methods. Other practical applications include the search for flight MH370 [34], although that specific search was unsuccessful.

2.6 Dynamic Programming for Search Problem

Optimal search in discrete cases can be viewed as a sequential decision-making problem under uncertainty. At each stage of the search, the searcher must decide where to search and how much of their limited resources to allocate to that location to optimize the chances of a successful search. This decision relies on the information currently available. DP [3], introduced by Bellman, is the foundational mathematical framework for solving such problems.

It provides a structured way for finding a provably optimal policy for sequential problems.

2.6.1 Framework

The core of DP is the ***Bellman's principle of optimality***. This principle states that: *"an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision"*. For instance, any segment of the shortest path between two vertices in a graph is also a shortest path between its endpoints.

Before applying DP to a search problem, we need to formalize it using its main components. A **state** x_t provides a complete description of a system at a given time t. In a search problem, the state could reflect all information gathered after t unsuccessful search attempts. This can include how many times each area has been visited, and how much of the search budget has been spent there. A **control** u_t indicates the decision made based on the knowledge provided from the state. In a discrete search problem, the control could correspond to the choice of the cell to search. The system dynamics, noted as :

$$x_{t+1} = f_t(x_t, u_t) \tag{2.4}$$

determines the next state, which is the consequence of taking a control u_t while in a state x_t. However, in many practical scenarios the outcome of a control is uncertain. This uncertainty is modeled as a variable w_t, which represents the randomness of the outcome. In this context, the next state x_{t+1} is expressed as:

$$x_{t+1} = f_t(x_t, u_t, w_t). \tag{2.5}$$

In a search problem, w_t could be a consequence of Assumption 1, representing the uncertainty that the target will be found or not after implementing the control u_t, which affects the next state.

A policy is a rule or function that dictates how to choose controls based on the current state. Let μ_t be a function that maps a given state x_t to a control u_t. The function μ_t is called a **deterministic policy** if it maps each state x_t to a specific control u_t with probability 1. However, if μ_t maps x_t to a control u_t with a certain probability $p(u_t|x_t)$, then μ_t is called a **stochastic policy**. In a discrete problem with a given initial state x_0 and a finite number of stages T, we can solve a DP problem using a policy $\pi = [\mu_0, \cdots, \mu_{T-1}]$ starting from x_0.

For a given problem, we denote Π the set of all admissible policies.

2.6.2 Objective Function

Another crucial notion in DP is the **objective function**. DP aims to identify a policy that optimizes a given objective function. This may involve minimizing the total cost incurred by a sequence of controls, or maximizing the gain derived from that sequence. Let $g_t(x_t, u_t, w_t)$ denote the cost at stage t after applying a control u_t while in state x_t (the disturbance w_t is considered as well for non-deterministic scenarios). Given an initial state x_0 and a policy π, the expected total cost of π can be expressed as:

$$J_\pi(x_0) = E\left\{g_T(x_T) + \sum_{t=0}^{T-1} g_t(x_t, \mu_t(x_t), w_t)\right\} \tag{2.6}$$

Here, $g_T(x_T)$ is the terminal cost at the final state x_T. The goal of DP is to find an optimal policy π^* that minimizes this expected cost:

$$J_{\pi^*}(x_0) = \min_{\pi\in\Pi} J_\pi(x_0) \tag{2.7}$$

Let define $U(x_t)$ as the set of all admissible control u_t for the state x_t. Using the principle of optimality, π^* can be found by proceeding via **backward induction** using the following algorithm, known as the Bellman equation:

$$\begin{cases} J_T(x_T) = g_T(x_T) \\ J_t(x_t) = \min\limits_{u_t\in U(x_t)} \mathbb{E}\limits_{w_t} \left\{g_t(x_t, u_t, w_t) + J_{t+1}(f_t(x_t, u_t, w_t))\right\}, \quad t = T-1, \ldots, 0 \end{cases} \tag{2.8}$$

Despite the guarantee of optimality, a significant challenge in applying DP to real-world search problems arises when dealing with large state spaces, or long problem horizons. For example, Chakraborty et al. [35] and Sarmiento et al. [14] explored searching with non-uniform travel times. They showed that when the cost of moving between search points is not constant—common in many robotics and SAR applications—the problem of finding the optimal search path becomes NP-hard. This NP-hardness emerges because the optimal sequence of future moves depends on the entire path taken so far to minimize travel costs, similar to routing problems such as the Traveling Salesman Problem (TSP). This computational challenge necessitates the development of Approximate Dynamic Programming (ADP). Techniques such as limited lookahead policies and rollout algorithms are ADP methods that

approximate the true cost to go J_{t+1} in the Bellman equation, allowing for high-quality and computationally feasible solutions. These methods are suitable for real-time decision-making with large-scale search scenarios.

2.6.3 Limited Lookahead Policies with Rollout Approximation

A useful method for making DP tractable is the limited lookahead policy. The idea is to truncate the full DP problem into a manageable, short-term horizon that looks ahead only k stages. At each time step t, the agent solves this k-step DP problem to identify the most promising action. This approach, however, creates a new challenge: to solve the k-step problem, we need to know the value (or cost-to-go) of the states at the end of that short horizon. Since we are not solving the entire DP, the true value function, J_{t+k}, is unknown. It must be replaced by an approximation, $\tilde{J}_{t+k}$.

This is where the rollout algorithm is useful. A rollout is a simulation-based method that approximates $\tilde{J}_{t+k}$. It proceeds by estimating the value of a state at the end of the lookahead (x_{t+k}) through numerous simulations from that state to the final time horizon, T. These simulations are based on a predefined, computationally simple base policy, π_{base}. Then, the approximate value, $\tilde{J}_{t+k}$, is simply the average of the total costs (or rewards) from all the simulated trajectories.

The final, combined heuristic—and the one used in our study—is the k-step lookahead with the rollout algorithm. This combined approach benefits from both worlds: it performs an exact, detailed DP optimization for the immediate k steps, allowing for intelligent short-term planning, while also leveraging the fast and efficient rollout simulations to provide a reasonable estimate of the long-term future beyond that k-step window.

The depth k controls a crucial trade-off between performance and computational cost. A small k (e.g., $k = 1$) is computationally fast at each step but may be myopic at a certain point. As k increases, the short-term plan becomes more sophisticated and the policy's performance gets progressively closer to the true optimal DP solution. Ultimately, if we set $k = T$, this heuristic becomes the DP algorithm.

This hybrid approach has an important property: as the agent approaches the end of the mission, its decisions become provably optimal. If the fixed lookahead k is greater than or equal to the remaining time (i.e., $k \geq T - t$), the lookahead horizon effectively becomes the

remaining time, $T-t$. In this case, the agent solves the exact DP problem for the remainder of the search. The rollout algorithm is no longer needed because the lookahead reaches the true terminal time T, and the defined terminal value J_T (which is 0 in our case) is used.

As a simple example, if we consider a one-step lookahead ($k = 1$), the agent's action $\tilde{\mu}_t(x_t)$ at each step t is derived by finding the action that maximizes the immediate reward plus the rollout-approximated value of the next state:

$$\tilde{\mu}_t(x_t) = \arg \max_{u_t \in U(x_t)} \mathop{\mathbb{E}}_{w_t} \left\{ g_t(x_t, u_t, w_t) + \tilde{J}_{t+1}(f_t(x_t, u_t, w_t)) \right\}. \tag{2.9}$$

We should note that $k = 0$ corresponds to the **greedy policy**, which aims to optimize only the immediate outcome, regardless of the future, and $k = T$ corresponds to the complete DP.

2.6.4 Reinforcement Learning for Search Problems

Function approximators, such as **Neural Networks (NNs)**, are powerful tools used to learn complex, high-dimensional functions directly from input data. In the context of Reinforcement Learning (RL), they are essential for solving problems with large state spaces. Instead of calculating and storing a value for every possible state, an NN learns to approximate this value function, generalizing from previous experience to make predictions about new observations. This offers a powerful alternative to DP for large-scale search problems, particularly those with a significant number of search cells and a long search horizon.

However, for good performance, the state representation-how the NN observes the environment-plays a critical role. Matzliach et al. [36] on searching for static and mobile targets is one of the examples of using NN to solve search problems. They successfully trained a Deep Q-learning (DQN) [37] agent to locate targets in a 2D occupancy grid space. To manage the complexity of the problem, they defined the agent's observation as a 2D occupancy grid, which represents the agent's belief about the probability of the target being in each cell. This entire grid was used as the input to a neural network, which learned to map this belief state to an effective search action. Another example comes from the work of Matzliach et al. [17], who also included the agent's belief in the observation provided to the neural network. This demonstrates that a neural network can effectively learn a search policy directly from the targets' belief distribution. This provides a foundation for our use of NN-based methods as a scalable heuristic for complex search problems.

2.7 Correlated Search in Literature

The foundational work in search theory, as explored by Stone [8], focuses on the classic search problem of finding a single hidden target. While this is a crucial starting point, many real-world scenarios involve more complexity, such as searching for multiple targets or using multiple searchers.

Over the years, many studies have explored these multi-target problems [5, 22, 36, 38, 39]. Various methods for tackling them have been proposed. However, a common thread used in much of these works is the simplifying assumption that the locations of the different targets are independent of one another. While this assumption makes the problem much easier to solve, it overlooks a significant strategic advantage: the valuable information that the relationship between targets can provide. By ignoring this correlation, these methods miss out on the opportunity to make the search strategy more intelligent and efficient. Other studies investigated search problems with correlated objects [15, 16, 40]; however, these approaches often rely on ad-hoc or greedy strategies, which are simple heuristics, on-the-spot rules rather than a fully optimized plan.

Another concept commonly used in correlated search scenarios is the **indirect search** [13, 40, 41], where a mobile robot first locates a common, easy-to-find intermediate object to exploit its known spatial relationship with a primary target. This structure primarily exploits spatial relation, and finds applicability in many object navigation problems in robotics such as the Habitat Navigation Challenge 2023 [42]. While conceptually powerful, this strategy's efficiency is contingent on the target actually being in the vicinity of the intermediate object.

More recent work has sought to formalize this correlational search from a general probabilistic point of view. El-Hadidy [39], for instance, modeled a search for two correlated targets but constrained this relationship with the simplifying assumption that the related object, still randomly hidden, must always be in one of the cells adjacent to the primary target. While this makes the problem mathematically tractable, it limits the model's applicability to this specific spatial and probabilistic configuration. In contrast, the approach by Zheng et al. [43] uses a more flexible, data-driven framework based on POMDPs. Their model learns the complex spatial relationships between multiple object types from simulations and uses these learned correlations to update its belief about a target's location when a related object is observed.

Our work builds upon these ideas but addresses a distinct gap. It differs from prior works in

two key aspects: (1) while the model by Zheng et al. is powerful, it relies on a learned model within a complex POMDP framework. Our approach, conversely, operates on a pre-defined, explicit correlation model for two objects. By representing this relationship with a joint probability matrix, p_0, we avoid imposing rigid structural constraints—such as adjacency—on how the objects' locations are related. This allows our method to capture any possible statistical relationship, from targets being in the same location to being on opposite sides of the search environment. This formulation makes our approach more flexible and applicable to a wider range of search scenarios where the exact nature of the correlation might not be perfectly defined; (2) our model explicitly accounts for the dual-outcome nature of each search action. Many search models focus solely on the primary target, treating related objects only as contextual clues. In our framework, every search for the primary target in the location O_1 is also simultaneously a search for the related object in the location O_2. The optimal policy therefore learns a more complex strategy: it must dynamically balance the immediate probability of finding the main target against the long-term informational value of incidentally discovering the secondary object, which would trigger a more informed search for the remainder of the mission.

2.8 Research Objective

This study addresses a search problem involving a primary target whose location is correlated with a secondary object in a discrete environment. Our main objective is to develop a search strategy that leverages the objects' correlation to identify the policy that maximizes the chance of a successful search. To achieve this, we start by developing **a general model for correlational search**. We aim to create a framework that leverages the probabilistic relationship between objects without any limiting structural or spatial constraints on their correlation, which is a common approach to simplify the problem in existing literature. Secondly we aim to build a **forward-looking search policy**. In contrast to greedy approaches that only consider the best immediate action, we use methods like DP to derive a policy that is globally optimal over the entire, finite search horizon. The output of our strategy is a policy that maximize the chance of a successful search.

We consider a search problem where a **primary target** and a **related object** are hidden within the same environment, which is divided into n discrete cells. Their locations are not independent; their relationship is described by a known joint probability distribution. The searcher's goal is to find the primary target within a finite number of search stages, T, using imperfect sensors that are subject to missed detections (false negatives) but not false

positives.

The most straightforward approach is to conduct a single-object search. This involves extracting the target's marginal distribution from the joint prior and applying an optimal strategy, such as those derived from the work of Stone [8]. While this method is optimal for the single-object case, it can be inefficient if the sensor's ability to detect the primary target is low, as many search actions may be wasted.

However, a more sophisticated strategy can leverage the correlation between the objects. During the search, the agent might incidentally detect the related object, perhaps because it is larger or more easily detected by the sensor. The **data-processing inequality** [44], a principle from Information Theory, states that additional information cannot increase uncertainty. This implies that discovering the true location of the related object provides valuable intelligence for the remainder of the search. When this happens, the agent can discard its old belief about the target's location and adopt the new, more informative **posterior distribution**, conditioned on the related object's known position. A search strategy guided by this new posterior performs at least as well as the one based on the original marginal distribution, as "information never hurts".

While the problem simplifies to a standard single-object search after the related object is found, the true challenge lies in the initial phase when both objects are hidden. The searcher requires a unified strategy that can reason about all possible future outcomes. The challenge, therefore, is to develop a policy that efficiently searches for the primary target while intelligently considering that at any stage, it might discover the secondary object, fundamentally changing the information state for all subsequent steps.

2.9 Conclusion

This chapter has provided an overview of the optimal search theory in the literature, beginning with the foundational principles of single-object search and extending to more complex multi-object problems. It reviews the core mathematical tools, such as Bayesian inference for belief updates and DP for policy optimization, that are commonly used to solve these problems. Furthermore, we analyze the existing work on correlated search, highlighting the key differences in how prior studies have modeled these relationships compared to the more general approach adopted in this study.

A key takeaway from this review is that while many methods exist, they often rely on simplifying assumptions—such as targets independence or restrictive correlation models—to maintain tractability. This reveals a clear gap in the literature for a method that can handle

a general, unconstrained correlation model while still providing a framework for computing an optimal policy.

By contextualizing our research within this landscape, we have established the foundation for our contribution. The subsequent chapters will now detail our proposed model, which directly addresses this gap by combining a flexible joint probability model with an DP solver, and will then explore scalable, approximate methods for applying this framework to large-scale problems.

CHAPTER 3 ONE-OBJECT SEARCH DYNAMIC PROGRAMMING

This intermediate chapter introduces DP as an approach to finding the optimal policy for the single-object search problem. The main objective is to establish this as a foundational step towards addressing the two-object search problem in the next chapters.

3.1 The Search Problem

Given a finite search horizon $T \in \mathbb{N}$ and a search environment subdivided into a finite number, $n \in \mathbb{N}$, of cells, we aim to determine the policy that maximizes the chance of finding a hidden target in one of the cells. The target is stationary but randomly located in one of these cells. We denote by O_1 the target's location within the search environment. Before the search starts, the searcher possesses a prior belief about where the target might be, represented by a probability distribution

$$p_0 = [p_{0,1}, \ldots, p_{0,n}]$$

over the n cells, where $p_{0,i} = P[O_1 = i]$ is the probability that the target is in cell i. The prior is updated to a posterior if an unsuccessful search occurs, where the posterior becomes the prior for the next search iteration. So, at any search stage $t \in [T]$, the searcher chooses a cell $a_t \in [n]$ in which to search for the target.

We consider Assumption 1 and Assumption 2 of **Section 2.3**, and formally define the miss-detection rate γ_i as:

$$\gamma_i = P[\text{No detection in } i \mid O_1 = i].$$

As a consequence of Assumption 1, we have that:

$$P[\text{Detection in } i \mid O_1 \neq i] = 0.$$

This leaves the sensor's performance to be entirely defined by its false negative rate, or miss-detection probability. With this we ensure that the history of sequential non-detections is a **sufficient summary** of the information gathered so far. This approach spares us the complexity of tracking belief states, which belong to a continuous space. Relying on the belief state directly would transform the problem into a Partially Observable Markov Decision Process (POMDP), which is generally computationally intractable for methods like Dynamic Programming.

