



Planning under Partial Observability and Object Discovery with Reusable Local Strategy Priors

— 24 pt research project (S1 2026)

A report submitted for the course
COMP8800, Advanced Computing Research Project

By:
Zikang Tang

Supervisor:
Dr. Patrik Haslum

May 2026

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the [Academic Integrity Rule](#);
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or LMS course site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

I acknowledge that I am expected to have undertaken Academic Integrity training through the Epigeum Academic Integrity modules prior to submitting an assessment, and so acknowledge that ignorance of the rules around academic integrity cannot be an excuse for any breach.

May (2026), Zikang Tang

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Patrik Haslum, for his guidance and constructive feedback throughout this research project. His advice was especially valuable in helping me clarify the research direction, refine the problem formulation, and think more carefully about the relationship between the proposed planning framework and existing planning models. His comments on the experimental design and result interpretation also helped me improve the empirical analysis and present the limitations of the framework more rigorously.

I am also grateful to the School of Computing and the College of Systems and Society at the Australian National University for providing a supportive academic environment for my study and research. The coursework and research training I received during my Master of Computing program formed an important foundation for this thesis.

Finally, I would like to thank my family, friends, and classmates for their support and encouragement during the completion of this project. In particular, I am deeply grateful to my parents for their continuous support of my education and for giving me the opportunity and encouragement to pursue my studies.

Abstract

Partially observable online planning problems with dynamic object discovery and initially unstable identities are characterised by substantial uncertainty, strong sequential decision-making requirements, and a nontrivial exploration–execution trade-off. Network penetration provides a representative instance of this class of problems, since the agent initially lacks knowledge of the full network topology, node roles, stable identities, and exact connectivity, and must instead discover objects online, update an internal knowledge graph, and complete a target query under a limited budget. To address this setting, this thesis proposes a hierarchical online planning framework that combines knowledge-graph-based state representation, role beliefs, local subnet construction, and budgeted Monte Carlo Tree Search. To preserve local trade-offs among success probability, intermediate progress, and execution cost, the framework uses an ϵ -constraint sweep to retain an approximate set of non-dominated local options rather than collapsing local decisions into a single weighted score. It further abstracts subnet-level search outcomes into action templates indexed by structural signatures in a generalized policy library, enabling cross-episode reuse of local strategies. At the global decision stage, rule-based actions, local subnet candidates, and library-retrieved candidates are aggregated within a unified selection process, while online feedback, slow forgetting, and successful trajectory replay are used to refine stored experience. Experimental results show that pure MCTS remains stronger in raw success robustness and wall-clock runtime. However, when the subnet-based framework succeeds, it often reaches terminal success in fewer interaction steps and preserves more remaining budget. Same-type policy-library transfer provides the strongest evidence for reusable local strategy priors, while cross-type transfer is useful but less reliable. Overall, the results suggest that combining local subnet search, approximate non-dominated option organisation, and abstract strategy reuse can improve interaction-level efficiency and same-type transfer in partially observable online planning problems with dynamic object discovery, while leaving computational efficiency and cross-type robustness as limitations for future work.

Table of Contents

Acknowledgements	iii
1 Introduction	1
1.1 Research Background	1
1.2 Research Problem	2
1.3 Overview of the Proposed Method	3
1.4 Main Contributions	5
2 Background	7
2.1 Sequential Decision Making and Online Planning	7
2.2 Partially Observable Planning and Belief Representation	8
2.3 Dynamic Object Discovery and Object-Centric Representation	9
2.4 Identity Uncertainty and Object Association	10
2.5 Local Search and Monte Carlo Tree Search	11
2.6 Multi-Objective Decision Making and Non-Dominated Trade-Offs	11
2.7 Strategy Abstraction, Action Templates, and Experience Reuse	12
2.8 Chapter Summary	13
3 Related Work	15
3.1 Partially Observable Planning and Online Search	15
3.2 Object-Centric Representation, Relational Abstraction, and Generalized Planning	16
3.3 Local Search, Hierarchical Planning, and Restricted-View Solving	16
3.4 Network Penetration Planning and Uncertain Attack Graph Models	17
3.5 FOND, Strong Cyclic Policies, and Policy-Oriented Solving	18
3.6 Multi-Objective Planning and Approximate Non-Dominated Local Options	18
3.7 Options, Skills, and Transferable Local Strategy Abstraction	19
3.8 Positioning of This Thesis	19
4 Problem Definition and Environment Modelling	21
4.1 Problem Setting and Domain Instantiation	21
4.1.1 Relation to Network Penetration Concepts	22
4.2 Real Environment State	23
4.3 Internal Knowledge State	23
4.3.1 Internal Node Representation	24

Table of Contents

4.3.2	Internal Edge Representation	25
4.4	Observation Model and Identity Association	26
4.5	Action Space	27
4.6	Action Semantics	27
4.6.1	<code>scan</code>	27
4.6.2	<code>traffic_sense</code>	28
4.6.3	<code>move()</code>	29
4.6.4	<code>exploit</code>	29
4.6.5	<code>confirm_identity</code>	30
4.6.6	<code>query.db</code>	31
4.7	Internal State Update	32
4.8	Goal Condition, Budget Constraint, and Evaluation Criteria	32
4.9	Internal Progress and Local Planning Criteria	33
4.10	Formal Problem Statement	34
4.11	Chapter Summary	35
5	Internal State Representation: Knowledge Graph, Identity Association, and Role Beliefs	37
5.1	Knowledge Graph as an Evolving Belief State	37
5.2	Internal Node Records	38
5.3	Internal Edge Records	39
5.4	Observation-Induced Knowledge-Graph Updates	40
5.5	Identity Association	41
5.5.1	Stable-Identity Matching	41
5.5.2	Similarity-Based Association	42
5.6	Object Merging	42
5.7	Role Beliefs	43
5.8	Role-Belief Update	44
5.9	Task Feedback and Database Belief	45
5.10	Use of the Internal Representation in Later Planning Components	45
5.11	Chapter Summary	46
6	Action Models and the Online Interaction Loop	47
6.1	Role of This Chapter	47
6.2	Action Enumeration and Applicability	48
6.3	Operational Roles of Primitive Actions	49
6.4	Heterogeneous Observation Handling	49
6.5	Internal Update Pipeline	50
6.6	Online Interaction Algorithm	51
6.7	Prediction, Execution Feedback, and Model Revision	51
6.8	Chapter Summary	52
7	Local Subnet Search and Approximate Non-Dominated Option Generation	53
7.1	Pure Global MCTS Baseline	53
7.2	Construction of Local Subnets	54

7.3	Frontier Nodes and Frontier Score	55
7.4	Frontier-Directed Path Expansion	57
7.5	Anchor Selection and the Active Subnet Set	58
7.6	Strict Subnet Views and Restricted Solving Domains	59
7.7	Budgeted MCTS Within a Subnet	61
7.8	Progress Score as a Belief-State Potential	62
7.9	ϵ -Sweep and Approximate Non-Dominated Local Options	63
7.10	From Local Options to Global Decision Making	64
7.11	Chapter Summary	65
8	Action-Template Abstraction and the Generalized Policy Library	67
8.1	From Local Options to Action Templates	68
8.2	Movement-Goal and Target-Bucket Abstraction	69
8.3	Template Instantiation in a New Context	69
8.4	Structural Signatures and Local Context Indexing	71
8.5	Subnet Fingerprints and Cache-Level Structure	72
8.6	Library Items and Retrieval	72
8.7	Online Update, Successful Replay, and Slow Forgetting	73
8.8	State-Bucket-to-Template Strategy Abstraction	74
8.9	What the Generalized Policy Library Provides	75
8.10	Chapter Summary	76
9	Global Decision Making and Triggered Rebuilding	77
9.1	The Global Decision Loop	77
9.2	Candidate Instantiation and Unified Candidate Sets	79
9.2.1	Instantiation from Local Options	79
9.2.2	Instantiation from the Generalized Policy Library	80
9.2.3	Rule-Based Fallback Candidates	80
9.3	Global Aggregation Score	81
9.4	Stochastic Selection and Rule-Based Overrides	83
9.5	Progress Monitoring and Anti-Stagnation Mechanisms	83
9.6	Online Feedback to Local and General Sources	85
9.7	Trigger 2: Rebuilding Driven by Local Mismatch	85
9.8	Chapter Summary	86
10	Experimental Design	89
10.1	Experimental Objectives	89
10.2	Environment and Task Setting	90
10.3	Network Buckets and Instances	90
10.4	Compared Methods	91
10.5	Ablation and Sensitivity Settings	91
10.6	Evaluation Metrics	92
10.7	Reporting Protocol	93
10.8	Chapter Summary	94

Table of Contents

11	Experimental Results and Analysis	95
11.1	Reporting and Variability	95
11.2	Pure MCTS versus the Empty-Library Subnet Framework	96
11.3	Policy-Library Transfer	98
11.3.1	Same-Type Transfer	98
11.3.2	Cross-Type Transfer	99
11.4	Ablation and Sensitivity Analysis	100
11.4.1	Frontier-Directed Path Expansion	100
11.4.2	Other Sensitivity Results	102
11.5	Summary of Findings	103
12	Discussion	107
12.1	Trade-Off Relative to Pure MCTS	107
12.2	Role of Strict Subnet Solving	108
12.3	Role of Frontier-Directed Path Expansion	108
12.4	Progress-Aware Decision Making	109
12.5	Probability Gate and Search-Phase Transition	110
12.6	Reusable Local Strategy Learning	110
12.7	Same-Type versus Cross-Type Transfer	111
12.8	Computation-Time Trade-Off	112
12.9	Interpretation of Sensitivity Experiments	112
12.10	Experimental Limitations	113
12.11	Chapter Summary	114
13	Conclusion	115
A	Progress Score Implementation Details	119
A.1	Implementation Form of the Progress Score	119
A.1.1	Database-Potential Term	119
A.1.2	Discovered-Structure Term	120
A.1.3	Local Uncertainty Term	120
A.1.4	Capability Term	120
A.1.5	Role in the Planner	121
	Bibliography	123

Introduction

1.1. Research Background

Planning under partial observability is commonly formalised through partially observable Markov decision processes (POMDPs) and related models of decision making under uncertainty (Kaelbling et al., 1998). More broadly, automated planning studies how an agent can select actions to achieve goals under a specified model of states, actions, and transitions (Ghallab et al., 2004). However, many practical online settings require the agent to act before the full environment model is known. Examples include robotic exploration, network operations, network penetration analysis, and other interactive systems in which the agent must gather information while pursuing a task objective. In autonomous exploration, for example, frontier-based methods explicitly reason about the boundary between known and unknown regions in order to guide information gathering (Yamauchi, 1997).

This thesis studies a related but more specialised setting: partially observable online planning with dynamic object discovery and initially unstable identities. In this setting, the agent begins with limited prior knowledge. It cannot directly access the full environment structure, the true properties of objects, or the exact relations among them. Instead, it must interact with the environment over time, progressively discover new objects, revise its internal state representation, and advance toward its goal under budget or resource constraints. Throughout this process, the agent must actively acquire information while avoiding low-value local probing, move toward high-value regions while controlling execution cost and failure risk, and balance exploration with task-directed progress. These characteristics make the problem sequential and uncertain, while also combining partial observability, object discovery, multi-objective trade-offs, and online model revision.

Network penetration provides a representative test domain for this broader class of problems. Classical attack-graph and network-vulnerability analysis methods typically construct an explicit graph of attack states, dependencies, or paths from known network configuration and vulnerability information (Phillips and Swiler, 1998; Sheyner et al., 2002; Ou et al., 2005, 2006; Wang et al., 2007). Bayesian and probabilistic attack-graph approaches further attach uncertainty or risk estimates to such graph structures (Poolsappasit et al., 2012). These methods are useful for modelling and analysing multi-stage attacks, but they generally assume that the relevant hosts, vulnerabilities, reachability relations, and dependency structure are specified before analysis.

By contrast, the setting studied in this thesis is an online discovery problem. The agent initially does not know the complete network topology, the true roles of hosts, their stable identities, or the location

1. Introduction

of target nodes. It must instead rely on information acquisition, structural sensing, lateral movement, privilege escalation, identity confirmation, and target verification in order to progressively construct an internal understanding of the environment. New nodes and edges enter the agent’s internal representation only through interaction. Moreover, the same real entity may initially appear as a temporary handle or partial local fingerprint, with its stable identity revealed only after further confirmation. As a result, the planner must address not only where to move next, but also whether an observed object corresponds to a previously known entity. For this reason, the problem goes beyond static graph search or shortest-path planning under full information.

Classical planning models usually assume a fixed set of state variables and actions defined before planning begins (Ghallab et al., 2004). Similarly, attack-graph-based security analysis usually starts from known network configuration, reachability, and vulnerability information in order to construct the graph to be analysed (Sheyner et al., 2002; Ou et al., 2005; Wang et al., 2007). These assumptions are appropriate for many offline planning and security-analysis settings, but they do not directly capture the online discovery process considered in this thesis, where the internal object set, edge set, role beliefs, and identity associations are revised during execution.

1.2. Research Problem

This thesis studies the following problem: under a limited budget, how can an agent with only limited initial prior knowledge plan and interact online in a partially observable environment with dynamic object discovery and initially unstable identities, so as to maximise the probability of successful target verification while reducing expected cost and improving search efficiency? To make the problem concrete, this thesis instantiates it in a simulated network penetration domain, where the agent starts from an entry node, progressively discovers network objects and revises its internal knowledge graph, and attempts to access a database node and complete a query successfully.