Under this assumption, the **state** s_t is defined by the search history up to time t, represented by the sequence $h_t = [a_0, \ldots, a_{t-1}]$, which stores the cells that have already been visited. This search history is crucial since it serves as the sufficient statistic required to determine the current belief of the target location. We should also point out that a cell $i \in [n]$ can be visited more than once because the searcher might have missed detecting the target in cell i in previous looks; thus, it can appear multiple times in h_t. The state can also **take** a terminal value, denoted by $\perp$, where $s_t = \perp$ represents that the search has stopped, i.e., a detection has already occurred. $\perp$ is independent of the specific search history h_t at the moment the target is found; once the target is found, the state transitions directly to $\perp$[1], regardless of the specific history that led to its discovery or how many times other cells have been visited. This property is essential for modeling the problem as a Markov Decision Process (MDP), as it allows for a precise, history-independent terminal state. Let us define

$$[n]^t = \underbrace{[n] \times \cdots \times [n]}_{t \text{ times}} = \{h_t \mid a_\tau \in [n], \forall \tau \in \{0, \ldots, t-1\}\}$$

as the set of all possible search histories at time t. We denote the state at time t by $s_t \in [n]^t \cup \{\perp\}$.

Let's consider the case of sequential non-detections. Having the search history, defined by the sequence h_t, means the target was not detected at the beginning of stage t. We are interested in finding the probability of this event, conditioned on the target's true location O_1. Thus, for each cell $i \in [n]$, we can denote the probability of not finding the target after sequentially searching in h_t, given that the target is located in cell i as:

$$P[\text{No detection in } h_t \mid O_1 = i] = \prod_{\tau=0}^{t-1} P[\text{No detection in } a_\tau \mid O_1 = i]. \tag{3.1}$$

The product in (3.1) is explained with the fact that searches are independent events. Additionally, we have the following definition:

$$P[\text{No detection in } a_\tau \mid O_1 = i] = \begin{cases} \gamma_i, & \text{if } a_\tau = i \\ 1, & \text{otherwise.} \end{cases} \tag{3.2}$$

Applying (3.2) in (3.1), we determine that for each cell $i \in [n]$, at a given search stage t, the probability of non-detection up to stage t given that the target is in cell i can be written as

[1] $\perp$ is a singleton state that acts as an absorbing state in the MDP; once the system enters $\perp$, all subsequent transitions remain in $\perp$ with zero future rewards.

follows:

$$P[\text{No target detection in } h_t \mid O_1 = i] = \gamma_i^{z_i}, \tag{3.3}$$

with z_i the number of times cell i appears in h_t[2].

We can observe from (3.3) that the order of appearance of cells in the search history doesn't influence the probability of non-detection, only their respective number of appearance matters. Now that we have the probability of non-detection up to stage t, we are interested in extracting the posterior of the target location, since the searcher needs to update the belief after each unsuccessful detection. Hence, using Bayes' law, we have the following:

$$P[O_1 = i \mid \text{No detection in } h_t] = \frac{P[\text{No detection in } h_t \mid O_1 = i]}{P[\text{No detection in } h_t]} \cdot P[O_1 = i]. \tag{3.4}$$

Using the law of total probability, we can further expand the denominator on the right side of (3.4) as follows:

$$P[O_1 = i \mid \text{No detection in } h_t] = \frac{P[\text{No detection in } h_t \mid O_1 = i] \cdot P[O_1 = i]}{\sum_{j=1}^{n} P[\text{No detection in } h_t \mid O_1 = j] \cdot P[O_1 = j]}. \tag{3.5}$$

Using (3.3), we can express (3.5) as:

$$P[O_1 = i \mid \text{No detection in } h_t] = \frac{\gamma_i^{z_i} \cdot p_{0,i}}{\sum_{j=1}^{n} \gamma_j^{z_j} \cdot p_{0,j}}. \tag{3.6}$$

Hence, (3.6) provides the posterior probability (the belief) that the target is located in cell i given the search history h_t.

The **action space** $\mathcal{A}$ is simply $[n]$, where each $i \in [n]$ corresponds to a specific cell of the search environment. So, performing an action a_t at stage t corresponds to selecting a specific cell in $[n]$.

[2]This is equivalent to the number of time cell i has been previously visited

3.2 Search Vector

Appending every cell selection after each unsuccessful search becomes inefficient when the search horizon is large, as storing a long list of actions becomes computationally expensive. However, the expression in (3.6) shows that the posterior belief only depends on the number of searches in each cell, not their specific order in the search history. Therefore, instead of storing a dynamic list of search history, we propose to represent the search history in a compact way using a search vector

$$z = [z_1, \ldots, z_n],$$

where $z_i \geq 0$ is the number of times cell i has been searched. For example, if we consider a scenario with 3 cells, the search history $[cell_3, cell_2, cell_1, cell_2, cell_1, cell_2]$ at stage $t = 6$ can be encoded in a simpler way as $z = [2, 3, 1]$.

So considering any possible z, we can formally define this posterior belief $p_i(z)$ as:

$$p_i(z) = \frac{p_{0,i}\gamma_i^{z_i}}{\sum_{j=1}^{n} p_{0,j}\gamma_j^{z_j}}. \tag{3.7}$$

So the vector $p(z) = [p_1(z), \cdots, p_n(z)]$ is the posterior distribution of the target's location at stage after given a search vector z.

At any time t, the state s_t could simply be a vector z corresponding to the search counts at that stage. This vector must satisfy the constraint that the sum of its components equals the time step (i.e., $\sum_{i=1}^{n} z_i = t$). Consequently, at the initial stage (i.e., $t = 0$), the search history is $z = [0, \ldots, 0]$. The example vector $[2, 3, 1]$ mentioned earlier would be the vector z_6.

Thus, we can formally define the complete state space $\mathcal{S}$ as the set of all possible search vectors for all time steps, plus the terminal state:

$$\mathcal{S} = \left\{ z \mid z \in \mathbb{N}^n, \sum_{i=1}^{n} z_i = t \text{ for some } t \leq T, \text{ and } z_i \leq t \right\} \cup \{\perp\}$$

.

3.3 State Transitions

The **transition model** $\mathcal{T}$ describes the probabilities of moving between states given a specific action. When an action a is taken (i.e., decide searching cell a), the search history vector z is updated to a new vector by incrementing by 1 the component at index a in z. We formalize this update by defining a basis vector δ_a which is a vector of zeros except for a 1 at index a, reflecting that cell a has been visited one more time. Therefore, the new vector is $z + \delta_a$, where "+" denotes element-wise addition. At each search stage t, assuming the target has not yet been found (i.e., $s_t \neq \perp$), the transition model outlines two possible outcomes after taking action a: either the target is found, leading to $\perp$, or the target is not found, causing the search to continue with an updated history vector $z+\delta_a$. These probabilities are formally expressed as $P[s_{t+1} = \perp \mid z, a]$ and $P[s_{t+1} = z + \delta_a \mid z, a]$, respectively, and they must satisfy the total probability rule:

$$P[s_{t+1} = \perp \mid z, a] + P[s_{t+1} = z + \delta_a \mid z, a] = 1. \tag{3.8}$$

To formalize the DP of the search problem, we need to determine each of these probabilities. We can use the law of total probability to determine the expression for $P[s_{t+1} = \perp \mid z, a]$. This represents the probability that a search in cell a is successful, causing the system to transition to the terminal state $\perp$. Therefore, we have:

$$P[s_{t+1} = \perp \mid z, a] = \sum_{i=1}^{n} P[s_{t+1} = \perp \mid z, a, O_1 = i] \cdot P[O_1 = i \mid z, a]. \tag{3.9}$$

The action selection doesn't influence the posterior of the target's location until we observe its outcome. Thus, $P[O_1 = i \mid z, a] = P[O_1 = i \mid z]$, which is the posterior $p_i(z)$ computed with (3.7). We can split the sum in (3.9) into two scenarios: (1) the target is in the cell being searched ($i = a$), which has a detection probability of $1 - \gamma_a$, and (2) the target is in any other cell ($i \neq a$), which leads to a detection probability of 0. Therefore, we have :

$$
\begin{aligned}
P[s_{t+1} = \perp \mid z, a] &= P[s_{t+1} = \perp \mid z, a, O_1 = a] \cdot p_a(z) \\
&\quad + \sum_{i \neq a} P[s_{t+1} = \perp \mid z, a, O_1 = i] \cdot p_i(z) \\
&= (1 - \gamma_a) p_a(z) + \sum_{i \neq a} 0 \times p_i(z) \\
&= (1 - \gamma_a) p_a(z). \qquad (3.10)
\end{aligned}
$$

And from (3.8) and (3.10) we can write the probability of non-detection as :

$$
\begin{aligned}
P[s_{t+1} = z + \delta_a \mid z, a] &= 1 - P[s_{t+1} = \perp \mid z, a] \\
&= 1 - (1 - \gamma_a) p_a(z). \qquad (3.11)
\end{aligned}
$$

The last component to complete the MDP of the search problem is the **reward function** R. We provide details about the reward function in Section 3.4.

3.4 Reward Function

In designing a search strategy, it is useful to first motivate the structure of the reward function. A search cost could, in principle, depend on complex factors like the travel time between a previous cell at time $t-1$ and the current cell at time t. However, such a formulation introduces path-dependency into the problem. This would require tracking the sensor's current position as part of the state, fundamentally making the search problem a combination of allocation and routing challenges, which is significantly more complex and often NP-hard.

To maintain a more structured problem, we adopt a standard simplification from search theory by making the following assumption:

Assumption 3 *The transition cost between any two cells is negligible.*

With Assumption 3 the cost depends only on the cell being searched (i at time t) regardless of the previous cell a_{t-1}. We therefore define c_i as the cost incurred for performing an independent search in cell i. This modeling choice allows us to separate the problem of where to search from the problem of how to get there, allowing us to focus solely on maximizing the chance of detection. On the other hand, the primary goal of the search problem is to locate

the target, which yields a terminal reward of 1, representing a success. Thus, we formalize the immediate reward of taking the action a as :

$$R(a) = \begin{cases} 1 - c_a & \text{if the target is found,} \\ -c_a & \text{otherwise.} \end{cases} \tag{3.12}$$

Since searching a can yield two possible outcomes, we work with the expected immediate reward considering the current search vector z. This is calculated by weighting each possible cost by the probability of its related event as follows:

$$\begin{aligned} R(z, a) &= \mathbf{E}\left[R(a) \mid z\right] \\ &= P[s_{t+1} = \perp \mid z, a] \cdot (1 - c_a) + P[s_{t+1} = z + \delta_a \mid z, a] \cdot (-c_a) \end{aligned}$$

Using (3.10) and (3.11) we have:

$$\begin{aligned} R(z, a) &= (1 - \gamma_a) p_a(z) \cdot (1 - c_a) + (1 - (1 - \gamma_a) p_a(z)) \cdot (-c_a) \\ &= (1 - \gamma_a) \cdot p_a(z) - c_a. \end{aligned} \tag{3.13}$$

Note: $R(\perp, a) = 0$

The cost c_a is crucial to define the search objective. A positive cost (i.e. $c_a > 0$) penalizes each search, and leads to policies that aim to locate the target as early as possible. This is crucial for time-sensitive search, such as SAR operations, where it is crucial to find people in distress as soon as possible. $c_a < 0$ leads to policies that aim to find the target later, which sounds unrealistic for real-world problems.

However, there are scenarios where the search goal is to find the optimal policy that **maximizes the overall probability of target detection within the time horizon** T, regardless of when the detection occurs. In such scenarios the primary concern is the ultimate success of the mission within a strict time frame—such as in searching for a submerged wreckage with an unlimited budget, the target's condition does not degrade over time and the cost incurred is not a concern—This goal is formally achieved by setting the search cost to zero in all cells (i.e., $c_a = 0$ for $a \in [n]$). Thus, the expected reward in this case simplifies to the

probability of finding the target at the current step given the current belief:

$$R(z, a) = (1 - \gamma_a) \cdot p_a(z). \tag{3.14}$$

3.5 Objective Function

Having the reward function, we can now formulate the DP objective function of the search problem. Thus, for a given prior p_0, we define the objective function $J_\pi(z_0)$ under a policy $\pi = [\mu_0, \cdots, \mu_{T-1}]$ as:

$$J_\pi(z_0) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{T-1} R\left(z_t, \mu_t(z)\right)\right]. \tag{3.15}$$

The expression (3.15) provides the expected net value of the policy π, which balances the total probability of detection against the cumulative search cost. An optimal policy π^* is the one that maximizes $J_\pi(z_0)$. This means it is the one that finds the most efficient search strategy. We can write this as:

$$J_{\pi^*}(z_0) = \max_{\pi \in \Pi} J_\pi(z_0). \tag{3.16}$$

3.6 Backward Recursion

To solve the problem defined in (3.16), we apply the principle of optimality from DP, formalized through Bellman equation (2.8), but we maximize the reward instead of minimizing the cost. This approach allows us to break down the search problem into a sequence of overlapping subproblems, which can be solved recursively backwards. Before we provide the backward formalization, let's formally define f that takes a state s_t and action a_t as input and outputs the next state:

$$f(z, a) = \begin{cases} z + \delta_a, & \text{if the target is still hidden after searching in } a \\ \perp, & \text{if the target is found after searching in } a. \end{cases} \tag{3.17}$$

Thus we can formalize the backward induction as follows:

$$J_t(\perp) = 0 \tag{3.18}$$

$$J_T(z) = 0 \tag{3.19}$$

$$J_t(z) = \max_{a\in[n]} \{R(z,a) + \mathbb{E}\left[J_{t+1}(f(z,a))\right]\}, \quad t = T-1\ldots,0, \tag{3.20}$$

where $J_t(z)$ represents the maximum expected reward to go from stage t onward, assuming the target has not been detected in previous stages. To find an explicit form of (3.20), we need to compute the expected value of $J_{t+1}(f(z,a))$.

$$\begin{aligned}\mathbb{E}\left[J_{t+1}(f(z,a))\right] = {} & P[f(z,a) = \perp \mid z,a] \cdot J_{t+1}(\perp) \\ & + P[f(z,a) = z + \delta_a \mid z,a] \cdot J_{t+1}(z+\delta_a).\end{aligned} \tag{3.21}$$

However, $J_{t+1}(\perp) = 0$, and from (3.11) we know that

$$P[f(z,a) = z + \delta_a \mid z,a] = 1 - (1-\gamma_a)p_a(z).$$

Therefore, we have:

$$\mathbb{E}\left[J_{t+1}(f(z,a))\right] = (1 - (1-\gamma_a)p_a(z)) \cdot J_{t+1}(z+\delta a). \tag{3.22}$$

Substituting (3.13) and (3.22) into Equation (3.20), we have:

$$J_t(z) = \max_{a\in[n]} \{(1-\gamma_a)p_a(z) - c_a + (1-(1-\gamma_a)p_a(z)) \cdot J_{t+1}(z_+\delta a)\}, \quad t < T. \tag{3.23}$$

Algorithm 1 outlines the procedure to derive the policy $\pi = \{\mu_0, \ldots, \mu_{T-1}\}$, which maximizes the chance of a successful search given the prior distribution p_0 and the initial search vector $z_0 = [0, \ldots, 0]$.

Algorithm 2 performs a forward traversal in the policy dictionary generated with Algorithm 1 to extract the plan that optimally solve the problem. This algorithm starts from the initial state z_0 and iteratively looks up the best action for the current state, records it in a list, and transitions to the next state until the search horizon is reached. The extracted path serves

Algorithm 1 Backward Recursion for Dynamic Programming

Input:

- T: Time horizon (total number of search stages).
- n: Number of cells (the size of the action space).
- p_0: Prior probability vector, $p_0 \in \mathbb{R}^n$ with $p_{0,i} \geq 0$ for $i \in [n]$, and $\sum_i p_{0,i} = 1$.
- γ: Sensor detection rates vector, $\gamma \in \mathbb{R}^n$ with $\gamma_i \in [0, 1], \forall i \in [n]$.

Data Structures & Initialization:

- Define the action set $\mathcal{A} = \{1, 2, \ldots, n\}$.
- Initialize two global dictionaries:
  - $J_{values} =$ to store the value function for each stage.
  - $Policy =$ to store the optimal action for each stage.
- Define the state space for the final stage, $S_T = \{z \in [T]^n \mid \sum_{i=1}^n z_i = T\}$.
- Create a dictionary for the final stage value function: $J_T = \{\}$.
- **For each** state $z \in S_T$ **do**:
  - $J_T[z] = 0$ ▷ The value is zero at the end of the time horizon.
- $J_{values}[T] = J_T$.