Compared with static planning and graph-search problems, the problem considered here involves several additional challenges. First, the environment is partially observable: the true structure, object roles, edge reliability, and target locations are not directly available at the start, so the agent must accumulate local evidence through interaction and make decisions under incomplete information. Second, objects and edges are discovered dynamically rather than being present in the internal representation from the beginning. Third, identity association is uncertain: the same real entity may appear in different observations only through unstable local features, forcing the planner to decide whether to merge the observation with an existing object or create a new internal node. Fourth, the problem is naturally multi-objective: success probability, execution cost, and intermediate information-gathering progress represent different criteria, and scalarising them with fixed weights can hide useful trade-offs (Ehrgott, 2005). Fifth, local and global reasoning are tightly coupled. Purely local greedy behaviour can lead to stagnation, while in the implemented setting direct planning over the full discovered graph becomes increasingly costly as the graph expands. Sixth, experience abstraction and transfer are nontrivial: if planning starts from scratch in every episode, useful experience cannot be reused, yet if experience remains tied to a specific instance, it will not transfer to structurally similar networks with different node identities.

1.3. Overview of the Proposed Method

Accordingly, the goal of this thesis is not to develop a domain-specific attack algorithm for network penetration alone. Rather, the goal is to study an online planning framework capable of handling partial observability, object discovery, and identity uncertainty, and to instantiate and evaluate that framework in the structured test domain of simulated network penetration.

The formal problem components, including the hidden true state, observations, actions, action preconditions and effects, budget dynamics, and the internal knowledge state, are defined in Chapter 4. The update rules induced by observations are then described in the chapters on knowledge representation and belief revision. The local subnet construction and local option-generation mechanism are defined in Chapter 7, while the reusable template library and retrieval signatures are defined in Chapter 8. The global candidate-instantiation and aggregation mechanisms are defined in Chapter 9.

1.3. Overview of the Proposed Method

To address the above problem, this thesis develops a hierarchical online planning framework that combines internal state representation, local search, experience abstraction, global action selection, and online revision. The central idea is that, in partially observable environments with dynamic object discovery, useful planning experience should not be stored as concrete trajectories or node-specific actions. Instead, it should be constructed locally, abstracted into state-bucket-to-template strategy priors, reused in structurally similar contexts, and revised through online execution feedback. This thesis studies this process in a simulated network-penetration domain, where objects, identities, and useful structure are progressively discovered during interaction.

This viewpoint explains the organisation of the proposed framework. The knowledge graph provides an evolving object-level belief state. Strict subnet search constructs local planning contexts and generates local trade-off options. Action-template abstraction converts concrete local actions into reusable strategy units. The generalized policy library stores and retrieves these units across episodes. Finally, the global decision layer combines current local search, reusable experience, rule-based fallback candidates, and execution feedback to choose an executable primitive action.

The framework is hierarchical in a limited planning sense: global decision making is supported by locally generated options and reusable action templates. This is related in spirit to temporal abstraction and options in sequential decision making (Sutton et al., 1999), but the options used here are finite-budget local planning outputs rather than learned temporally extended policies. Thus, the framework does not aim to learn a complete generalized policy over all possible networks. Its more modest goal is to learn reusable local first-action priors that can guide online search in similar structural contexts.

First, the framework uses a knowledge graph together with node-level role beliefs as its internal state representation. Knowledge graphs are generally used to represent entities and relations in graph-structured form (Hogan et al., 2021). In this thesis, however, the term knowledge graph is used in a restricted planning sense: it denotes the agent’s internal graph-structured belief state, not a general-purpose semantic knowledge base. It stores discovered objects, temporary handles, confirmed stable identities, role beliefs, edge confidence, estimated movement costs, posterior estimates of movement success, and local semantic evidence. A precise definition of this internal representation is given in Chapter 4, and the way

1. Introduction

observations update this representation is described in the chapters on knowledge representation and online interaction.

Second, rather than performing a single global search directly on the entire discovered graph, the framework constructs a bounded local subnet around the current planning context and runs budgeted Monte Carlo Tree Search (MCTS) inside the restricted local view. MCTS and its UCT variant provide a sampling-based approach to online decision search in large decision spaces (Kocsis and Szepesvári, 2006; Browne et al., 2012). In this thesis, local subnet search is used to reduce the effective branching factor and to focus computation on the parts of the discovered graph that are most relevant to the current decision. The subnet construction also maintains frontier information, where frontier nodes are treated as planning-relevant boundary points between the currently explored local structure and regions that may lead to further discovery. The role of frontier-directed path expansion is defined in Chapter 7 and evaluated by ablation in the experimental chapters.

Third, to avoid committing too early to a single scalar evaluation of local actions, the framework uses an ϵ -constraint sweep to generate an approximate set of non-dominated local options (Mavrotas, 2009). Each local option is evaluated in terms of predicted success, predicted progress, and predicted cost. This allows the planner to retain multiple useful local trade-offs instead of selecting a single locally best action immediately. In particular, an action may be valuable even if it does not immediately increase terminal success probability, because it may reveal new structure, improve role beliefs, support capability gain, or move the agent toward promising frontier regions. The detailed definition of these option values and the pruning procedure is given in Chapter 7.

Fourth, the framework abstracts local search outcomes into action templates that are independent of concrete node identifiers. A concrete action such as moving to a particular node is transformed into an abstract template such as moving toward a database-relevant target selected by an information-gain or database-seeking selector. Together with a structural signature of the local context, this produces a state-bucket-to-template strategy prior. In other words, the library stores recommendations of the form: in this class of local belief states, this type of abstract action is likely to be useful. The precise definition of action templates, selector functions, structural signatures, and library items is given in Chapter 8.

Finally, during global decision making, the planner instantiates local options and library-retrieved templates into executable candidates, combines them with rule-based fallback candidates, and selects one primitive action to execute. After execution, online feedback is used to update local predictions, revise stored library experience, and trigger subnet rebuilding when local predictions become inconsistent with observed outcomes. The global candidate-instantiation, aggregation, feedback, and rebuilding mechanisms are described in Chapter 9.

In this sense, the proposed method does not attempt to answer the classical question of how to compute an optimal path on a fully known graph. Rather, it addresses how to plan, abstract useful local experience, reuse that experience, and adapt online in an environment where objects are progressively discovered, identities are gradually clarified, and local information changes throughout execution.

1.4. Main Contributions

The main contributions of this thesis are as follows.

1. This thesis proposes an online planning framework for environments with partial observability, dynamic object discovery, and initially unstable identities, and instantiates and evaluates the framework in a simulated network penetration domain. Unlike formulations that assume a known object set and fixed identities, the proposed framework explicitly models temporary observations, stable identities, object association, and node merging, allowing the internal representation to be revised continuously as the environment is progressively revealed.
2. This thesis develops a local option generation mechanism based on strict subnets, budgeted Monte Carlo Tree Search, and an ϵ -constraint sweep. Instead of producing a single local action score, the method retains an approximate non-dominated set of local options over predicted success, predicted progress, and predicted cost. This makes the framework better suited to partially observable environments in which immediate success and the creation of future opportunities must be balanced.
3. This thesis introduces a generalized policy library based on structural signatures, enabling the abstraction and reuse of local strategic experience. The library does not depend on concrete node identifiers, but instead uses local structural features and action templates as its basic units of transfer. It is updated through successful trajectory replay, online negative feedback, and slow forgetting. Experimental results show that this form of local experience abstraction can improve interaction efficiency and same-type transfer performance, while also exhibiting limited but less reliable transferability across different network types.
4. This thesis integrates local search, global aggregation, and online rebuilding into a single closed-loop decision process. Through progress shaping and prediction-execution mismatch handling, the system can preserve search direction when no clear terminal opportunity is yet visible, and can adaptively rebuild affected subnets and refresh local frontiers when local models become unreliable. As a result, the framework supports not only one-shot local solving, but also continuous revision of local models and reusable experience throughout ongoing interaction.

Background

This chapter introduces the technical background required to understand the problem formulation and method developed in this thesis. The purpose of this chapter is not to review all recent literature, but to establish the main concepts, notation, and standard technical ideas on which the remainder of the thesis builds. In particular, this chapter covers sequential decision making and online planning, partially observable planning and belief representation, dynamic object discovery, identity association, local search and Monte Carlo Tree Search, multi-objective decision making through non-dominated trade-offs, and the abstraction and reuse of local strategic experience.

2.1. Sequential Decision Making and Online Planning

The problem studied in this thesis belongs to the general class of sequential decision-making problems. In sequential decision making, an agent repeatedly interacts with an evolving environment rather than computing a complete decision once and for all. Each action may produce an immediate reward or cost, but it also changes the future decision space by affecting the subsequent state of the environment and the information available to the agent. Markov decision processes provide a standard mathematical foundation for many such problems when the state is fully observable and the transition model is specified (Puterman, 1994). Reinforcement learning and online planning further emphasise decision making through interaction, feedback, and incremental improvement of behaviour (Sutton and Barto, 2018).

At a high level, the interaction process considered in this thesis can be written as

$$(I_t, a_t) \mapsto (o_{t+1}, c_t, I_{t+1}), \quad (2.1)$$

where I_t denotes the agent’s internal state at time t , a_t the action chosen at time t , o_{t+1} the observation returned by the environment, c_t the execution cost, and I_{t+1} the updated internal state. This notation is not intended to define a complete probabilistic model by itself; rather, it introduces the closed-loop interaction structure used throughout the thesis. The full problem components are defined formally in Chapter 4.

A trajectory can therefore be represented as

$$\tau = (I_0, a_0, o_1, c_0, I_1, a_1, o_2, c_1, \dots, I_T). \quad (2.2)$$

2. Background

Given a finite budget B , a cumulative execution cost can be written as

$$C(\tau) = \sum_{t=0}^{T-1} c_t, \quad (2.3)$$

with the budget constraint

$$C(\tau) \leq B. \quad (2.4)$$

Unlike classical offline planning, in which a complete plan may be computed before any execution takes place, online planning allows the agent to revise its internal model continuously and make decisions incrementally during execution (Ghallab et al., 2004). This is especially suitable for settings in which the environment is only partially known, objects are revealed progressively, and action outcomes are uncertain. In such settings, planning is not merely search over a fixed state space; it is a closed-loop process that combines information acquisition, model revision, and task-directed action selection.

2.2. Partially Observable Planning and Belief Representation

In fully observable environments, the agent can directly access the current true state and plan accordingly. In many realistic domains, however, the full state is not directly observable. Instead, the agent receives only partial, local, or noisy observations. The standard formal framework for such problems is the Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998).

A POMDP is commonly defined by the tuple

$$\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R \rangle, \quad (2.5)$$

where $\mathcal{S}$ is the hidden state space, $\mathcal{A}$ the action space, $\mathcal{O}$ the observation space, $T(s' | s, a)$ the transition model, $O(o | s', a)$ the observation model, and $R(s, a)$ the reward function. Because the true state $s_t \in \mathcal{S}$ is not directly accessible, the agent maintains a belief state, typically denoted by b_t , where

$$b_t(s) = \Pr(s_t = s | a_{0:t-1}, o_{1:t}). \quad (2.6)$$

The standard Bayesian belief update takes the form

$$b_{t+1}(s') \propto O(o_{t+1} | s', a_t) \sum_{s \in \mathcal{S}} T(s' | s, a_t) b_t(s). \quad (2.7)$$

This formulation highlights a key feature of planning under partial observability: the value of an action depends not only on how far it moves the agent toward the goal, but also on how much information it reveals. An action that does not immediately improve terminal success may still be valuable because it reduces uncertainty and improves later decisions.

The problem studied in this thesis is closely related to POMDPs, since the agent must also act under partial observability and continuously revise internal beliefs from action–observation feedback. However, the problem is not modelled as a standard finite-state POMDP. The main reason is that the relevant object set is not fixed in advance, but is discovered progressively through interaction. Moreover, the identity of an observed entity may itself be uncertain and require later confirmation. For this reason,

2.3. Dynamic Object Discovery and Object-Centric Representation

the internal representation used in this thesis is expressed as an object-centric graph-structured belief state rather than as a fixed-dimensional global belief vector.

Belief nevertheless remains a central concept in this setting. Rather than maintaining a single global distribution over all possible hidden states, the framework developed in this thesis maintains local beliefs over key properties of the environment, including node-role beliefs, edge existence confidence, and posterior estimates of action success. In this broader sense, belief provides the internal basis for decision making under incomplete information. The concrete internal belief representation used in this thesis is defined in Chapter 4 and updated through the knowledge-maintenance procedures described in later chapters.

2.3. Dynamic Object Discovery and Object-Centric Representation

Many classical planning models assume that the set of relevant objects or state variables is fixed before planning begins (Ghallab et al., 2004). Object-centric and relational representations relax this flat-state view by representing environments in terms of objects, properties, and relations, which can support generalisation across structurally similar states (Boutilier et al., 2001; Diuk et al., 2008). In the problems considered here, however, there is an additional difficulty: the agent does not initially know the complete object set. Additional objects and relations become visible only through interaction. This leads to a planning problem with dynamic object discovery.