Backward Recursion:

1. **For** $t = T - 1$ down to 0 **do**:

   1.1 Initialize empty dictionaries for the current stage: $J_t = \{\}$, $\mu_t = \{\}$.

   1.2 Define the state space for the current stage, $S_t = \{z \in \mathbb{Z}^n_{\geq 0} \mid \sum_{i=1}^n z_i = t\}$.

   1.3 **For each** state $z \in S_t$ **do**:

   - Compute the posterior probability vector $p(z)$ based on the search history z.
   - Find the best action and its value:

$$J_t[z] = \max_{a \in \mathcal{A}} \left[(1 - \gamma_a)p_a(z) - c_a + (1 - (1 - \gamma_a)p_a(z)) \cdot J_{values}[t + 1][z + \delta_a]\right]$$

$$\mu_t[z] = \arg\max_{a \in \mathcal{A}} \left[(1 - \gamma_a)p_a(z) - c_a + (1 - (1 - \gamma_a)p_a(z)) \cdot J_{values}[t + 1][z + \delta_a]\right]$$

   1.4 Store the completed dictionaries for the current stage in the global dictionaries:

   - $J_{values}[t] = J_t$
   - $Policy[t] = \mu_t$

Output:

- J_{values}: A dictionary where keys are time stages $t \in [0, T]$. Each value $J_{values}[t]$ is a dictionary mapping a state vector z to its optimal expected cumulative reward (or value).
- $Policy$: A dictionary where keys are time stages $t \in [0, T - 1]$. Each value $Policy[t]$ is a dictionary mapping a state vector z to the optimal action (cell to search) $\mu_t(z)$.

as a plan that can guide the search, suggesting which cell to search sequentially. In practice, the search either ends early with a successful detection before reaching the time limit T, or it runs for the full horizon without success.

Algorithm 2 Optimal Plan Reconstruction (Forward Pass)

Input:

- T: Time horizon (total number of search stages).
- *Policy*: from Algorithm 1.

Initialization:

- Initialize the current state vector: $z = [0, 0, \ldots, 0] \in \mathbb{R}^n$. ▷ At $t = 0$, no searches have occurred.
- Initialize an empty list to store the sequence of actions: `optimal_plan` $= [\,]$.

Path Reconstruction:

1. **For** $t = 0$ up to $T - 1$ **do**:
   - 1.1 **Look up the optimal action** for the current state z:
     - $a = Policy[t][z]$
   - 1.2 **Record the action** in the path:
     - Append a to `optimal_plan`.
   - 1.3 **Update the state** to transition to the next stage:
     - $z = z + \delta_a$ ▷ Increment the search count for the cell that was just searched.

Output:

- `optimal_plan`: A list of length T containing the sequence of optimal cells to search up to T.

3.7 Simulation and Validation

In this section we show an illustrative problem where we apply DP to identify the optimal policy that solves the problem. We consider a problem with negligible costs (i.e., $c_a = 0$). This allows us to encode the suggested plan extracted from DP as an optimal search vector z^* since the specific order of searches does not affect the total reward when costs are zero.

Problem Definition

A mobile robot is tasked with locating a misplaced item within a floor of a building of 10 offices. The robot is allowed to perform 10 independent searches across these offices. The

robot's visual sensory system is imperfect, and the probability of failing to detect the item varies between offices due to differing levels of clutter. These miss-detection rates are given by the vector:

$$\gamma = [0.2, 0.25, 0.32, 0.15, 0.35, 0.2, 0.3, 0.23, 0.27, 0.1].$$

Additionally, the robot is provided with an initial prior probability distribution, p_0, representing the belief that the item is in each office:

$$p_0 = [0.02, 0.2, 0.05, 0.1, 0.01, 0.03, 0.1, 0.3, 0.09, 0.1].$$

Given these constraints, the objective is to determine the optimal allocation of 10 searches to maximize the probability of finding the item.

Solution and Discussion

The problem can be modeled with a discrete search horizon $n = 10$ search cells, and a search horizon of $T = 10$. By applying the DP solver from Algorithm 1, we determined the optimal plan that proposes how to sequentially use T across the n offices. This suggested plan was further condensed to the allocation vector:

$$z^* = [0, 2, 1, 1, 0, 1, 1, 2, 1, 1]$$

This allocation results in a maximum theoretical probability of detection of 84.03%. To validate this, we ran experiments over 100 random seeds, with 100 search episodes per seed to average their overall outcomes. At the beginning of each of these search episodes, the target's location was sampled from the prior distribution. The experiment yielded an average success rate of $83.76 \pm 0.04\%$, which validates the theoretically computed detection probability mentioned earlier.

This result demonstrates the DP algorithm's ability to systematically process the prior beliefs and sensor characteristics to produce a non-intuitive but optimal search policy.

3.8 Summary

In this chapter, we presented a classic method for solving the single-object search problem in a discrete setting. By combining the principles of DP and Bayesian inference, we developed a complete framework for computing an optimal search policy. This policy specifies the best action to take at any search stage to maximize the chance of a successful search within

the remaining search horizon. However, the policy is built under key assumptions such as imperfect sensors, no false alarms, and negligible costs.

A central insight of our approach was the simplification of the search history. We demonstrated that instead of tracking the entire sequence of past searches $[a_1, a_1, \ldots, a_{t-1}]$, all relevant information for future decisions is captured by the simpler search count vector, z. This is because the posterior belief about the target's location depends only on how many times each cell has been searched, not the specific search order. Using this z as a sufficient statistic for the state allows us to formalize the problem with DP.

The proposed framework is flexible, capable of generating policies for different strategic objectives by modifying the reward function. It can produce policies that aim to find the target as quickly as possible (by penalizing search effort), or those that aim only to maximize the final probability of detection within the given horizon, in which case the search effort can be split into a effort allocation vector determining how many times each cell is searched. We validated our method on a practical example, showing that the theoretical probability of success computed by the DP solver closely matched the empirical success rate obtained from simulating the optimal policy.

While this DP approach provides a provably optimal solution, its main drawback is its computational complexity; the number of possible z_t states can become very large for bigger problems. We provide in Appendix A a more efficient memorization method that significantly improves the use of memory, by storing the index of every possible state vector instead of the full state. However, this strategy remains inefficient for very large problem.

The true value of this single-object model is its role as a foundation. It has introduced all the core concepts—states, belief updates, and the value iteration framework—that we will now expand upon in the next chapter to solve a more complex and realistic scenario: the two-object search problem.

CHAPTER 4 TWO-OBJECT SEARCH DYNAMIC PROGRAMMING

This chapter introduces DP as a method to solve search problems involving two related objects in the same search environment. In addition to the primary target, there is a hidden related object whose exact location is also unknown to the searcher. The main objective is to leverage that shared information to identify the policy that maximizes the chance of a successful search.

4.1 Two-Object Search Problem

We consider a finite search horizon $T \in \mathbb{N}$ and a search environment subdivided into a finite set of n cells. There is a target in the location $O_1 = i$, with $i \in [n]$, which We aim to find within the horizon T. However, there is a second object hidden in the location $O_2 = j$, with $j \in [n]$. The second object's location is related to the target's location. Both objects are stationary but randomly located across these n cells. They are not necessarily hidden in the same cell, but their possible locations are correlated. A practical example is two lost hikers, where one might remain at a shelter (perhaps due to injury) while the other leaves to find help. In this work we model the objects' correlation as an $n \times n$ joint probability distribution matrix p_0:

$$p_0 = \begin{bmatrix} p_{0,11} & p_{0,12} & \cdots & p_{0,1n} \\ p_{0,21} & p_{0,22} & \cdots & p_{0,2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0,n1} & p_{0,n2} & \cdots & p_{0,nn} \end{bmatrix},$$

where $p_{0,ij} \geq 0$ is the probability that the target is in cell $i \in [n]$ (i.e., the row), and the related object in cell $j \in [n]$ (i.e., the column).

We consider a scenario where the search can only happen in one cell at a time. Thus, various events can occur from searching in a given cell a. If both objects are found, or only the target is found, the system transitions directly to the special terminal state $\perp$ defined in Chapter 3 because of Assumption 1. It is also possible that none of the objects are found after searching cell a, or only the related object is found. In these situations, the search continues as long as the search horizon is not yet exhausted. For these two latter cases, we define the state s_t at time t as the combination of the history $h_t = [a_0, \ldots, a_{t-1}]$ which stores the sequence of cells that have already been visited, and a variable $\theta \in \{0\} \cup [n]$ which indicates the cell where the related object has been located, with $\theta = 0$ indicating that both objects are still hidden.

Hence, we denote the state at time t by $s_t \in [n]^t \times \{0, \ldots, n\} \cup \{\perp\}$.

Similar to Chapter 3, we consider an imperfect search process, as the sensor's capability can be affected by factors like environmental clutter within a cell or the unique physical signature of each object. We model this real-world constraint using misdetection probabilities. Formally, for each cell i, we define the probability of the sensor failing to detect the target and the related object, given they are present, as:

$$\gamma_{1,i} = P[\text{No target detection in } i \mid O_1 = i]$$

and

$$\gamma_{2,i} = P[\text{No related object detection in } i \mid O_2 = i].$$

We also make the following assumption :

Assumption 4 *Objects' detections are independent given their respective locations.*

Assumption 4 simply means failing to detect one object doesn't influence the sensor's ability to detect the other object given their respective locations. So, we quantify the probability of no object detection in $a_{0:t-1}$ given both objects' locations as:

$$\begin{aligned} &P[\text{No detection in } h_t \mid O_1 = i, O_2 = j] \\ &\quad = \prod_{\tau=0}^{t-1} P[\text{No detection in } a_\tau \mid O_1 = i, O_2 = j]. \end{aligned} \tag{4.1}$$

The expression in (4.1) quantifies how likely the search history (no detections) is, given the objects' true locations, which is precisely the information needed to update the belief about where the objects might now be. We need to determine this because it serves as the likelihood term within the Bayesian update rule to calculate the posterior belief of the objects' locations after a sequence of unsuccessful searches.

Let us first consider the case where both objects are still hidden (i.e., $\theta = 0$). This implies that for the entire history h_t, the outcome of every search was "no objects detected". Let's consider $\tau \in \{0, \ldots, t-1\}$, the probability of this outcome when searching a single cell a_τ given the objects are at locations i and j, is:

$$P[\text{No detection in } a_\tau \mid O_1 = i, O_2 = j] = \begin{cases} \gamma_{1,i}\gamma_{2,j} & \text{if } a_\tau = i = j \\ \gamma_{1,i} & \text{if } a_\tau = i \text{ and } a_\tau \neq j \\ \gamma_{2,j} & \text{if } a_\tau \neq i \text{ and } a_\tau = j \\ 1 & \text{if } a_\tau \neq i \text{ and } a_\tau \neq j. \end{cases} \tag{4.2}$$

On the right side of (4.1), we use the product over the entire search history because we consider the search history to be conditionally independent given objects' locations. So, for a pair of locations i, j, the probability of not finding anything in h_t can be expressed as :

$$P[\text{No detection in } h_t \mid O_1 = i, O_2 = j] = \gamma_{1,i}^{z_i} \cdot \gamma_{2,j}^{z_j}, \tag{4.3}$$

where $z_k \geq 0$ is the number of times the cell $k \in [n]$ appears in h_t.

With the state space, prior probability distribution, and detection model now defined, we have all the essential components to formalize the search problem. The information gained at each search stage—whether an object is found or not—allows the searcher to refine their belief about the objects' locations. In the next section we detail this process by deriving the Bayesian equations used to update the belief of the objects' locations after each observation.

4.1.1 Belief Update

Since the search is guided by probabilities, it is essential to update the belief about the objects' location based on the outcome of each individual search. An unsuccessful search provides valuable information: it lowers the probability that objects are in the searched cell and, consequently, increases the probability that they are located elsewhere. To make optimal decisions at each step, the searcher must incorporate this new information to refine their knowledge of the objects' likely locations.

Bayesian inference is the standard way to perform this update. To apply Bayes' rule, we need the prior belief (that we are given before the search starts) and the likelihood of the observation (in this case, the observation is "no detection"). Fortunately, we determined this likelihood term in (4.1). We can now calculate the posterior distribution (perform the belief

update) after $t-1$ unsuccessful searches. So, applying Bayes' rule, we have:

$$\begin{aligned} &P[O_1 = i, O_2 = j \mid \text{No detection in } h_t] \\ &\quad = \frac{P[\text{No detection in } h_t \mid O_1 = i, O_2 = j]}{P[\text{No detection in } h_t]} \cdot P[O_1 = i, O_2 = j]. \end{aligned} \tag{4.4}$$

Using the law of total probability, we can further expand the denominator on the right side of (4.4) as follows:

$$\begin{aligned} &P[O_1 = i, O_2 = j \mid \text{No detection in } h_t] \\ &\quad = \frac{P[\text{No detection in } h_t \mid O_1 = i, O_2 = j] \cdot P[O_1 = i, O_2 = j]}{\sum_{\alpha=1}^{n} \sum_{\beta=1}^{n} P[\text{No detection in } h_t \mid O_1 = \alpha, O_2 = \beta] \cdot P[O_1 = \alpha, O_2 = \beta]}. \end{aligned} \tag{4.5}$$

Using (4.3), we can express (4.5) as:

$$P[O_1 = i, O_2 = j \mid \text{No detection in } h_t] = \frac{\gamma_{1,i}^{z_i} \cdot \gamma_{2,j}^{z_j} \cdot p_{0,ij}}{\sum_{\alpha=1}^{n} \sum_{\beta=1}^{n} \gamma_{1,\alpha}^{z_\alpha} \cdot \gamma_{2,\beta}^{z_\beta} \cdot p_{0,\alpha\beta}}. \tag{4.6}$$

Hence, (4.6) provides the new belief at each stage $t \in [T]$ of both objects' locations, knowing that they are still hidden. Also, similarly to Chapter 3, if we consider the vector $z = [z_1, \cdots, z_n]$ that tracks how many times each cell has been visited in h_t, we can define from (4.7) the posterior distribution of objects' locations given a search vector z as:

$$p_{ij}(z) = \frac{\gamma_{1,i}^{z_i} \cdot \gamma_{2,j}^{z_j} \cdot p_{0,ij}}{\sum_{\alpha=1}^{n} \sum_{\beta=1}^{n} \gamma_{1,\alpha}^{z_\alpha} \cdot \gamma_{2,\beta}^{z_\beta} \cdot p_{0,\alpha\beta}}. \tag{4.7}$$

We emphasize that the probability computed in (4.7) concerns only sequences where both objects are still hidden, in other words it concerns states of the form $s_t = (h_t, 0)$ since we still ignore the location of the related object. However, detecting only the related object can occur at any search stage, which leads to states of the form $s_t = h_t, \theta)$, with $\theta \in [n]$. In this case, the problem reduces to a single-object search problem, the target, which we discussed in Chapter 3. Thus, subsequent searches will be guided by the conditional posterior distribution

$$p_{.|\theta}(z) = [p_{1|\theta}(z), \cdots, p_{n|\theta}(z)],$$

where $p_{i|\theta}(z)$ indicates the conditional probability of finding the target in cell i given that

the related object was found in cell θ with the search vector z.

Let $P[O_1 = i \mid \text{No target detection in } h_t, O_2 = \theta]$ be the posterior probability that the target might be in cell i. So, using the Bayes' rule we have:

$$\begin{aligned} &P[O_1 = i \mid \text{No target detection in } h_t, O_2 = \theta] \\ &\quad = \frac{P[\text{No target detection in } h_t \mid O_1 = i, O_2 = \theta]}{P[\text{No target detection in } h_t \mid O_2 = \theta]} \cdot P[O_1 = i \mid O_2 = \theta]. \end{aligned} \tag{4.8}$$

Using the law of total probability we can further expand the denominator on the right side of (4.8) as follows:

$$\begin{aligned} &P[O_1 = i \mid \text{No target detection in } h_t, O_2 = \theta] \\ &\quad = \frac{P[\text{No target detection in } h_t \mid O_1 = i, O_2 = \theta] \cdot P[O_1 = i \mid O_2 = \theta]}{\sum_{j=1}^{n} P[\text{No target detection in } h_t \mid O_1 = j, O_2 = \theta] \cdot P[O_1 = j \mid O_2 = \theta]}. \end{aligned} \tag{4.9}$$

From (4.1) and Assumption 4 we conclude that

$$P[\text{No target detection in } h_t \mid O_1 = i, O_2 = \theta_2] = \gamma_{1,i}^{z_i},$$

if we consider the search vector z. Thus, $P[O_1 = i \mid \text{No target detection in } h_t, O_2 = \theta]$ is equivalent to $p_{i|\theta}(z)$ that we formalize from (4.9) as :

$$p_{i|\theta}(z) = \frac{\gamma_{1,i}^{z_i} \cdot p_{0,i|\theta}}{\sum_{\alpha=1}^{n} \gamma_{1,\alpha}^{z_\alpha} \cdot p_{0,\alpha|\theta}}. \tag{4.10}$$

Let's define $p_{0,.\theta} = \sum_{i=1}^{n} p_{0,i\theta}$. So, with the Bayes' rule we have:

$$p_{0,i|\theta} = \frac{p_{0,i\theta}}{p_{0,.\theta}}.$$

So we can further simplify (4.10) as follows:

$$P_{i|\theta}(z) = \frac{\gamma_{1,i}^{z_i} \cdot \frac{p_{0,i\theta}}{p_{0,.\theta}}}{\sum_{\alpha=1}^{n} \gamma_{1,\alpha}^{z_\alpha} \cdot \frac{p_{0,\alpha\theta}}{p_{0,.\theta}}} = \frac{\gamma_{1,i}^{z_i} \cdot p_{0,i\theta}}{\sum_{\alpha=1}^{n} \gamma_{1,\alpha}^{z_\alpha} \cdot p_{0,\alpha\theta}}. \tag{4.11}$$

We can notice that the expression in (4.11) is similar to the standard single-object update formula from (3.7). This allows us to apply the single-object DP solver we discussed in 3 once the related object is detected.