To represent such a setting, it is useful to maintain an evolving discovered graph

$$\hat{G}_t = (\hat{V}_t, \hat{E}_t), \quad (2.8)$$

where $\hat{V}_t$ is the set of currently discovered internal objects and $\hat{E}_t$ is the set of currently discovered or hypothesised relations among them. In the context of this thesis, this discovered graph is part of the agent’s internal knowledge graph. More generally, knowledge graphs provide graph-structured representations of entities and relations (Hogan et al., 2021). Here, however, the term is used in a restricted planning sense: the knowledge graph is the agent’s internal, evolving, graph-structured belief state, not a general-purpose semantic knowledge base.

In an object-centric representation, each node $v \in \hat{V}_t$ may carry locally maintained information such as observational fingerprints, role beliefs, uncertainty estimates, and visitation statistics, while each edge $e \in \hat{E}_t$ may carry information such as existence confidence, estimated cost, and estimated success probability. The exact node and edge fields used in the implemented framework are defined in Chapter 4 and further described in the chapters on knowledge representation and belief update.

Dynamic discovery means that the internal representation is not fixed. In simple discovery-only settings one might have

$$\hat{V}_t \subseteq \hat{V}_{t+1}, \quad \hat{E}_t \subseteq \hat{E}_{t+1}, \quad (2.9)$$

but this monotonic view is too simple for the setting studied here. Because identity association may later merge internal nodes, and because edge beliefs may be revised as new evidence arrives, the discovered graph can expand, be corrected, and be reorganised over time.

2. Background

Compared with flat state-vector representations, an object-centric graph is better suited to problems in which object existence itself remains uncertain. It also supports structural reasoning, local subnet extraction, and strategy abstraction over relational patterns. In this thesis, the knowledge graph is therefore not merely a passive data structure for storing observations; it forms the basis of local search, experience abstraction, and global decision making.

2.4. Identity Uncertainty and Object Association

A further difficulty in dynamic object discovery problems is that newly observed objects do not necessarily come with stable identities. The same real-world entity may appear at different times only through temporary handles, partial fingerprints, or incomplete local attributes. In this case, the agent must determine whether the current observation corresponds to a previously known object or should instead be treated as a new entity. This is related to the broader problem of record linkage or entity resolution, where the task is to decide whether different observed records refer to the same underlying entity ([Fellegi and Sunter, 1969](#)).

Let x denote a newly observed entity description. The internal update problem can be viewed abstractly as deciding among three possibilities:

$$x \rightarrow \text{CREATE}, \quad x \rightarrow \text{ASSOCIATE}, \quad x \rightarrow \text{MERGE}. \quad (2.10)$$

Here, CREATE means introducing a new internal object, ASSOCIATE means attaching the new evidence to an existing object without restructuring the graph, and MERGE means unifying two internal objects once stronger identity evidence becomes available.

This can be represented more generally by an update operator

$$\hat{G}_{t+1} = U(\hat{G}_t, x, o_{t+1}), \quad (2.11)$$

where the operator U may perform creation, association, or merging depending on the available evidence. The importance of this decision is substantial. If the same real entity is split incorrectly into multiple internal objects, beliefs become fragmented and structural reasoning becomes unreliable. If distinct entities are merged incorrectly, the resulting graph may be semantically inconsistent and lead to poor action choices.

A useful conceptual distinction is therefore that between temporary observable identifiers and stable identity information. Temporary identifiers may be available early but are not sufficient for reliable association; stable identity information often requires stronger evidence or dedicated confirmation actions. Identity uncertainty is thus not merely a perception issue, but a central part of the planning problem itself. The specific identity-confirmation and node-merging rules used in this thesis are defined in the formal problem and knowledge-update chapters.

2.5. Local Search and Monte Carlo Tree Search

In environments with progressive discovery and continually changing local structure, direct planning over the entire discovered graph may become expensive because the branching factor grows with the graph and with the number of available information-gathering actions. A natural response is to employ local search: construct a restricted search problem around the current local structure and reason within that smaller view. In this thesis, this restricted view is called a local subnet and is defined formally in Chapter 7.

Monte Carlo Tree Search (MCTS) is a family of sampling-based heuristic search methods for sequential decision problems. It has been widely used in large search spaces because it can estimate action values through simulated trajectories without exhaustively enumerating the full state space (Kocsis and Szepesvári, 2006; Browne et al., 2012). The standard MCTS cycle consists of four stages: selection, expansion, simulation or rollout, and backup. Through repeated sampling of possible future trajectories, MCTS estimates the value of actions without requiring a globally exact value function.

A typical selection rule is based on the Upper Confidence Bound principle. If a node has been visited N times and one of its children has been visited n times with empirical value $\bar{X}$, then a common selection score is

$$\bar{X} + c\sqrt{\frac{\ln N}{n}}, \quad (2.12)$$

where $c > 0$ controls the exploration–exploitation trade-off (Kocsis and Szepesvári, 2006). This equation is not specific to the present thesis, but illustrates the standard idea that a promising child should be selected on the basis of both empirical value and uncertainty.

In the setting considered in this thesis, MCTS is not applied directly to the entire discovered graph. Instead, it is applied to restricted local subnets under explicit budget constraints. This is important because a locally promising plan should be evaluated not only in terms of eventual success, but also in terms of whether it can be executed within the available resources. For this reason, local MCTS is used to generate alternatives involving different trade-offs among predicted success, predicted progress, and predicted cost, rather than a single scalar recommendation.

2.6. Multi-Objective Decision Making and Non-Dominated Trade-Offs

In single-objective optimisation, candidate decisions can be ranked by a single scalar value. In the present problem, however, local actions often affect several competing criteria at once. A candidate action may increase the probability of eventual success but incur a higher immediate cost, while another may be inexpensive but provide only intermediate progress. Local decision making is therefore naturally multi-objective. Multi-objective optimisation studies such settings by considering trade-offs among several criteria rather than reducing all decisions to a single objective from the beginning (Ehrgott, 2005).

Suppose a local candidate option o is evaluated by three quantities:

$$p(o) \quad (\text{predicted success}), \quad g(o) \quad (\text{predicted progress}), \quad c(o) \quad (\text{predicted cost}). \quad (2.13)$$

2. Background

For two options o_i and o_j in a finite candidate set, one may say that o_i dominates o_j if

$$p(o_i) \geq p(o_j), \quad g(o_i) \geq g(o_j), \quad c(o_i) \leq c(o_j), \quad (2.14)$$

and at least one of these inequalities is strict. The non-dominated candidates are those not dominated by any other candidate in the same set.

This distinction is important for the terminology used later in the thesis. In formal multi-objective optimisation, a Pareto set usually refers to the complete set of Pareto-optimal solutions over the full feasible decision space, or to a set that represents all non-dominated objective vectors (Ehrgott, 2005). The local solver in this thesis does not compute such a complete mathematical Pareto set. Instead, because local options are generated under finite MCTS budget and from a restricted local subnet, the method retains an approximate set of non-dominated local options from the candidates actually sampled and evaluated. Throughout the rest of the thesis, this set should therefore be understood as an approximate local trade-off set, not as an exact Pareto set.

The advantage of this organisation is that it preserves multiple locally meaningful trade-offs instead of forcing all candidates into a single scalar score too early. In the context of this thesis, progress does not denote terminal success itself, but rather intermediate value that may enable future success, such as revealing useful structure, reducing uncertainty, improving role beliefs, acquiring credentials, or moving toward promising frontier regions. Retaining non-dominated local alternatives allows these different forms of value to coexist until later decision stages, where remaining budget, global context, policy-library candidates, and rule-based actions can be taken into account.

The ϵ -constraint method provides one standard way to explore multi-objective trade-offs by optimising one objective while imposing bounds on others (Mavrotas, 2009). In this thesis, an ϵ -constraint sweep is used as a practical local option-generation mechanism. Its implementation and the definitions of predicted success, predicted progress, and predicted cost are given in Chapter 7.

2.7. Strategy Abstraction, Action Templates, and Experience Reuse

If planning begins from scratch in every episode, the system cannot exploit useful experience accumulated in previous runs. On the other hand, if experience is stored only in terms of concrete object identifiers or node indices, it becomes difficult to transfer that experience to new instances. A useful intermediate representation is therefore needed to abstract from instance-specific actions to structurally meaningful strategy units.

This idea is related to temporal abstraction in sequential decision making, where actions can be organised into higher-level behaviours or options (Sutton et al., 1999). The templates used in this thesis are not standard options in the reinforcement-learning sense: they are not learned temporally extended policies with initiation sets and termination conditions. Instead, they are local action abstractions extracted from successful or useful planning decisions. Their purpose is to make local experience less dependent on concrete node identities.

2.8. Chapter Summary

Let a denote a concrete local action. An abstraction operator may be written as

$$\tau = \Gamma(a), \quad (2.15)$$

where τ is an action template. Rather than encoding a specific action such as “move to node 17”, an action template may encode a structurally meaningful instruction such as moving toward a likely target, moving toward a potential credential source, or performing lateral exploration according to a selector.

To make such experience reusable, the system also requires a structural description of the local context. Let

$$\sigma = \Sigma(I_t) \quad (2.16)$$

denote a structural signature extracted from the current internal state. The signature does not encode the entire graph, but rather those local properties that are most relevant to action choice, such as target relevance, frontier structure, role beliefs, privilege state, and uncertainty.

Together, an action template and a structural signature define a reusable experience unit. At an abstract level, a generalized policy library may be viewed as a mapping

$$(\sigma, \tau) \mapsto \eta, \quad (2.17)$$

where η denotes the stored experience associated with taking template τ in a local structural context of type σ . This stored experience is not a complete policy. In this thesis, it consists of statistics such as predicted success, predicted progress, predicted cost, similarity, and recency-weighted estimates. In this sense, the generalized policy library stores local decision priors rather than node-specific actions or full multi-step control policies. The precise definition of templates, selector evaluation, structural signatures, and library retrieval is given in Chapter 8.

2.8. Chapter Summary

This chapter has introduced the main technical background required for the remainder of the thesis, including sequential decision making and online planning, partially observable planning and belief representation, dynamic object discovery, identity association, local search and Monte Carlo Tree Search, multi-objective decision making through non-dominated trade-offs, and the basic ideas of action templates and experience reuse. Together, these concepts form the conceptual foundation for the formal problem definition and method design that follow in the next chapters.

Related Work

This chapter positions the thesis relative to several bodies of prior work: online planning under partial observability, object-centric and relational representation, generalized planning, local and hierarchical search, network penetration planning, FOND planning, multi-objective sequential decision making, and reusable strategy abstraction. The aim is not to provide an exhaustive survey of each field, but to clarify which ideas this thesis builds on and where the proposed framework differs from existing formulations.

3.1. Partially Observable Planning and Online Search

Partially observable planning is commonly formalised through the framework of Partially Observable Markov Decision Processes (POMDPs). Classical work on POMDPs established both the expressive power of this framework and the computational challenges that arise from the growth of state, observation, and history spaces (Kaelbling et al., 1998). Because exact POMDP solving is computationally difficult in large domains, a substantial body of work has developed approximate and online planning methods.

Representative online POMDP planners include POMCP, which combines particle-based belief representation with Monte Carlo Tree Search for large POMDPs (Silver and Veness, 2010), and DESPOT, which uses scenario sampling and regularisation to construct a sparse approximation of the belief tree (Ye et al., 2017). These methods demonstrate that online search can support decision making under partial observability without explicitly enumerating the complete belief space.

The present thesis is closely related to this tradition. The agent cannot observe the true environment state directly and must update internal beliefs from action–observation feedback during execution. In this respect, the framework follows the general closed-loop structure of online POMDP planning. However, the setting studied here is not modelled as a standard finite-state POMDP. The relevant object set and relation set are not fixed at the outset, but are discovered progressively through interaction. In addition, early observations may provide only temporary handles or partial fingerprints, so object identity itself can be uncertain. The contribution of this thesis is therefore not a new general-purpose POMDP solver, but an object-centric online planning framework that combines partial observability with dynamic object discovery, identity association, graph-structured belief maintenance, local subnet solving, and cross-episode reuse of local strategic experience.

3. Related Work

3.2. Object-Centric Representation, Relational Abstraction, and Generalized Planning

Object-centric and relational representations are often more suitable than flat propositional encodings when decision quality depends on entities, attributes, and relations. Object-Oriented MDPs (OO-MDPs), for example, model environments in terms of objects and their interactions, enabling more structured representations and potential generalisation across related states (Diuk et al., 2008). Relational and first-order approaches to decision making similarly aim to exploit repeated structure across objects and problem instances (Boutilier et al., 2001).

A related line of work is generalized planning, which seeks strategies that solve multiple instances of a domain rather than a single concrete problem. Work by Bonet, Geffner, and collaborators has shown that generalized plans can be expressed over abstract features and abstract actions, allowing related concrete instances to be represented at a common level of abstraction (Bonet and Geffner, 2018; Bonet et al., 2019). This line of work is relevant because it demonstrates the value of moving from instance-specific plans toward reusable structural descriptions of action choice.

The present thesis is connected to both object-centric representation and generalized planning, but differs in scope and mechanism. The environment is represented internally through a graph-structured object-level belief state with role beliefs, edge confidence, temporary handles, and stable identities. This is consistent with the object-centric view that objects and relations should be explicit modelling units. However, the thesis does not attempt to compute a complete generalized policy for an entire problem family. Instead, it abstracts local search outcomes into action templates indexed by structural signatures. The resulting mechanism is better described as *local strategy abstraction and transfer* than as full generalized planning.