Furthermore, the posterior in (4.7) and (4.11) depends only on the number of times each cell has been visited, not on the specific order. Hence, similarly to Chapter 3, we can represent the search history h_t by a vector $z = [z_1, \cdots, z_n]$, where $z_i \geq 0$ is the number of times each cell i appears in h_t. Therefore, a state can be fully represented by (z, θ) without loss of information. Thus, the complete state space is

$$\mathcal{S} = \left\{ z \mid z \in \mathbb{N}^n, \sum_{i=1}^{n} z_i = t, \text{ for some } t \leq T \right\} \times \{0, 1, \cdots, n\} \cup \{\perp\}.$$

Now that we have a more efficient state representation, (z, θ), which captures all necessary information from the search history without needing to track the specific sequence of actions. We formulate the state space dynamics in the next section.

4.1.2 States Transition

The **transition model** $\mathcal{T}$ describes the probabilities of moving from one state to its potential next states after taking a specific action. Similar to Chapter 3, we define δ_a as a vector of zeros with a 1 at the index corresponding to the action a. After selecting an action a, the search history vector z is updated to a new vector $z + \delta_a$ by incrementing by 1 the component at index a in z, where "+" denotes element-wise addition.

We consider two categories of states that influence the state transitions, besides the special state $\perp$. The **first category** includes states where both objects are still hidden (i.e. states of the form $s_t = (z, 0)$). Taking an action a in this category leads to one of three possible

states: $(z+\delta a, 0)$, $(z+\delta_a, a)$ or $\perp$. The probabilities of these transitions should satisfy the following total probability law:

$$P[s_{t+1}=(z+\delta_a,0)\mid(z,0),a]+P[s_{t+1}=(z+\delta_a,a)\mid(z,0),a]+P[s_{t+1}=\perp\mid(z,0),a]=1 \quad (4.12)$$

We use the law of total probability to express each probability on the left side of (4.12) in terms of p_0, z, γ_1 and γ_2.

Let us start with $P[s_{t+1}=(z+\delta a,0)\mid(z,0),a]$. We have the following :

$$\begin{aligned}P[s_{t+1}=(z+\delta a,0)\mid(z,0),a]=\sum_{i=1}^{n}\sum_{j=1}^{n}&P[s_{t+1}=(z+\delta a,0)\mid(z,0),a,O_1=i,O_2=j]\\&\cdot P[O_1=i,O_2=j\mid(z,0),a]. \quad (4.13)\end{aligned}$$

Since the choice of the action, by itself, does not provide new information, we conclude that

$$\begin{aligned}P[O_1=i,O_2=j,\mid(z,0),a]&=P[O_1=i,O_2=j\mid(z,0)]\\&=P[O_1=i,O_2=j,\mid\text{No detection in } a_{0:t-1}],\end{aligned}$$

which is $p_{ij}(z)$ we developed in (4.7). Additionally,

$$P[s_{t+1}=(z+\delta a,0)\mid(z,0),a,O_1=i,O_2=j]$$

is simply the probability of no object detection in a given the current $t-1$ unsuccessful searches, and that $O_1=i$ and $O_2=j$. Furthermore, there are four scenarios in (4.13): (1) $a=i$, the cell containing the target and not the related object; (2) $a=j$, the cell containing the related object but not the target; (3) $a=i=j$, the cell containing both objects; and (4) $a\neq i\neq j$, which means a doesn't contain any object. So, using (4.2), we have :

$$\begin{aligned}P[s_{t+1}=(z+\delta a,0)\mid(z,0),a]&=\sum_{i=1}^{n}\sum_{j=1}^{n}\underbrace{P[\text{No detection in } a\mid O_1=i,O_2=j]}_{\text{Likelihood}}\cdot\underbrace{p_{ij}(z)}_{\text{Belief}}\\&=\underbrace{(\gamma_{1,a}\gamma_{2,a})\cdot p_{aa}(z)}_{O_1=a,O_2=a}+\underbrace{\gamma_{1,a}\sum_{j\neq a}p_{aj}(z)}_{O_1=a,O_2\neq a}\\&\quad+\underbrace{\gamma_{2,a}\sum_{i\neq a}p_{ia}(z)}_{O_1\neq a,O_2=a}+\underbrace{\sum_{i\neq a}\sum_{j\neq a}p_{ij}(z)}_{O_1\neq a,O_2\neq a}. \quad (4.14)\end{aligned}$$

Thus (4.14) gives us the probability to transition to a state where both objects are still hidden at stage $t+1$ after searching in cell a.

Next, we need to compute the probability to transition to a state where the related object is found after searching in cell a_t. For this, we consider all possibilities for where the target might be, because the next state s_{t+1} can be $(z+\delta_a, a)$ only if a is the true location of the related object (i.e., $O_2 = a$), which implies that the probability that $O_2 = j$ for any cell $j \neq a$ is zero. We have:

$$\begin{aligned} P[s_{t+1} = (z+\delta a, a) \mid (z,0), a] = \sum_{i=1}^{n} & P[s_{t+1} = (z+\delta a, a) \mid (z,0), a, O_1 = i, O_2 = a] \\ & \cdot P[O_1 = i, O_2 = a, \mid (z,0), a]. \end{aligned} \tag{4.15}$$

Similar to the previous category, we establish that

$$\begin{aligned} P[O_1 = i, O_2 = a, \mid (z,0), a] &= P[O_1 = i, O_2 = a, \mid (z,0)] \\ &= P[O_1 = i, O_2 = a, \mid \text{No detection in } a_{0:t-1}] \\ &= p_{ia}(z). \end{aligned}$$

We also note that $P[s_{t+1} = (z+\delta a, a) \mid (z,0), a, O_1 = i, O_2 = a]$ is simply the probability to transition to a state s_{t+1} where only the related object is detected after searching in cell a. Thus, we have:

$$\begin{aligned} P[s_{t+1} = (z+\delta a, a) \mid (z,0), a] &= \sum_{i=1}^{n} P[\text{Find only the related object in } a \mid O_1 = i, O_2 = a] \cdot p_{ia}(z) \\ &= \underbrace{\gamma_{1,a}(1-\gamma_{2,a})p_{aa}(z)}_{O_1 \text{ is also in } a} + \underbrace{\sum_{i \neq a}(1-\gamma_{2,a})p_{ia}(z)}_{O_1 \text{ is elsewhere}} \\ &= (1-\gamma_{2,a})\left(\gamma_{1,a}p_{aa}(z) + \sum_{i \neq a} p_{ia}(z)\right). \end{aligned} \tag{4.16}$$

Now we can establish $P[s_{t+1} = \perp \mid (z,0), a]$ based on (4.12), (4.14) and (4.16). We have :

$$
\begin{aligned}
P[s_{t+1} = \perp \mid (z,0), a] &= 1 - \Bigg((\gamma_{1,a}\gamma_{2,a}) \cdot p_{aa}(z) + \gamma_{1,a} \sum_{j \neq a} p_{aj}(z) + \gamma_{2,a} \sum_{i \neq a} p_{ia}(z) \\
&\quad + \sum_{i \neq a} \sum_{j \neq a} p_{ij}(z) + (1 - \gamma_{2,a}) \left(\gamma_{1,a} p_{aa}(z) + \sum_{i \neq a} p_{ia}(z) \right) \Bigg) \\
&= 1 - \left(\gamma_{1,a} \sum_{j \neq a} p_{aj}(z) + \sum_{i \neq a} \sum_{j \neq a} p_{ij}(z) + \gamma_{1,a} p_{aa}(z) + \sum_{i \neq a} p_{ia}(z) \right) \\
&= \quad 1 - \left(\gamma_{1,a} \left(\sum_{j \neq a} p_{aj}(z) + p_{aa}(z) \right) + \sum_{i \neq a} \sum_{j \neq a} p_{ij}(z) + \sum_{i \neq a} p_{ia}(z) \right) \\
&= \quad 1 - \left(\gamma_{1,a} \sum_{j=1} p_{aj}(z) + \sum_{i \neq a} \sum_{j=1} p_{ij}(z) \right). \qquad (4.17)
\end{aligned}
$$

However, with the law of total probability we have:

$$
\sum_{i \neq a} \sum_{j=1} p_{ij}(z) = 1 - \sum_{j=1} p_{aj}(z).
$$

Therefore, we have:

$$
\begin{aligned}
P[s_{t+1} = \perp \mid (z,0), a] = \quad & 1 - \left(\gamma_{1,a} \sum_{j=1} p_{aj}(z) + 1 - \sum_{j=1} p_{aj}(z) \right) \\
&= (1 - \gamma_{1,a}) \sum_{j=1} p_{aj}(z). \qquad (4.18)
\end{aligned}
$$

Hence, we have the probabilities of all state transitions given a state $s_t = (z, 0)$, and an action a.

The **second category** includes states where the precise location of the related object is known (i.e. states of the form $s_t = (z, \theta)$ with $\theta \neq 0$). This category leads to two possible next states after searching cell a: $(z + \delta a, \theta)$ and $\perp$, which should satisfy the following total probability relation :

$$
P[s_{t+1} = (z + \delta a, \theta) \mid (z, \theta), a] + P[s_{t+1} = \perp \mid (z, \theta), a] = 1. \qquad (4.19)
$$

We observe a similarity between (3.8) and (4.19). This shows that once the related object is located, future state transitions fall into a dynamic of single-object search problem. Thus, by analogy to (3.11), we can conclude that :

$$P[s_{t+1} = (z + \delta a, \theta) \mid (z, \theta), a] = 1 - (1 - \gamma_{1,a}) \cdot p_{a|\theta}(z). \tag{4.20}$$

And from (4.19) we have:

$$P[s_{t+1} = \perp \mid (z, \theta), a] = (1 - \gamma_{1,a}) \cdot p_{a|\theta}(z). \tag{4.21}$$

Now that we have the transition model, we can predict how the system evolves probabilistically in response to any search action. However, to determine the optimal sequence of actions, we need a way to measure the desirability of these transitions accounting for the cost each search action involves. We develop that in the next section.

4.1.3 Reward Function

The reward function for the two-object problem follows the same logic as in Section 3.4. The search cost, c_i depends only on the cell i. We therefore consider the immediate reward defined in (3.12). However, the expected reward in this case needs to account for all possible outcomes of an action a, which depend on whether the related object has been found (i.e., $\theta \neq 0$) or not (i.e., $\theta = 0$).

First, let's consider the case where **neither object has been found** (i.e, $s_t = (z, 0)$). The expected reward for taking action a is the sum of all possible outcomes weighted by their probabilities:

$$\begin{aligned} R((z,0),a) = {} & P[s_{t+1} = \perp \mid z, 0, a] \cdot (1 - c_a) \\ & + P[s_{t+1} = (z + \delta_a, 0) \mid z, 0, a] \cdot (-c_a) \\ & + P[s_{t+1} = (z + \delta_a, a) \mid z, 0, a] \cdot (-c_a). \end{aligned} \tag{4.22}$$

By factoring out $-c_a$ and using the law of total probability, (4.22) simplifies to:

$$\begin{aligned} R((z,0),a) &= P[s_{t+1} = \perp \mid z, 0, a] - c_a \\ &= (1 - \gamma_{1,a}) \sum_{j=1}^{n} p_{aj}(z) - c_a. \end{aligned} \tag{4.23}$$

Similar to the single-object case, setting the cost $c_a = 0$ corresponds to policies whose unique goal is to maximize the target detection probability within the given search horizon, without penalty for how long it takes. The expected reward for such problems is:

$$R((z,0),a) = (1 - \gamma_{1,a}) \sum_{j=1}^{n} p_{aj}(z). \tag{4.24}$$

For the second category of states, where the related object's location is known (i.e., $s_t = (z, \theta)$

with $\theta \neq 0$), we apply by analogy the expected reward from Section 3.4. The belief $p_a(z)$ is simply replaced by the conditional belief $p_{a|\theta}(z)$. For problems that aim to find the target as early as possible ($c_a > 0$), the expected reward is:

$$R((z,\theta),a) = (1-\gamma_{1,a})p_{a|\theta}(z) - c_a. \tag{4.25}$$

And for problems whose unique goal is to maximize the probability of detection within the given horizon ($c_a = 0$), the expected reward is:

$$R((z,\theta),a) = (1-\gamma_{1,a})p_{a|\theta}(z). \tag{4.26}$$

4.1.4 Objective Function

The objective in the two-object search problem remains consistent with the standard DP framework introduced in Chapter 3: to find a policy π that maximizes the total expected reward $J_\pi(s_0)$ over the finite horizon T, with $s_0 = (z_0, 0)$. Formally, this objective is given by:

$$J_\pi(s_0) = \underset{\pi}{E}\left[\sum_{t=0}^{T-1} R(s_t, \mu_t(s_t))\right], \tag{4.27}$$

where the state s_t is now the tuple (z,θ) representing the search history and the status of the related object, and R is the immediate reward function defined previously, incorporating the search cost c_a.

4.1.5 Backward Recursion

To solve the problem defined in (4.27), we proceed similarly to Section 3.6. However, the state transition dynamics, denoted by f, for this case are different, depending on the random outcome after searching in a_t, and the category of the states in the system: states with $\theta = 0$, and those with $\theta \neq 0$, and determine the backward recursion formulation for each case.

We start with the category of $s_t = (z, 0)$, (i.e., $\theta = 0$). We need to consider any of the following outcomes after selecting an action a: (1) only the related object is found (i.e., $O_2 = a$), (2) both objects stay hidden,or (3) the target is found (i.e., $O_1 = a$). Formally, we have:

$$f((z,0),a) = \begin{cases} (z+\delta a, 0), & \text{if no detection occurs} \\ (z+\delta a, a), & \text{if only the related object is found in } a \\ \perp, & \text{if the target is found in } a_t. \end{cases} \tag{4.28}$$

This allows us to write the backwards using the following formulation:

$$J_t(\perp) = 0, \text{for any } t \in [T] \tag{4.29}$$

$$J_T((z,0)) = 0 \tag{4.30}$$

$$J_t((z,0)) = \max_{a\in[n]} \mathbb{E}\left[R((z,0),a) + J_{t+1}(f((z,0),a))\right], \quad t = 0,1,\cdots,T-1. \tag{4.31}$$

Thus, to find an explicit form of (4.31) for $\theta = 0$, we need to formalize $J_{t+1}(f((z,0),a))$ as the average over the values of these three outcomes, weighted by their transition probabilities. For that we consider $s_{t+1} = f((z,0),a)$, and we have:

$$\begin{aligned}\mathbb{E}\left[J_{t+1}(f((z,0),a))\right] = {} & P[s_{t+1} = \perp \mid (z,0),a] \cdot J_{t+1}(\perp) \\ & + P[s_{t+1} = (z+\delta_a, 0) \mid (z,0),a] \cdot J_{t+1}((z_+\delta_a,0)) \\ & + P[s_{t+1} = (z+\delta_a, a) \mid (z,0),a] \cdot J_{t+1}((z+\delta_a,a)).\end{aligned} \tag{4.32}$$

However, $J_{t+1}(\perp) = 0$ and from (4.14), and (4.16) we have:

$$\begin{aligned}\mathbb{E}\left[J_{t+1}(f((z,0),a))\right] = {} & \left(\gamma_{1,a}\gamma_{2,a}p_{aa}(z) + \gamma_{1,a}\sum_{j\neq a} p_{aj}(z) + \gamma_{2,a}\sum_{i\neq a} p_{ia}(z) \right.\\ & \left. + \sum_{i\neq a}\sum_{j\neq a} p_{ij}(z)\right) J_{t+1}((z+\delta a, 0)) \\ & + (1-\gamma_{2,a})\left(\gamma_{1,a}p_{aa}(z) + \sum_{i\neq a} p_{ia}(z)\right) J_{t+1}((z+\delta a, a)).\end{aligned} \tag{4.33}$$

Substituting (4.24) and (4.33) into Equation (4.32), we have:

$$\begin{aligned}J_t((z,0)) = \max_{a_t\in[n]} \Bigg\{ & (1-\gamma_{1,a})\sum_{j=1}^{n} p_{aj}(z) - c_a + \left(\gamma_{1,a}\gamma_{2,a}p_{aa}(z) + \gamma_{1,a}\sum_{j\neq a} p_{aj}(z) \right.\\ & \left. + \gamma_{2,a}\sum_{i\neq a} p_{ia}(z) + \sum_{i\neq a}\sum_{j\neq a} p_{ij}(z)\right) \cdot J_{t+1}((z+\delta a, 0)) \\ & + (1-\gamma_{2,a})\left(\gamma_{1,a}p_{aa}(z) + \sum_{i\neq a} p_{ia}(z)\right) \cdot J_{t+1}((z+\delta a, a))\Bigg\}, \\ & \text{for } t < T,\end{aligned} \tag{4.34}$$

which computes the cost to go from any state where objects are still are hidden.

Now we need to establish the backward recursion for the second type of states, where we already know the location of the related object. This follows by analogy from Section 3.6. However, the state transition dynamics f for this specific case are defined as follows:

$$f((z,\theta),a) = \begin{cases} (z+\delta a,\theta), & \text{if no detection occurs in } a \\ \perp, & \text{if the target is found in } a \end{cases} \tag{4.35}$$

Thus, similarly to (3.23), and using (4.20), and (4.21), we can write the cost to go for states with $\theta \neq 0$, as follows:

$$\begin{aligned} J_T(z,\theta) &= 0 \\ J_t(z,\theta) &= \max_{a\in[n]} \Big\{ (1-\gamma_{1,a}) p_{a|\theta}(z) - c_a \\ &\qquad + \Big(1-(1-\gamma_{1,a})p_{a|\theta}(z)\Big)\cdot J_{t+1}(z+\delta a,\theta) \Big\}, \quad t<T, \end{aligned} \tag{4.36}$$

where the value of $J_t((z,\theta))$ represents the maximum expected reward from search stage t onward, given this valuable information.

Algorithm 3 outlines the procedure to derive the policy $\pi^* = \{\mu_0, \cdots, \mu_{T-1}\}$, that optimizes the expression in (4.27). The procedure incorporates Algorithm 1 to handle possible subsequent trajectories where the related object might be found in a cell $\theta \neq 0$.

In summary, the complete set of Bellman equations required to solve the two-object search problem are defined. The solution's recursive structure elegantly splits into two distinct forms: a complex, multi-outcome recursion for when both objects are hidden, and a simpler, single-object recursion that is initiated upon the discovery of the related object. These equations, while mathematically detailed, provide the formal foundation for DP algorithm that computes the provably optimal search policy.

4.2 Advantages of Correlational Search

The DP formulation derived provides the optimal solution for the two-object search problem, given the joint probability distribution. This optimal solution will always achieve an expected reward

Algorithm 3 Backward Recursion for Two-Object Search

Input:

- T: Time horizon (total number of search stages).
- n: Number of cells (the size of the action space).
- p_0: Prior joint probability distribution matrix, $p_0 \in \mathbb{R}^{n \times n}$.
- γ_1, γ_2: Sensor miss-detection rates vectors, $\gamma_1, \gamma_2 \in \mathbb{R}^n$.

Data Structures & Initialization:

- Define the action set $\mathcal{A} = \{1, 2, \ldots, n\}$.
- Initialize two global dictionaries: $J_{values} = \{\}$ and $Policy = \{\}$.
- Define the state space for the final stage, $S_T = \{(z, \theta) \mid z \in \mathbb{Z}^n_{\geq 0}, \sum z_i = T, \theta \in \{0, \ldots, n\}\}$.
- Create a dictionary for the final stage value function: $J_T = \{\}$.
- **For each** state $(z, \theta) \in S_T$ **do**:
  - $J_T[(z, \theta)] = 0$ ▷ Value is zero at the end of the time horizon.
- $J_{values}[T] = J_T$.

Backward Recursion:

1. **For** $t = T - 1$ down to 0 **do**:
   1.1 Initialize empty dictionaries for the current stage: $J_t = \{\}$, $\mu_t = \{\}$.
   1.2 Define the state space for the current stage, $S_t = \{(z, \theta) \mid z \in \mathbb{Z}^n_{\geq 0}, \sum z_i = t, \theta \in \{0, \ldots, n\}\}$.
   1.3 **For each** state $(z, \theta) \in S_t$ **do**:
   - **if** $\theta == 0$: ▷ Case 1: Both objects are hidden.
     - Compute the joint posterior matrix p_t based on history z using (4.7).
     - Find the best action and its value:

$$J_t[(z, 0)] = \max_{a \in \mathcal{A}} \{\ldots\} \quad \triangleright \text{ Using (4.34)}$$
$$\mu_t[(z, 0)] = \arg\max_{a \in \mathcal{A}} \{\ldots\}$$

   - **else**: ▷ Case 2: O_2 has been found in cell θ_2.
     - Compute the conditional posterior vector $p_{t,.|\theta}$ based on history z and the prior $p_{0,.|\theta}$.
     - Find the best action and its value:

$$J_t[(z, \theta)] = \max_{a \in [n]} \{\cdots\} \quad \triangleright \text{ Using (4.36)}$$
$$\mu_t[(z, \theta)] = \arg\max_{a \in \mathcal{A}} \{\cdots\}$$

   - Store the results: $J_{values}[t] = J_t$, $Policy[t] = \mu_t$.

Output:

- J_{values}: A dictionary mapping each time stage t to a dictionary where keys are state tuples (z, θ) and values are the optimal expected detection probabilities.
- $Policy$: A dictionary mapping each time stage t to a dictionary where keys are state tuples (z, θ) and values are the optimal actions $\mu_t((z, \theta))$.

greater than or equal to the optimal policy derived from only the target's marginal distribution (the single-object approach). However, solving the two-object DP is significantly more computationally complex. Therefore, it is valuable to identify conditions under which the magnitude of the performance gap between the two-object and single-object solutions is enough to justify the increased computational effort. While many factors can influence this gap, we highlight two particularly intuitive conditions related to information and detectability.

4.2.1 Information Gain from the Related Object

The potential benefit of the two-object strategy increases if finding the related object substantially reduces uncertainty about the target. We can quantify this using Shannon Entropy. Let p_0^{target} be the target's initial marginal distribution. The initial uncertainty is its entropy:

$$H(p_0^{\text{target}}) = -\sum_{i=1}^{n} p_{0,i}^{\text{target}} \log_2 p_{0,i}^{\text{target}}. \tag{4.37}$$

If $O_2 = j$, the remaining uncertainty about the target is the entropy of the conditional distribution $p_{0,\cdot|j}$, denoted $H(p_{0,\cdot|O_2=j})$:

$$H(O_1 \mid O_2 = j) = -\sum_{i=1}^{n} p_{0,i|j} \log_2 p_{0,i|j}. \tag{4.38}$$

Since we don't know where the related object might be found, we consider the expected conditional entropy, averaging over all possibilities for j, weighted by the marginal probability $p_{0,j}^{\text{related}}$ of the related object being in cell j:

$$\mathbb{E}[H(O_1|O_2)] = \sum_{j=1}^{n} p_{0,j}^{\text{related}} H(O_1 \mid O_2 = j). \tag{4.39}$$

A significant information gain occurs when the expected uncertainty after finding the related object is much lower than the initial uncertainty. This suggests a condition where the two-object model is likely to offer a substantial performance advantage:

Condition 1 (Significant Information Gain) $H(p_0^{target}) \gg \mathbb{E}[H(O_1|O_2)]$.

While the two-object DP policy will always leverage whatever information is available, the practical benefit over a simpler single-object search is most pronounced when finding the related object provides a large reduction in uncertainty about the target.

4.2.2 Relative Detectability

Another factor that can amplify the advantage of the two-object strategy is the relative ease of detecting the two objects. If the related object is significantly easier to detect than the target in many cells, the information gain discussed above can be acquired more quickly and with less effort. This suggests a second condition contributing to a larger performance gap:

Condition 2 (Relative Detectability) $\gamma_{2,i} \ll \gamma_{1,i}$ *for many* i

When the related object is much easier to find, the optimal policy can more effectively exploit the correlation. It might prioritize searching locations where the related object is likely and easy to detect, even if the initial probability for the target there is lower. Finding the easily detectable related object then provides valuable information (due to Condition 1), enabling a more targeted and efficient subsequent search for the harder-to-find target.

It is important to note that these two conditions are neither exhaustive nor strictly necessary for the two-object policy to outperform the single-object one. The optimal two-object DP solution inherently considers all available information and will always yield a result at least as good as the optimal single-object policy. However, Conditions 1 and 2 highlight intuitive scenarios where the magnitude of this improvement is expected to be most significant, justifying the use of the more complex model. Other factors, such as specific structural patterns in the joint distribution or the relative costs of searching different cells, can also influence the performance difference.

4.3 Practical Problem Definition

In this section, we define a practical scenario to empirically evaluate and compare the performance of the single-object and two-object search policies. The problem is specifically designed to satisfy the key conditions for strategic advantage previously identified: **Condition 1** (Information Gain) and **Condition 2** (Relative Detectability).

4.3.1 Problem

A team of technicians is preparing for a series of experiments across five rooms. They have misplaced a crucial **sensor probe** and, to save time, have tasked an autonomous robot with finding it.

During previous work, the technicians often carried the probe on a **large-wheeled medical cart**, which has also been misplaced. The team provides the robot with a joint probability distribution, p_0, representing their belief about the possible locations of both the probe and the cart:

$$p_0 = \begin{bmatrix} 0.152 & 0.0039 & 0.003 & 0.0108 & 0.011 \\ 0.0038 & 0.0052 & 0.117 & 0.0162 & 0.165 \\ 0.0057 & 0.195 & 0.015 & 0.009 & 0.011 \\ 0.0038 & 0.0091 & 0.0075 & 0.027 & 0.011 \\ 0.0247 & 0.0468 & 0.0075 & 0.117 & 0.022 \end{bmatrix}$$

The robot is constrained to a total of $T = 10$ search actions. Each search action consumes a certain amount of time and energy, which varies by room. For example, a small, simple room is quick to search, while a large, cluttered room requires more effort. We define the cost c for searching each room (as a normalized value between 0 and 1) as: $c = [0.15, 0.2, 0.25, 0.1, 0.2]$.

Due to its size, the medical cart is relatively easy for the robot's camera to detect, though its visibility varies with each room's clutter level. The corresponding miss-detection rates are given by $\gamma_2 = [0.2, 0.1, 0.25, 0.15, 0.2]$. Conversely, the small sensor probe is much harder to find, with higher miss-detection rates given by $\gamma_1 = [0.8, 0.65, 0.82, 0.75, 0.7]$.

The robot's objective is to determine the optimal search policy (which cell to search at each step) that maximizes the total expected reward within the $T = 10$ action limit. This policy must now intelligently balance the probability of finding the target against the cost incurred for each search.

4.3.2 Problem Analysis

We can notice that the problem satisfies Condition 2, as $\gamma_{1,i} > \gamma_{2,i}$ for all $i \in [n]$. This confirms that the related object is easier to detect.

To validate Condition 1, we must quantify the information gain finding the related object could provide on average. First, we compute the target marginal distribution p_0^{target} from the prior p_0:

$$p_0^{target} = \begin{bmatrix} 0.1807 \\ 0.3072 \\ 0.2357 \\ 0.0584 \\ 0.218 \end{bmatrix}.$$

The initial uncertainty about the target's location, measured as Shannon Entropy with (4.37), is:

$$H(p_0^{target}) = 2.1789 \text{ bits}$$

Next, we calculate the expected uncertainty that would remain after finding O_2, $H(P_{0,\cdot|\cdot})$. We first extract the matrix of conditional priors $p_{0,\cdot|\cdot}$ from p_0:

$$p_{0,\cdot|\cdot} = \begin{bmatrix} 0.8 & 0.015 & 0.02 & 0.06 & 0.05 \\ 0.02 & 0.02 & 0.78 & 0.09 & 0.75 \\ 0.03 & 0.75 & 0.10 & 0.05 & 0.05 \\ 0.02 & 0.035 & 0.05 & 0.15 & 0.05 \\ 0.13 & 0.18 & 0.05 & 0.65 & 0.10 \end{bmatrix}.$$

And the entropy of each conditional distributions are given as follows:

- $H(p_{0,\cdot|1}) = 1.02$
- $H(p_{0,\cdot|2}) = 1.13$
- $H(p_{0,\cdot|3}) = 1.157$
- $H(p_{0,\cdot|4}) = 1.59$
- $H(p_{0,\cdot|5}) = 1.29.$

We then use (4.39) to find the expected conditional entropy $H(O_1 \mid O_2)$, which gives us:

$$H(O_1 \mid O_2) = 1.2304 \text{ bits.}$$

Since, 1.2304 bits $<$ 2.1789 bits (i.e. $H(O_1 \mid O_2) < H(p_0^{target})$), we conclude that finding the related object provided a significant reduction in uncertainty, satisfying Condition 1. Based on this we could expect a significant difference in performance between the Single-Object Search and the Two-Object Search.

4.4 Simulations and Validation

The primary objective of this simulation is to demonstrate the performance benefit of the two-object search model over a conventional single-object approach using the problem defined in 4.3. We chose a single-object DP for the single object because, unlike methods in [8], modifying the reward is enough to test them in different scenarios. Let define $\vec{0} = [0, \ldots, 0]$, a vector of n zeros. We evaluated both solvers across two distinct configurations using the total expected reward $J_\pi(\vec{0})$ as reference.

We ran both solvers on the problem defined in Section 4.3.1, a single-object DP using the target's marginal distribution as its prior, and a two-object DP using the joint distribution as prior. For a fair comparison, we ran empirical simulations for each solver, averaging the results over 100 **different seeds**, with each seed representing 100 **search episodes**. At the beginning of each episode, the

true object locations were sampled from the prior distribution and remained static until the episode ended.

4.4.1 Detection Probability Maximization ($c = \vec{0}$)

In this scenario, we ignore search costs and focus only on the reward gained from finding the target within $T = 10$ steps. This turns the problem object into maximizing the detection probability.

Metric	**single-object policy**	**two-object policy**
Optimal Value (Theoretical)	0.5089	0.6989
Avg. Episode Reward	0.5109 ± 0.0483	0.6960 ± 0.0468
Avg. Success Rate	0.5109 ± 0.0483	0.6960 ± 0.0468
Avg. Episode Length	7.38 ± 0.3138	6.35 ± 0.3323

Table 4.1 Comparison between single-object and two-object DP solvers for detection probability maximization ($c = \vec{0}$) within $T = 10$ search steps.

The theoretical optimal value of the two-object policy is significantly higher (≈ 0.7) compared to the single-object's one (≈ 0.51). We also notice that the two-object DP achieves a significantly higher empirical average reward of 0.696, compared to just 0.5109 for the single-object policy. This large difference proves that paying attention to both objects helps the searcher to finding the target much more often. Additionally, the two-object policy is more efficient, earning this higher reward while taking fewer steps (6.35) on average than the single-object policy (7.38).

4.4.2 Mixed Objective (non-null Cost and Reward)

This section looks at the complete problem where the agent has to balance the reward of finding the target against the cost of searching for it.

Metric	**Single-Object DP**	**Two-Object DP**
Optimal Value (Theoretical)	-0.7464	-0.4601
Avg. Episode Reward	-0.7334 ± 0.0887	-0.4536 ± 0.0973
Avg. Success Rate	0.3337 ± 0.0465	0.6263 ± 0.0439
Avg. Episode Length	7.59 ± 0.3600	6.40 ± 0.3380

Table 4.2 Comparison between single-object and two-object DP solvers for a mixed objective that balances successful search against search cost.

Combining the two objective provided a similar result as the in the previous two cases. The results show that the two-object policy outperformed the single-object policy in reward theoretically (-0.4601 vs -0.7464) and empirically (-0.4536 vs -0.7334). The single-object policy often struggles to find the target, failing about two-thirds of the time (33.37% success rate). Because it fails so

often, its average reward is low. In contrast, the Two-Object agent uses the relationship between objects to succeed more often (62.63%) and faster (6.40 steps).

4.4.3 Experimental Summary

The comparison across these scenarios confirms that the two-object policy consistently outperforms the single-object policy by leveraging the informational value of the related object. By exploiting objects' correlation, the two-object policy achieves higher performance in optimizing search objectives.

4.5 Summary

This chapter extended the search problem to a more complex two-object scenario involving a single target and a related object. We developed a complete framework for computing an optimal search policy that leverages the probabilistic correlation between a primary target and a secondary, related object. The agent's initial belief is no longer a simple vector but an $n \times n$ joint probability matrix, which captures the relationship between the two objects' locations.