3.3. Local Search, Hierarchical Planning, and Restricted-View Solving

Hierarchical planning and decomposition are common strategies for reducing the complexity of large planning problems. Hierarchical planning introduces abstraction layers, subtask decompositions, or higher-level control structures so that the original problem can be solved through smaller and more manageable units (Bercher et al., 2019). In reinforcement learning and sequential decision making, related ideas appear in temporal abstraction and the options framework, where temporally extended behaviours are treated as higher-level decision units (Sutton et al., 1999). These lines of work address a common challenge: direct global search in a complex environment may be computationally expensive or poorly structured.

The framework in this thesis is related to hierarchical and localised search, but it does not assume a fixed task hierarchy, a predefined HTN decomposition, or a library of complete temporally extended options. Instead, it constructs a restricted local search view dynamically from the current internal knowledge graph. This restricted view is called a strict subnet in later chapters. The purpose of the subnet is to

3.4. Network Penetration Planning and Uncertain Attack Graph Models

focus local MCTS computation on a bounded part of the discovered graph while still preserving enough structure to support progress toward deeper or more informative regions.

This makes the proposed method different from both classical global graph search and standard hierarchical planning. The subnet is generated online, revised as new objects and edges are discovered, and rebuilt when local predictions become inconsistent with execution outcomes. Furthermore, local subnet solving does not directly produce a final policy for the whole task. It produces local candidate options with predicted success, progress, and cost estimates, which are later combined with rule-based actions and policy-library candidates by the global aggregation procedure. The method is therefore best understood as an object-centric, online, reconstructable local approximation framework rather than as a straightforward extension of classical hierarchical planning.

3.4. Network Penetration Planning and Uncertain Attack Graph Models

Network penetration and vulnerability analysis have often been modelled using attack graphs, vulnerability dependency graphs, or attack-path search. Early attack-graph work typically constructs an explicit graph of attack states, dependencies, or paths from known network configuration and vulnerability information (Phillips and Swiler, 1998; Sheyner et al., 2002). Logic-based systems such as MulVAL generate attack graphs from declarative descriptions of vulnerabilities, host configurations, and network reachability (Ou et al., 2005, 2006). Other work uses attack graphs to derive security metrics or to attach probabilistic risk estimates to attack paths (Wang et al., 2007; Poolsappasit et al., 2012).

These approaches are highly relevant because they show how network penetration can be represented as a structured multi-stage problem rather than as an isolated single-step action. However, they usually assume that a substantial amount of network structure, reachability information, and vulnerability information is available before analysis. This differs from the online discovery setting studied in this thesis, where the internal graph is built gradually through interaction and may need to be revised when identities are confirmed or when observations reveal new structure.

More closely related to this thesis is work that formulates penetration testing as planning under uncertainty. Sarraute, Buffet, and Hoffmann argued that penetration testing can be modelled as a POMDP, integrating scanning, exploitation, uncertain action outcomes, and incomplete knowledge into a decision-theoretic framework (Sarraute et al., 2012, 2013). This line of work makes clear that penetration testing is not merely a static path-search problem; it can also be viewed as a sequential decision problem under uncertainty.

The present thesis shares this application domain, but places emphasis on different modelling and algorithmic issues. Prior POMDP-style penetration-planning work focuses on integrating scanning and exploitation into a unified uncertain decision model. By contrast, this thesis further emphasises dynamic object discovery, initially unstable identities, internal graph revision, strict subnet solving, and reusable local strategy abstraction. Network penetration is therefore used here primarily as a structured test domain for studying online planning with partial observability, object discovery, identity association, budget constraints, and reusable local experience.

3. Related Work

3.5. FOND, Strong Cyclic Policies, and Policy-Oriented Solving

Fully Observable Non-Deterministic (FOND) planning studies how to generate policies rather than single linear plans when the environment is fully observable but actions may have multiple nondeterministic outcomes. Foundational work on weak, strong, and strong cyclic planning characterises different notions of solution under nondeterministic action outcomes, including plans that may succeed, plans that guarantee success, and cyclic policies that guarantee progress under fairness assumptions (Cimatti et al., 2003).

The present thesis is related to FOND primarily through this policy-oriented viewpoint. If partial observability, dynamic object discovery, and identity uncertainty were removed, then stochastic success or failure of actions such as `move`, `exploit`, and `query_db` would create a structure reminiscent of non-deterministic planning. However, the thesis setting cannot be reduced to standard FOND. First, the environment is not fully observable. Second, the internal object set and relation set are not completely enumerated before planning begins. Third, observations can change the internal representation by creating, associating, or merging objects. These features lie outside standard FOND models defined over a fixed fully observable state space. The relation to FOND is therefore conceptual rather than direct: this thesis also seeks reusable decision rules rather than single paths, but it does so in a partially observable online discovery setting.

3.6. Multi-Objective Planning and Approximate Non-Dominated Local Options

Multi-objective sequential decision making studies planning and learning problems in which outcomes are evaluated by multiple criteria rather than by a single scalar reward. Roijers et al. survey such problems and discuss cases in which scalarising all objectives into a single reward can be impossible, infeasible, or undesirable (Roijers et al., 2013). This is relevant to the present thesis because local actions may trade off predicted terminal success, intermediate progress, execution cost, risk, and information value.

Classical multi-objective optimisation uses Pareto dominance to reason about such trade-offs (Ehrgott, 2005). The ϵ -constraint method provides one standard way to explore trade-offs by optimising one objective while constraining others (Mavrotas, 2009). These ideas motivate the local option organisation used in this thesis. The local solver in this thesis does not compute a complete Pareto set over the full feasible decision space. Instead, it operates inside a restricted subnet and uses finite-budget MCTS. As a result, it only observes a finite set of sampled candidate options.

For this reason, the method should be understood as retaining an *approximate non-dominated local option set*, not an exact Pareto frontier. Each local option is associated with predicted success, predicted progress, and predicted cost. Dominance pruning is applied only to the sampled candidates produced by the local solver. This design is useful because it avoids reducing all local decisions to one fixed linear score too early, while remaining computationally feasible in an online setting. It allows the planner to retain actions that may support future progress, even when they do not immediately maximise predicted

3.7. Options, Skills, and Transferable Local Strategy Abstraction

terminal success. The final action is then selected later by global aggregation, which also considers remaining budget, rule-based actions, policy-library candidates, and current execution context.

3.7. Options, Skills, and Transferable Local Strategy Abstraction

In reinforcement learning and hierarchical decision making, temporal abstraction and the options framework provide a formal way to represent temporally extended behaviours. A classical option is defined by an initiation set, an internal policy, and a termination condition (Sutton et al., 1999). This allows decision making to occur at a higher level than primitive actions and can support more efficient learning or planning when useful sub-behaviours recur across tasks.