In addition to using a search vector z as the sufficient statistic to express the posterior belief, the two-object problem requires a more comprehensive state, $s_t = (z, \theta)$, which tracks not only the search counts but also the status (location, if found) of the related object.

The derived solution is inherently complete. It includes control laws for states with both objects still hidden and states where the related object's location is known. In practice, the search begins with controls that balance searching for the target with the potential information gain from finding the related object. If the related object is found, the framework elegantly transitions to controls that act as a more informed single-object search. We validated this more powerful approach through a practical example, demonstrating superior results compared to the single-object solver from Chapter 3. This was confirmed by empirical simulations, where the new approach was superior on average in reward, detection rate.

While this two-object DP approach provides a provably optimal and superior solution, its primary drawback is an even greater computational complexity. The expansion of the state and belief spaces makes the "curse of dimensionality" more acute than in the single-object case. The memorization method we discuss in Appendix A may help alleviate this to some extent, but can still be inefficient for many large-scaled problems. Therefore, the true value of this model is its role as a normative benchmark. It establishes the theoretical "perfect score" for correlational search, against which we will evaluate more computationally tractable heuristic methods in the following chapters.

CHAPTER 5 APPROXIMATE METHODS

The preceding chapter established a rigorous framework for determining the optimal search policy in a two-object environment using DP. While this approach provides a normative benchmark for the best possible performance, its practical application is severely limited by the curse of dimensionality. As the number of search cells n or the time horizon T increases, the state space grows exponentially, rendering the computation of the optimal policy intractable for most practical problems.

This computational barrier necessitates a shift from exact, optimal methods to efficient, approximate methods. This chapter explores two heuristic strategies designed to overcome the scalability challenge. These heuristics sacrifice the guarantee of optimality in favor of computational feasibility, aiming to produce high-quality, effective search policies for large-scale problems with significantly reduced computation time.

To assess the efficacy of these approaches, their performance will be quantitatively benchmarked against the true optimal policy generated by the DP solution. This comparison will allow us to measure the trade-off each heuristic makes between computational cost and mission success.

5.1 Limited Lookahead & Rollout

To find a computationally tractable policy for this search problem, we employ a $k-$step lookahead heuristic we discussed in Section 2.6.3. This is an online planning method that determines the best action at the current time t by solving a smaller, $k-$step DP problem. The value of states at the end of this short lookahead horizon is approximated using a rollout algorithm, which in turn uses a simple base policy to run fast simulations. After that we will compare this heuristic with the greedy policy with a simulated large-scale problem, measuring the computational cost involved.

5.1.1 Problem Formulation

The core of the method is the $k-step$ lookahead value function, $\tilde{J}$, which is defined by the following Bellman equations for a $k-step$ horizon:

$$\begin{cases} \tilde{J}_{t+k}(z,\theta) & = \hat{J}_{t+k}(z,\theta) \\ \tilde{J}_{t+l}(z,\theta) & = \max\limits_{a\in[n]} \left[R((z,\theta),a) + \tilde{J}_{t+l+1}(f((z,\theta),a))\right], \quad l = k-1,\cdots,0 \end{cases} \tag{5.1}$$

where the terminal value, $\tilde{J}_{t+k}(z,\theta)$ is approximated by $\hat{J}_{t+k}(z,\theta))$, which is the expected future

reward obtained by following a **base policy** from stage k to the final horizon T. This is estimated by averaging the outcomes of many Monte Carlo simulations. For these simulations, we use a Greedy policy, $\mathcal{G}$, as the base policy. This policy simply selects the action that maximizes the immediate expected reward (i.e., the immediate probability of detection). Formally, the Greedy policy is defined as:

$$\mathcal{G}(z,\theta) = \arg\max_{a\in[n]} \mathbb{E}[R((z,\theta),a)] \tag{5.2}$$

Specifically, if both objects are still hidden (i.e., $\theta = 0$), the policy finds the cell that maximizes (4.23), and if the related object has been found ($\theta \neq 0$), it finds the cell that maximizes the (4.25). Formally, we have:

$$\mathcal{G}(z,\theta) = \begin{cases} \arg\max_{a\in[n]} \left\{(1-\gamma_{1,a})\sum_{j=1}^{n} p_{aj}(z) - c_a\right\}, & \text{if } \theta = 0 \\ \arg\max_{a\in[n]} \left\{(1-\gamma_{1,a}) p_{a|\theta}(z) - c_a\right\}, & \text{if } \theta \neq 0 \end{cases} \tag{5.3}$$

At any decision stage t, until the target is found, the agent operates by continuously re-planning based on its most current information. As long as the remaining time exceeds the lookahead horizon (i.e., $T - t > k$), the agent follows this procedure: The agent treats its current belief (i.e., the posterior at stage t) as the initial prior p_0' for the new subproblem. It solves a k-step lookahead problem of horizon $T-t$ (with **Algorithm 5**), using p_0' and the current status of the related object $\theta$to find the best immediate cell to search a. As the current belief p_0' already accounts for the search history, this subproblem starts with a "fresh" search history $z' = [0, \ldots, 0]$. If the search ends without success, the agent updates θ if necessary, and computes a new belief given the search vector δ_a and θ. Then, it repeats the same procedure at $t+1$ until the detection occurs or $T-t = k$.

During the k-step recursion in (5.1) the agent uses the Greedy rollout approximation (see **Algorithm 4**) to estimate the values of states at the end of the short horizon k. Once $T - t = k$, the lookahead process ends, and the search continues with the complete DP of horizon k (using the current p_0' and θ).

5.1.2 Empirical Comparison between the Greedy Policy and 1-Step Lookahead

Before evaluating deeper lookahead horizons, it is essential to establish a performance baseline by comparing the greedy policy ($k = 0$) with the 1-Step lookahead ($k = 1$), the most fundamental non-myopic extension. This comparison serves to validate the rollout algorithm; theoretically, a 1-step planner using a greedy policy as its base policy for rollouts should perform at least as good as the standalone greedy policy. Following the problem defined in Section 4.3.1, we performed experiments using the same configuration of 100 seeds and 100 episodes per seed. For the 1-Step Lookahead, each rollout output was obtained using 500 Monte Carlo simulations to approximate

Algorithm 4 Rollout

Input:

- T_{rol}: Rollout horizon
- n: Number of cells
- $z_0, p_0, \gamma_1, \gamma_2$
- θ_{rol}: initial related object position
- num_sim: number of rollout simulations
- c : cost vector

Initialization:

- $rewards = []$: empty list that will contain rewards of all simulated search

Simulations:

1. For $sim = 1$ to num_sim:
   - 1.1 $z \leftarrow z_0$
   - 1.2 $\theta \leftarrow \theta_{rol}$
   - 1.3 Sample true object locations based on p_0 and θ_{rol}
   - 1.4 $rew \leftarrow 0$
   - 1.5 **For** $t = 1$ to T_{rol} **do**:
     - 1.5.1 **If** $\theta = 0$ **do**:
       - Select action $a \leftarrow \mathcal{G}(z, \theta)$ using (4.23) given (z, θ)
       - $rew \leftarrow rew - c[a]$
       - If O_1 is found:
         - $rew \leftarrow rew + 1$
         - Break inner for-loop.
       - check for O_2
       - if O_2 is found :
         - $\theta \leftarrow a$
         - $p_0 \leftarrow p_{0,.|a}$
       - $z \leftarrow z + \delta a$
     - 1.5.2 **Else if** $\theta \neq 0$ **do**:
       - Select action $a \leftarrow \mathcal{G}(z, \theta)$ using (4.25).
       - $rew \leftarrow rew - c[a]$
       - If O_1 is found:
         - $rew \leftarrow rew + 1$
         - Break inner for-loop.
       - $z \leftarrow z + \delta_a$
   - 1.6 $rewards \leftarrow rewards + [rew]$
2. $avg \leftarrow sum(rewards)/len(rewards)$

Output:

- avg: The average rewards for num_sim search simulations.

the expected reward of terminal nodes of the lookahead horizon. However, **we consider only the scenario with mixed objective**, the cost $c \neq \vec{0}$, and the searcher gets a reward of 1 if the target is found. The empirical results are detailed in table 5.1.

Metric	**Greedy Policy**	**1-Step Lookahead**
Avg. Episode Reward	-0.5203 ± 0.0965	-0.5071 ± 0.0895
Avg. Success Rate	0.4885 ± 0.0502	0.6116 ± 0.0514
Avg. Episode Length	7.19 ± 0.3450	6.89 ± 0.3840

Table 5.1 Comparison between the Greedy baseline and the 1-Step Lookahead policy in a two-object search scenario (100 seeds, 100 episodes each, and 500 simulations per rollout). Results are reported as Mean ± Standard Deviation.

Table 5.1 shows that 1-Step lookahead policy outperformed the greedy baseline in average reward (-0.5071 vs -0.5203) with just 500 Monte Carlo simulations per rollout. This outcome is also supported by their respective average success rates. The lookahead policy succeeded 61.16% of the time, while the greedy policy only succeeded only 48.85% of the time. This shows that looking just one step ahead helps the searcher avoid failing more often than the greedy policy. The lookahead policy also finishes episodes faster on average (6.89 steps) than the greedy policy (7.19 steps). This gives a ground proof that for this specific problem the k-step lookahead is a better alternative to DP than the greedy policy.

5.1.3 Empirical Comparison between Different k: 1-Step vs 2-Step

The 1-step lookahead showed us a good benchmark, defeating the greedy policy. Now we aim to see how looking with deeper k might improve the search efficiency. We still considered the experiment from Section 5.1.2 on the problem defined in Section 4.3.1, except that this time we used 100 simulations per rollout. We compared 1-step lookahead against 2-step lookahead, linking them to the optimal two-object DP policy as the benchmark. A summary of the results is provided in Table 5.2.

Policy	**Success Rate ↑**	**Ep. Reward ↑**	**Ep. Length ↓**
1-Step Lookahead	0.599 ± 0.060	-0.568 ± 0.100	7.22 ± 0.4899
2-Step Lookahead	0.629 ± 0.052	-0.496 ± 0.094	6.66 ± 0.3600
Two-Object DP	0.626 ± 0.044	$\mathbf{-0.454 \pm 0.097}$	6.40 ± 0.3380

Note: Arrows indicate whether higher (↑) or lower (↓) values are better.

Table 5.2 Comparison of $k-$step Lookahead policies for $k \in \{1, 2\}$ with DP in a two-object search scenario (100 seeds, 100 episodes each, and 100 simulations per rollout)

The results in Table 5.2 demonstrate that the two-object DP policy remains the superior strategy for maximizing the problem's objective. The DP's Average Episode Reward (-0.454) remains the theoretical benchmark for performance, and it achieves the best search efficiency with the shortest Average Episode Length (6.40 steps). While the success rate of the 2-step lookahead appears numerically higher (0.629) than the DP's (0.626), the difference is not statistically significant as their standard deviations overlap.

The data confirms the expectation that a deeper lookahead horizon yields a more effective policy. The **2-step lookahead** (-0.496 reward) clearly outperforms the **1-step lookahead** (-0.568 reward). By looking deeper in k, the searcher can better leverage the information provided by finding the related object mid-search, leading to more sophisticated search sequences and shorter mission durations. This superiority to 1-step lookahead extends to other metrics as well.

5.2 Deep Q-Network

The previous methods we analyzed—both the optimal DP solver and the $k-$step lookahead heuristic—are model-based. They rely on having a model of the search's evolution (i.e., they can compute state transitions and probabilities) and use it to plan the best cell to search at each stage. The DP solver plans for all possible futures, making it optimal but intractable for a large horizon. The $k-$step lookahead plans for a limited future at every single step, which can become computationally crippling for real-time decisions as k increases.

This leads us to a fundamentally different, model-free approach: RL, which we introduced in Section 2.6.4. Instead of performing complex planning at decision time, an RL agent, such as a DQN [37], moves all the computational effort offline. By learning from many simulated search trajectories in a training phase, the agent learns to approximate the optimal action-value function, $Q_t^*(s, a)$. The drawback with this approach is the training phase, since the agent must experience as many trajectories as possible to accurately approximate $Q_t^*(s, a)$. To implement this, we start by formalizing the RL formulation of our search problem.

5.2.1 Problem Formulation

In our previous DP formulation for the two-object search, the non-terminal state at time t was represented as the tuple (z, θ). However, because the optimal policy in a finite horizon search problem is inherently time-dependent, the action-value function must reflect the remaining search horizon. Therefore, we redefine the the non-terminal state for the RL agent as the tuple (z, θ, t), where t is the current time step. For our Deep RL approach, we utilize a **Deep Q-Network (DQN)** agent, which has proven effective in similar discrete search scenarios [17, 36].

A neural network cannot take directly the tuple (z, θ, t) as an input. We need to encode this information into a single vector. Therefore, the observation given to the agent at each step is a vector of size $n + 2$, created by concatenating z, θ, and t. We can see a basic architecture of the DQN in Figure 5.1.

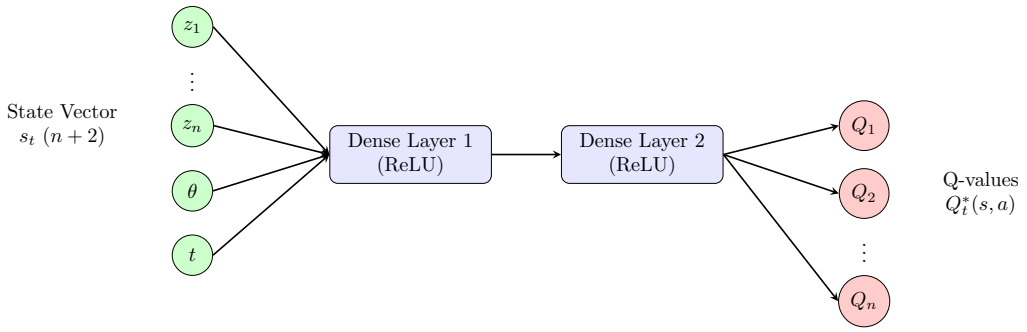

Figure 5.1 Schema of the DQN policy. The input state (a vector of size $n+2$ containing z, θ, and t) is passed through two dense hidden layers with ReLU activations. The output layer produces n Q-values, one for each cell to search.

The choice of the neural network architecture is motivated by the nature of this state representation. Unlike image-based search problems that benefit from Convolutional Neural Networks (CNNs), our input is a structured, low-dimensional feature vector rather than a matrix. Therefore, a **Multi-Layer Perceptron (MLP)** is an appropriate and efficient function approximator for this task. As shown in Figure 5.1, the network consists of two dense hidden layers with ReLU activations, providing the necessary non-linear capacity to approximate Q_t^* while remaining computationally tractable.

The action space remains the discrete set $[n]$, representing the choice of which cell to search. The state transition dynamics are governed by the environment, which must internally track the search history vector z, the status of the related object θ, and the current time t. After an action a is taken, the environment increments z, updates θ if applicable, and constructs the next state $(z+\delta_a, \theta', t+1)$ for the next iteration. The prior p_0 is used internally in the environment to sample objects' locations at the beginning of each search episode during training.

Because the agent does not have an explicit model of the environment, it does not directly perform the Bayesian update of the belief. Instead, it must learn the optimal search strategy directly from the raw inputs (z, θ, t) by sampling trajectories during training.

It is important to note that we are training this agent for a specific scenario with a fixed prior, p_0 Since p_0 is constant across all episodes, the agent does not need it as an input; it can implicitly "memorize" the characteristics of this specific p_0 within its network weights. If we were training a

general-purpose agent to handle any arbitrary prior of size $n \times n$, p_0 would need to be included as part of the input state.

5.2.2 Objective: The Optimal Action-Value Function

The goal of a DQN is to learn the optimal time-dependent action-value function, denoted as $Q_t^*(s, a)$. This function represents the maximum possible total future reward the agent can expect to receive if it is in a state $s = (z, \theta)$ at time t, takes action $a \in [n]$ (searches a specific cell), and then continues to act optimally for the remainder of the search horizon.

This ideal function is defined by the Bellman Equation for finite-horizon problems. It states that the value of the current action at time t is the immediate reward received plus the discounted value of the best possible action in the resulting state at time $t + 1$. Formally, we have:

$$Q_t^*(s, a) = \mathop{\mathbb{E}}_{s}\left[R(s, a) + \beta \max_{a' \in [n]} Q_{t+1}^*(s', a')\right], \tag{5.4}$$

where $R(s, a)$ is the expected immediate reward received as defined in Section 3.4. β represents the discount factor ($\beta = 1$ in our case because we are in a finite horizon problem). The term s' represents the next state, and $\max_{a'} Q_{t+1}^*(s', a')$ is the value of the best action at the next time step. For the final stage T, the value is defined as $Q_T^*(s, a) = 0$.

In practice, Q_t^* is unknown, so we aim to approximate it with a neural network $Q_t(s, a; \omega_t)$, parametrized by weights ω_t. The DQN algorithm learns to approximate Q_t^* by minimizing a loss function based on the Bellman equation. To improve stability, the algorithm utilizes a target network $Q_t(s, a; \omega_t^{target})$, which is updated periodically.

During training, the agent stores its experiences (s, a, R, s') in a **replay buffer** ($\mathcal{D}$), a memory module that stores a large history of past transitions, allowing the agent to sample random batches of experiences to break the correlation between consecutive experienced search steps. At each training step, the agent samples a batch from $\mathcal{D}$ and calculates the target value (y_t^{target} defined below) using the target network weights ω_t^{target}:

$$y_t^{target} = R(s, a) + \beta \max_{a' \in [n]} Q_{t+1}(s', a'; \omega_{t+1}^{target}). \tag{5.5}$$

The objective of the DQN is to minimize the Mean Squared Error (MSE) between y_t^{target} and the

current prediction:

$$L(\omega_t) = \mathop{\mathbb{E}}_{(s,a,R,s',t)\sim\mathcal{D}} \left[\left(\underbrace{y_t^{target}}_{\text{Target}} - \underbrace{Q_t(s,a;\omega_t)}_{\text{Prediction}} \right)^2 \right]. \tag{5.6}$$

The weights ω_t are adjusted to minimize $L(\omega_t)$, forcing the predicted Q-values to converge toward the stable target values using optimization algorithms like Adam or Stochastic Gradient Descent (SGD).

5.2.3 Basic Simulation

Similarly to previous methods, we trained a DQN agent in solving the problem defined in Section 4.3.1. We use the DQN implementation from **Stable Baselines3** [**45**]. We trained the model over 100 seeds, and for each seed the training was set for a total of $100,000$ timesteps, with a training batch of 64. After each $10,000$ training steps, the model was evaluated on 100 search scenarios, and the model with the highest average rewards was saved as the **best model**. At the end of the training, the performance scores of best models of all seeds were averaged and results are summarized in Table 5.3. Figure 5.2 shows the average training reward progress across all seeds, and Figure 5.3 shows the average evaluation rewards.

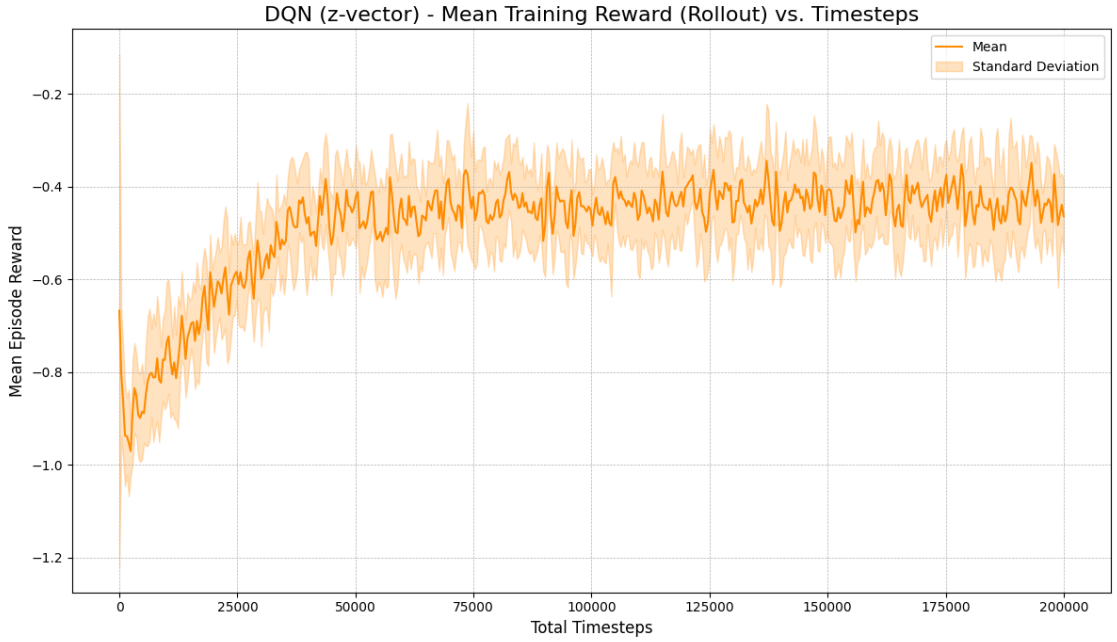

Figure 5.2 DQN - Mean Training Reward vs. Timesteps. The solid line shows the mean training reward per training batch, averaged across 100 seeds of 100 episodes each. The shaded area represents the standard deviation.

Results in Table 5.3 show that the DQN agent achieved a higher average reward (-0.430) compared

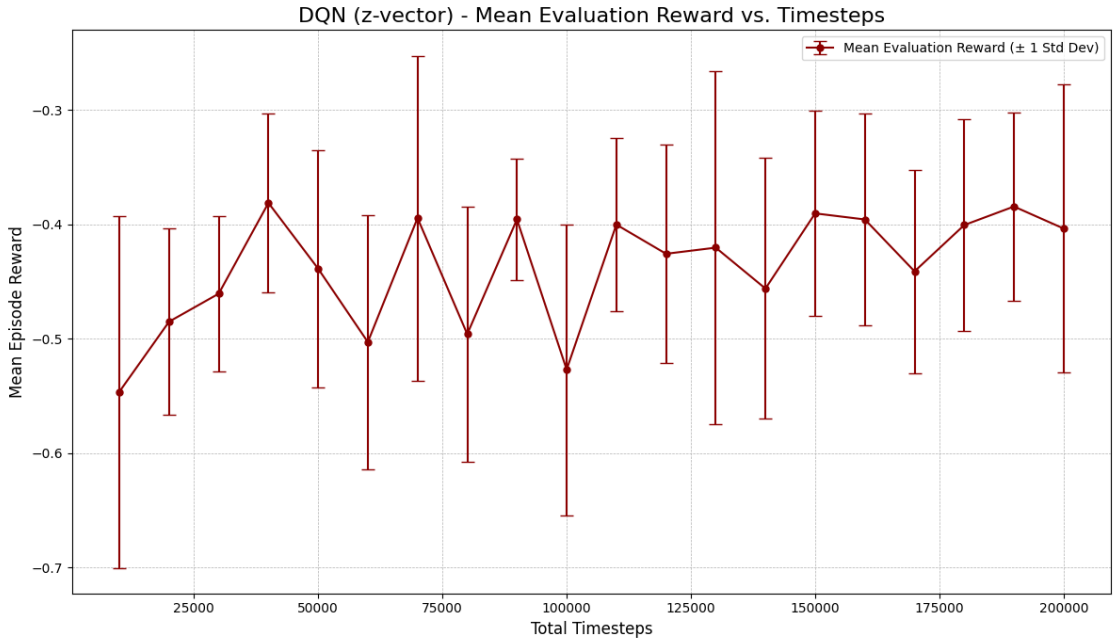

Figure 5.3 DQN - Mean Evaluation Reward vs. Timesteps. The dot shows the mean evaluation reward per evaluation timestep averaged across 100 seeds of 100 episodes each. The vertical line represents the standard deviation.

to the optimal DP baseline (-0.454). However, we must interpret this result with caution. The DQN displays a significantly higher standard deviation (± 0.116) compared to the two-object policy (± 0.0973). This indicates instability in the learning process. It suggests the DQN agent has not fully converged to the consistent, optimal path, resulting in a policy that is less reliable despite the higher average reward.

We also point out that the performance of DQN is influenced by our choice of hyperparameters. Maybe a different training setting would have yielded a more stable policy. However, we are satisfied to see a performance that remains competitive with model-based methods. This shows that Deep RL is a promising alternative to solve such correlated search problems.

5.3 Simulations of a Large Problem

In this section, we assess the effectiveness of the heuristic methods discussed in this chapter. This includes the $0-$step (Greedy) policy, the $k-$step lookahead policies, and our trained DQN agent. In the the previous section, we used a small $n = 5$ problem, which was useful for comparing our heuristics against the "perfect" score from the optimal DP solver. However, it is useful to test these heuristics to see how they perform on a problem that is too large for the optimal DP solver to even run.

Policy	Ep. Reward ↑	Success Rate ↑	Ep. Length ↓
DQN	-0.430 ± 0.1160	0.586 ± 0.057	6.97 ± 0.45
Two-Object DP	-0.454 ± 0.0973	0.626 ± 0.044	6.40 ± 3.38

Note: Arrows indicate whether higher (↑) or lower (↓) values are better.

Table 5.3 Comparison of DQN policies with DP in Two-Object Scenario (100 seeds, 100 episodes each). Results are reported as Mean ± Standard Deviation. The best performance for each metric is marked in bold.

5.3.1 Large Problem Setup

We define an intractable environment. To create this test, we'll generate a new, much larger problem with the following parameters: $\mathbf{n = 25, T = 20}$, this means p_0 would be a $\mathbf{25 \times 25}$ matrix. Such problem would be intractable for DP because only for the last stage, there will be $\binom{\mathbf{20+25-1}}{\mathbf{25-1}}$ states for each possible value of $\theta \in \{\mathbf{0}\} \cup [\mathbf{25}]$, which is astronomically large. We use code to randomly generate the problem parameters with a specific seed to ensure reproducibility. p_0 matrix will be generated with a strong correlation. The costs c_i and miss-detection rates γ_1, γ_2 will be randomly generated vectors of size 100.

5.3.2 Experimental Comparison

In this section we compare the 1−step Lookahead, 2−step Lookahead, and a Deep Q-Network (DQN) agent. The primary objective is to demonstrate that lookahead methods can still function as viable planning approximations in large environments where DP fails.

Due to the significant computational cost of online planning in this large state space, the experimental parameters were constrained to 10 seeds with 10 episodes each. Furthermore, the lookahead agents were limited to only 10 simulations per rollout, and the DQN agent utilized a basic training setup. The results are summarized in Table 5.4.

Policy	Success Rate ↑	Ep. Reward ↑	Ep. Length ↓
1-Step Lookahead	0.220 ± 0.087	-2.738 ± 0.245	18.11 ± 1.08
2-Step Lookahead	0.250 ± 0.163	-2.432 ± 0.372	17.94 ± 1.57
DQN	0.154 ± 0.053	-1.926 ± 0.127	18.02 ± 0.67

Note: Arrows indicate whether higher (↑) or lower (↓) values are better.

Table 5.4 Comparison of heuristics in a large problem setup (10 seeds, 10 episodes each, 10 simulations per rollout for lookahead).

The results in Table 5.4 indicate that the trend observed in the smaller experiments holds true: the 2-Step still outperformed the 1-step lookahead. This is expected since they used the same

experiment configuration, except for k.

While the DQN agent reports a numerically higher average reward, this should not be interpreted as superior performance in this context. These performance levels across all agents are attributable to the restricted experimental setup: the lookahead methods require significantly more rollout simulations to accurately approximate the value function, and the DQN agent requires a more extensive training regimen to converge in such a high-dimensional state space. Consequently, this comparison serves primarily to validate the feasibility of these methods in large-scale scenarios rather than to establish their maximum potential performance.

5.4 Summary

This chapter explored two effective heuristic methods: k-step lookahead and DQNs to address the computational limits of the DP solution. Both methods show promising pathways to finding admissible solutions for large-scale search problems, as they have higher success rates on average when compared to the baseline greedy policy.

The k-step lookahead approach framed the problem as a form of online planning. It solves a small, exact DP problem at each decision step and uses the known model of the environment to make smart, future-oriented choices. Its main strength is this explicit use of the model, which allows for complex reasoning and a flexible balance between planning depth (k) and decision-time cost. However, its biggest downside is that decision-time cost; needing to perform a recursive calculation at each step can make it unsuitable for tasks that require instantaneous decisions. Moreover, its long-term performance depends on the quality of the simpler rollout policy used to estimate future values.

On the other hand, the DQN algorithm tackled the problem with a model-free learning approach. By training a neural network policy through extensive offline simulation, it learns a direct link from the combined state vector (which includes the search history vector and related object status) to the expected value (Q-value) of each action. Its main advantages are its ability to handle large state spaces and its very fast execution speed once trained, making it well-suited for real-time use. The main trade-offs include the high upfront computation costs needed for training and the fact that the resulting policy is an approximation without a guarantee of being the best. The effectiveness of the final policy is also very sensitive to the chosen network architecture and the tuning of hyperparameters.

In the end, both the online planning of the k-step lookahead and the offline learning of DQN offer practical methods for balancing optimality and feasibility. We do not claim that deeper lookahead will always outperform DQN in every situation, even though it has demonstrated better search efficiency in this particular case.

Algorithm 5 k-step Lookahead Policy Computation for Two-Object Search

Input:

- T_{rem}: Remaining mission horizon (i.e., $T - t$)
- Static parameters: $n, p_0, \gamma_1, \gamma_2$
- θ_{init}: Current related object status
- k: Lookahead horizon
- num_sim: number of rollout simulations

Data Structures & Initialization:

- Define the action set $\mathcal{A} = \{1, 2, \ldots, n\}$.
- Initialize two global dictionaries: $J_{values} = \{\}$ and $Policy = \{\}$.
- $z_0 = [0, \ldots, 0]$: initial search vector

Base Case:

- State space for stage k if $\theta_{init} = 0$: $S_k = \{(z, \theta) \mid z \in \mathbb{Z}^n_{\geq 0}, \sum z_i = k, \theta \in \{0, \ldots, n\}\}$.
- State space for stage k, if $\theta_{init} \neq 0$: $S_k = \{(z, \theta_{init}) \mid z \in \mathbb{Z}^n_{\geq 0}, \sum z_i = k\}$.
- Create a dictionary for the final stage value function: $J_k = \{\}$.
- **For each** state $(z, \theta) \in S_k$ **do**:
  - $J_k[(z, \theta)] \leftarrow \text{Rollout}(z, T_{rem} - k, \theta, p_0, \gamma_1, \gamma_2, c, num\_sim)$ ▷ Apply **Algorithm 4** for $T_{look} - k$ steps
- $J_{values}[k] = J_k$.

Backward Recursion:

1. **For** $t = k - 1$ down to 0 **do**:
   - 1.1 Initialize empty dictionaries for the current stage: $J_t = \{\}$, $\mu_t = \{\}$.
   - 1.2 Define state space for stage, $S_t = \{(z, \theta) \mid z \in \mathbb{Z}^n_{\geq 0}, \sum z_i = t, \theta \in \{0, \ldots, n\}\}$ if $\theta_{init} = 0$, otherwise $S_t = \{(z, \theta_{init}) \mid z \in \mathbb{Z}^n_{\geq 0}, \sum z_i = t\}$.
   - 1.3 **For each** state $(z, \theta) \in S_t$ **do**:
     - **if** $\theta == 0$: ▷ Case 1: Both objects are hidden.
       - Compute the joint posterior matrix $p(z)$ using p_0 and z per (4.7).
       - Find the best action and its value:

$$J_t[(z, 0)] = \max_{a \in \mathcal{A}} \{\ldots\} \quad \triangleright \text{ Using (4.34)}$$
$$\mu_t[(z, 0)] = \arg\max_{a \in \mathcal{A}} \{\ldots\}$$

     - **else**: ▷ Case 2: O_2 has been found in cell θ.
       - Compute the conditional posterior vector $p_{\cdot|\theta}(z)$ based on history z and the prior $p_{0,\cdot|\theta}$.
       - Find the best action and its value:

$$J_t[(z, \theta)] = \max_{a \in [n]} \{\cdots\} \quad \triangleright \text{ Using (4.36)}$$
$$\mu_t[(z, \theta)] = \arg\max_{a \in \mathcal{A}} \{\cdots\}$$

     - Store the results: $J_{values}[t] = J_t$, $Policy[t] = \mu_t$.

Output:

- $Policy[0][([0, \ldots, 0], \theta_{init})]$: The optimal action for the initial state.

CHAPTER 6 CONCLUSION

This chapter brings our study to a close. It provides a final summary of the two-object search problem we investigated, highlighting our main contributions and acknowledging the limitations of our approach. We'll also look ahead to some future research directions where our work can serve as a benchmark for tackling even more realistic and advanced problems.

6.1 Summary and Contribution

This thesis has addressed the challenge of optimal search problem in a discrete environment where the primary target's location is correlated with a secondary object. We have moved beyond the classic single-object search paradigm to develop a more powerful framework that leverages this statistical relationship to improve search efficiency.

The primary contribution of this research is a general model for two-object search. By representing the object correlation with a full joint probability matrix, our approach can capture any form of statistical dependency without imposing the restrictive spatial constraints—such as adjacency—common in prior work. This allows our framework to capture any possible statistical dependency, from targets being co-located to being on opposite sides of the search environment, making it highly flexible for scenarios where a specific dependency can be estimated. Based on this model, we formulated a DP solution that computes a provably optimal policy. This policy is inherently complete; it optimally handles the fundamental change in the problem that occurs if the related object is found mid-search, elegantly transitioning to a new, more informed single-object problem using the same policy. A key innovation that enables this is the formulation of a new state representation, which consists of a pair of a vector that tracks the search count for each cell, disregarding the search order, and a variable that tracks the status of the related object. This framework provides a principled method for quantifying the value of information provided by the related object and formally establishes the conditions under which this correlational search is strategically advantageous.

To validate the theoretical model, we developed and evaluated two scalable, approximate methods for large-scale problems: an online planning heuristic based on k-step lookahead, and an offline learning approach using the RL algorithm, DQN. Our experiments confirmed that both heuristics can learn high-quality policies compared to the greedy policy, which served as the baseline heuristic, and that the performance of the $k-$step lookahead improves as its planning horizon gets deeper, approaching the optimal benchmark; however, with a high computational cost. The result of the DQN agent demonstrated that a NN can effectively learn a complex search strategy directly from

observing just the search history vector and the related object status, but also with a cost of high computation time for the training phase. This validates the use of Deep RL as a viable alternative for this class of problem.

6.2 Limitations

While our models provide a robust foundation for correlational search, their applicability is bounded by several key assumptions. The solutions presented require a known prior distribution and consider only static targets within a fixed time horizon. Crucially, we assume a constant sensor model with no false alarms, which ensures that any detection is true and definitive.

The most significant limitation, however, is scalability. The optimal DP solution, while serving as an invaluable theoretical benchmark, suffers from the curse of dimensionality. While we discussed how state encoding can improve memory efficiency, the computational burden of evaluating an exponentially large number of states remains the primary bottleneck. Even the heuristic methods, while more scalable, have their own computational trade-offs, from the online planning cost of the k-step lookahead to the extensive offline training time required for the DQN agent.

6.3 Future Research

The limitations of this work highlight several promising avenues for future research. An immediate extension would be to relax the static target assumption and develop models for searching for mobile or evasive targets, which would require a more complex state representation that includes target motion models. Another valuable direction would be to address the problem of unknown priors, where the agent must simultaneously learn the correlation model while planning its search strategy. Finally, expanding the framework to handle multi-agent search, where a team of coordinated agents collaborates to find the correlated objects, would be a significant step towards solving large-scale problem, real-world SAR operations.

REFERENCES

[1] S. Golchi, "Informative priors and bayesian computation," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 782–789. [Online]. Available: https://doi.org/10.1109/DSAA.2016.67

[2] L. D. Stone, C. M. Keller, T. M. Kratzke, and J. P. Strumpfer, "Search for the wreckage of air france flight af 447," *Statistical Science*, vol. 29, no. 1, pp. 69–80, 2014. [Online]. Available: http://www.jstor.org/stable/43288452

[3] D. Bertsekas, *Dynamic Programming and Optimal Control: Volume I*, ser. Athena Scientific optimization and computation series. Athena Scientific, 2012.

[4] P. N. Loxley and K.-W. Cheung, "A dynamic programming algorithm for finding an optimal sequence of informative measurements," *Entropy*, vol. 25, no. 2, 2023. [Online]. Available: https://www.mdpi.com/1099-4300/25/2/251

[5] D. Assaf and S. Zamir, "Optimal sequential search: A bayesian approach," *The Annals of Statistics*, vol. 13, no. 3, pp. 1213–1221, 1985. [Online]. Available: http://www.jstor.org/stable/2241134

[6] S. J. Benkoski, M. G. Monticino, and J. R. Weisinger, "A survey of the search theory literature," *Naval Research Logistics (NRL)*, vol. 38, no. 4, pp. 469–494, 1991.

[7] B. O. Koopman, "Search and screening," *OEG Rep.*, 1946.

[8] L. D. Stone, *Theory of Optimal Search*, ser. Mathematics in Science and Engineering. Elsevier Science, 1976.

[9] H. R. Richardson and J. H. Discenza, "The united states coast guard computer-assisted search planning system (casp)," *Naval Research Logistics Quarterly*, vol. 27, no. 4, pp. 659–680, 1980. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800270413

[10] B. O. Koopman, "Search and its optimization," *The American Mathematical Monthly*, vol. 86, no. 7, pp. 527–540, 1979. [Online]. Available: https://doi.org/10.1080/00029890.1979.11994850

[11] J. B. Kadane, "Optimal whereabouts search," *Oper. Res.*, vol. 19, no. 4, p. 894–904, Aug. 1971. [Online]. Available: https://doi.org/10.1287/opre.19.4.894

[12] L. D. Stone and J. B. Kadane, "Optimal whereabouts search for a moving target," *Operations Research*, vol. 29, no. 6, pp. 1154–1166, Dec. 1981. [Online]. Available: https://ideas.repec.org/a/inm/oropre/v29y1981i6p1154-1166.html

[13] A. Aydemir, K. Sjöö, and P. Jensfelt, "Object search on a mobile robot using relational spatial information," in *Intelligent Autonomous Systems 11*. IOS Press, 2010, pp. 111–120.

[14] A. Sarmiento, R. Murrieta, and S. Hutchinson, "An efficient strategy for rapidly finding an object in a polygonal world," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, 2003, pp. 1153–1158 vol.2.

[15] A. Aydemir, A. Pronobis, M. Göbelbecker, and P. Jensfelt, "Active visual object search in unknown environments using uncertain semantics," *IEEE Transactions on Robotics*, vol. 29, no. 4, pp. 986–1002, 2013.

[16] T. Kollar and N. Roy, "Utilizing object-object and object-scene context when planning to find things," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 2168–2173.

[17] B. Matzliach, I. Ben-Gal, and t. Kagan, "Detection of hidden moving targets by a group of mobile agents with deep Q-learning," *Robotics*, vol. 12, no. 4, 2023. [Online]. Available: https://www.mdpi.com/2218-6581/12/4/103

[18] D. Assaf and S. Zamir, "Continuous and discrete search for one of many objects," *Oper. Res. Lett.*, vol. 6, no. 5, p. 205–209, Nov. 1987. [Online]. Available: https://doi.org/10.1016/0167-6377(87)90050-2

[19] D. Assaf and A. Sharlin-Bilitzky, "Dynamic search for a moving target," *Journal of applied probability*, vol. 31, no. 2, pp. 438–457, 1994.

[20] A. Sharlin, "Optimal search for one of many objects hidden in two boxes," *European Journal of Operational Research*, vol. 32, no. 2, pp. 251–259, 1987, third EURO Summer Institute Special Issue Decision Making in an Uncertain World. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221787801479

[21] N. O. Song and D. Teneketzis, "Discrete search with multiple sensors," *Mathematical Methods of Operations Research*, vol. 60, no. 1, pp. 1–13, 2004. [Online]. Available: https://doi.org/10.1007/s001860400360

[22] L. D. Stone, R. L. Streit, T. L. Corwin, and K. L. Bell, *Bayesian Multiple Target Tracking, Second Edition*, ser. Radar/Remote Sensing. Artech House, 2013.

[23] S. Ivić, B. Crnković, H. Arbabi, S. Loire, P. Clary, and I. Mezić, "Search strategy in a complex and dynamic environment: The mh370 case," *Scientific Reports*, vol. 10, no. 1, p. 19640, 2020.

[24] M. Meghjani, S. Manjanna, and G. Dudek, "Multi-target search strategies," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2016, pp. 328–333.

[25] C. A. Riggs, K. Cornes, H. J. Godwin, S. P. Liversedge, R. Guest, and N. Donnelly, "The importance of search strategy for finding targets in open terrain," *Cognitive research: principles and implications*, vol. 2, pp. 1–17, 2017.

[26] J. M. Wolfe, G. A. Alvarez, R. Rosenholtz, Y. I. Kuzmova, and A. M. Sherman, "Visual search for arbitrary objects in real scenes," *Attention, Perception, & Psychophysics*, vol. 73, pp. 1650–1671, 2011.

[27] J. R. Frost, L. D. Stone, U. S. C. G. O. of Operations., U. C. G. R. . D. Center., I. Metron, and S. . Company, "Review of search theory: Advances and applications to search and rescue decision support," U.S. Coast Guard, Washington, DC, Tech. Rep. CG-D-15-01, 2001, prepared for U.S. Department of Transportation, U.S. Coast Guard (G-OPR). [Online]. Available: https://apps.dtic.mil/sti/tr/pdf/ADA397065.pdf

[28] J. Clarkson, K. Y. Lin, and K. D. Glazebrook, "A classical search game in discrete locations," *Math. Oper. Res.*, vol. 48, no. 2, p. 687–707, May 2023. [Online]. Available: https://doi.org/10.1287/moor.2022.1279

[29] R. M. Francos and A. M. Bruckstein, "Search for smart evaders with sweeping agents," *Robotica*, vol. 39, no. 12, pp. 2210–2245, 2021.

[30] T. H. Chung, G. A. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics: A survey," *Autonomous robots*, vol. 31, pp. 299–316, 2011.

[31] M. Sani, B. Robu, and A. Hably, "Pursuit-evasion game for nonholonomic mobile robots with obstacle avoidance using nmpc," in *2020 28th Mediterranean conference on control and automation (MED)*. IEEE, 2020, pp. 978–983.

[32] W. Jiang, J. Yang, H. Chen, C. Xiong, and Z. Zhao, "Research on submarine search strategy based on bayesian search theory," in *2024 3rd International Conference on Data Analytics, Computing and Artificial Intelligence (ICDACAI)*, 2024, pp. 1011–1017.

[33] R. Liu, Z. Chen, and P. Zhang, "Probabilistic method for optimizing submarine search and rescue strategy under environmental uncertainty," in *Advanced Intelligent Computing Technology and Applications*. Springer Nature Singapore, 2025, pp. 475–487.

[34] S. Davey, N. Gordon, I. Holland, M. Rutten, and J. Williams, *Bayesian Methods in the Search for MH370*. Springer Nature, 2016.

[35] N. Chakraborty, P. N. Kasthurirangan, J. S. Mitchell, L. Nguyen, and M. Perk, "Provable methods for searching with an imperfect sensor," *arXiv preprint arXiv:2410.06069*, 2024.

[36] B. Matzliach, I. Ben-Gal, and E. Kagan, "Detection of static and mobile targets by an autonomous agent with deep Q-learning abilities," *Entropy*, vol. 24, no. 8, p. 1168, 2022.

[37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[38] T. F. Lidbetter, "Search games with multiple hidden objects," *SIAM J. Control Optim.*, vol. 51, no. 4, p. 3056–3074, Jan. 2013. [Online]. Available: https://doi.org/10.1137/120893938

[39] M. A. A. El-Hadidy, "On maximum discounted effort reward search problem," *Asia-Pacific Journal of Operational Research*, vol. 33, no. 03, p. 1650019, 2016. [Online]. Available: https://doi.org/10.1142/S0217595916500196

[40] L. E. Wixson and D. H. Ballard, "Using intermediate objects to improve the efficiency of visual search," *International Journal of Computer Vision*, vol. 12, no. 2, pp. 209–230, 1994.

[41] P. Loncomilla, J. Ruiz-del Solar, and M. Saavedra A, "A bayesian based methodology for indirect object search," *Journal of Intelligent & Robotic Systems*, vol. 90, no. 1, pp. 45–63, 2018.

[42] K. Yadav, J. Krantz, R. Ramrakhya, S. K. Ramakrishnan, J. Yang, A. Wang, J. Turner, A. Gokaslan, V.-P. Berges, R. Mootaghi, O. Maksymets, A. X. Chang, M. Savva, A. Clegg, D. S. Chaplot, and D. Batra, "Habitat challenge 2023," https://aihabitat.org/challenge/2023/, 2023.

[43] K. Zheng, R. Chitnis, Y. Sung, G. Konidaris, and S. Tellex, "Towards optimal correlational object search," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE Press, 2022, p. 7313–7319. [Online]. Available: https://doi.org/10.1109/ICRA46639.2022.9812252

[44] T. M. Cover, *Elements of Information Theory*, ser. Wiley series in telecommunications and signal processing. Wiley-India, 1999.

[45] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[46] A. B. Siddique, S. Farid, and M. Tahir, "Proof of bijection for combinatorial number system," *arXiv preprint arXiv:1601.05794*, 2016.

APPENDIX A EFFICIENT STATE MEMORIZATION

We have seen previously that using the search vector z_t (the number of searches per cell at time t) allows us to disregard the specific order of past actions to obtain the current belief p_t. However, this strategy can still be impractical in terms of memory efficiency, for problems with a large number of search cells n, or a long search horizon T. In this section we provide an approach that provides a better memory efficiency for both Algorithm 1 and Algorithm 3.

In fact, in our settings we need to store in memory all possible z (where $\sum_i z_i = t$) at each stage t for referral during the backward recursion. The number of unique z vectors can be found using the **stars and bars** combinatorial principles. This corresponds to the number of ways to distribute t searches (i.e., t stars) into n distinct bins (i.e., $n-1$ bars). This number is given by $\binom{t+n-1}{n-1}$. Thus, the total number of vectors that must be stored for the whole horizon T, that we denote by η, is the sum over all possible states to store at each stage $t \in [T]$:

$$\eta = \sum_{t=1}^{T} \binom{t+n-1}{n-1}. \tag{A.1}$$

For large T and n, storing η vectors of size n can be computational expensive in memory consumption. For instance, if $n = 10$ cells and $T = 20$ steps, the number of states at $t = 20$ alone is

$$\binom{20+10-1}{10-1} \approx 20 \text{ million.}$$

Storing each of these as a 10-element integer vector (i.e., 8 bytes per integer) would require roughly $20,000,000 \times 10 \times 8 = 1600$ Megabytes just for the final time step. So, summing over all $t \in [T]$ makes the total memory requirement substantial.

To address this, we need a more memory-efficient way to store states without altering the backward recursion algorithm. Using the principle of the combinatorial number system [46], we can define a family of functions $\{encode_t\}_{t\in\{0,\cdots,T\}}$ that map each unique vector to a unique index. At each stage t, $encode_t$ creates a bijection between the set of all possible vectors z and set of integers $\left[\binom{t+n-1}{n-1}\right]$. The function *encode* can be constructed using an algorithm based on the combinatorial number system. By ordering all possible z_t vectors lexicographically, we can assign a unique integer index e_t to each one. For instance, if $n = 3$ and $T = 3$, we can have the following encoding:

$$encode_3([0,0,3]) \mapsto 1$$
$$encode_3([0,1,2]) \mapsto 2$$
$$encode_3([0,2,1]) \mapsto 3$$
$$encode_3([0,3,0]) \mapsto 4$$
$$encode_3([1,0,2]) \mapsto 5$$
$$encode_3([1,1,1]) \mapsto 6$$
$$encode_3([1,2,0]) \mapsto 7$$
$$encode_3([2,0,1]) \mapsto 8$$
$$encode_3([2,1,0]) \mapsto 9$$
$$encode_3([3,0,0]) \mapsto 10.$$

Therefore, instead of storing in memory all vectors of size n, we can store their encoded indices. For instance, if $n = 10$ cells and $T = 20$ steps, instead of using 1600 megabits of memory just for this stage, this encoding will require only 160 megabits, since we will use only integers in place of 10-sized vectors.

We also define the function $decode_t = (encode_t)^{-1}$, the inverse of $encode_t$ which takes a given integer index back to its vector form (i.e., $decode_t(e_t) \mapsto z_t$). This is an essential step to allow the algorithm to correctly identify the vector associated with a specific index stored in memory.

Note: For the case of two-object search, the total number of possible states is $(\mathbf{n} + \mathbf{1}) \cdot \eta$ because $\theta \in \{0\} \cup [n]$.

APPENDIX B LINKS TO SIMULATION CODES

In this section we provide links to source codes of problems simulated problem solved in our this work.

B.1 Code for Chapter 3

This section contains links to experimental codes of the problem defined in Section 3.7.

Here is the code for the single-object DP solver experiment

B.2 Code for Chapter 4

This section contains links to experimental codes of the problem defined in Section 4.3.1.

1. Here is the code for the single-object DP solver experiment
2. Here is the code for the two-object DP solver experiment

B.3 Code for Chapter 5

This section contain links to experimental codes of the problem defined in Section 4.3.1 and Section 5.3, for each heuristic.

B.3.1 Code for Problem defined in Section 4.3.1

1. Here is the code for the greedy policy ($k = 0$)
2. Here is the code for 1−step lookahead
3. Here is the code for 2−step lookahead
4. Here is the code for DQN

B.3.2 Code for Problem defined in Section 5.3

1. Here is the code for 1−step lookahead
2. Here is the code for 2−step lookahead
3. Here is the code for DQN