The present thesis is related to this idea because it also seeks to transform repeated local decision patterns into reusable strategic units. However, it does not adopt the full classical option formalism. The stored units in this thesis are not complete multi-step closed-loop policies with their own termination conditions. Instead, the framework stores action templates, structural signatures, and statistical experience about predicted success, progress, and cost. These templates are used as local decision priors and are re-instantiated in new contexts by selector functions.

The policy-library mechanism in this thesis is also related to inductive policy selection for first-order MDPs. Yoon, Fern, and Givan learn policies for compact first-order MDP representations by inducing decision-list policies in a taxonomic concept language from training data generated on small instances (Yoon et al., 2002). The common motivation is to avoid memorising concrete states and instead learn action choices that generalise across object configurations. The representation used in this thesis is weaker and more lightweight: it does not use an explicit first-order policy language, quantified relational formulas, or symbolic decision lists. Instead, it maps discretised structural signatures to abstract action templates with learned success, progress, and cost statistics. The resulting library is therefore best understood as a relational-policy-inspired state-bucket-to-template abstraction rather than as a fully symbolic first-order policy.

Thus, the proposed library is closer to a transferable library of local decision priors than to a conventional option library. This design is intentionally lightweight: it can be coupled directly with finite-budget local MCTS and updated through successful trajectory replay, online negative feedback, and slow forgetting. Its limitation is that generalisation depends on the quality of the structural signatures and template selectors. This is why cross-type transfer is more difficult than same-family transfer in the experiments.

3.8. Positioning of This Thesis

Taken together, the above research traditions position this thesis at the intersection of several areas. It is not a standard POMDP solver, because the object set and identity structure are not fixed in advance. It is not a standard FOND planner, because the environment is partially observable and the internal graph grows and reorganises during execution. It is not a classical attack-graph method, because the

3. Related Work

graph is not fully specified before planning begins. It is also not a standard options framework, because the reusable units are not full temporally extended policies.

Instead, the thesis proposes an online planning framework for budget-constrained interaction under partial observability, dynamic object discovery, and initially unstable identities. At the representation level, it uses an object-centric knowledge graph with role beliefs and identity association. At the search level, it uses strict-subnet local MCTS and approximate non-dominated local option generation. At the experience level, it uses structural signatures and action templates to reuse local strategic priors across episodes. The central contribution is therefore the integration of graph-structured belief maintenance, local restricted-view solving, approximate non-dominated option organisation, and reusable local strategy abstraction into a single online decision loop.

Problem Definition and Environment Modelling

This chapter formalises the problem studied in this thesis and describes the environment model used for empirical evaluation. The target problem is a budget-constrained online planning problem under partial observability, dynamic object discovery, and initially unstable identities. To make the problem concrete, the thesis instantiates it in a simulated network penetration domain. This domain is not intended as the only possible application of the proposed framework; rather, it serves as a structured test environment for studying the joint effects of partial observability, object discovery, identity uncertainty, stochastic actions, capability preconditions, and online planning.

The formulation is related to partially observable planning in the POMDP tradition ([Kaelbling et al., 1998](#)), but differs from standard finite-state models in two important ways. First, the relevant object set is not fixed in advance: new internal objects and relations may be created during execution. Second, early observations do not necessarily provide stable object identities, so the planner must also reason about whether a newly observed entity should be created as a new internal object, associated with an existing object, or merged after stronger identity evidence becomes available. The formulation is also related to object-centric state representations, in which entities and their relations form the basic modelling units ([Diuk et al., 2008](#)), and to network penetration planning under uncertainty ([Sarraute et al., 2012](#)). However, the focus of this thesis is not to propose a new domain-specific penetration-testing algorithm. The domain is used as a testbed for a more general planning setting involving online discovery and evolving internal representation.

4.1. Problem Setting and Domain Instantiation

We consider an agent interacting with an initially partially unknown environment. The agent has an initial budget $B_0 > 0$, starts from an entry object, and repeatedly selects actions until either the task succeeds or the available budget is exhausted. The agent does not initially know the full environment topology, the complete object set, the roles of all objects, the reliability and cost of all traversable relations, or the stable identities of newly observed objects.

In the simulated network penetration domain, real objects correspond to network entities such as entry hosts, intermediate hosts, credential-relevant hosts, and database-related hosts. The agent must discover the network structure through interaction, gain sufficient capability when needed, identify or reach a

4. Problem Definition and Environment Modelling

true database host, and execute a successful database query. The task succeeds only when `query.db` succeeds under the required true conditions.

4.1.1. Relation to Network Penetration Concepts

The simulated domain abstracts several common concepts from network penetration testing, but it does not attempt to reproduce the full technical detail of real penetration-testing workflows. Its purpose is to provide a controlled planning environment in which network reachability, capability acquisition, information gathering, and target verification can be studied under partial observability.

Table 4.1 summarises the correspondence between common network-penetration concepts and the abstract model used in this thesis.

This abstraction makes several deliberate simplifications. A connection in the model should be understood as network reachability or possible lateral access, not as physical connectivity. A vulnerability is represented through the abstract `exploit` action and node security attributes, rather than through a specific vulnerability database entry or CVE identifier. Similarly, the database query action represents terminal target verification, not a detailed database exploitation procedure. These simplifications allow the experiments to focus on the planning problem: how an agent should gather information, move through a partially known network, gain required capability, and verify a target under a limited budget.

This domain is representative of a broader class of online planning problems because it combines six difficulties:

1. **Partial observability:** the agent cannot directly observe the full true environment state.
2. **Dynamic object discovery:** nodes and edges are added to the internal representation only after observations.
3. **Identity uncertainty:** early observations may provide temporary handles or partial fingerprints rather than stable identities.
4. **Stochastic execution:** actions such as movement and exploitation may succeed or fail.
5. **Capability preconditions:** some terminal or high-value actions require privilege or credentials.
6. **Budget constraint:** every action consumes budget, so information gathering and goal-directed execution must be balanced.

The remainder of this chapter defines these components explicitly.

4.2. Real Environment State

The simulator maintains a hidden true environment graph

$$G^* = (V^*, E^*), \quad (4.1)$$

where each node $v \in V^*$ represents a real environment object and each edge $(v, w) \in E^*$ represents a true traversable relation. The planner never has direct access to the full graph G^* .

Each true node v has a latent node state

$$z_v^* = (\text{id}_v^*, r_v^*, \phi_v^*, \text{sec}_v^*), \quad (4.2)$$

where id_v^* is a stable identity, r_v^* is the true role of the node, ϕ_v^* is a collection of observable or partially observable fingerprint attributes, and sec_v^* denotes security- or difficulty-related attributes. In the network domain, the true role r_v^* may indicate whether a node is an entry node, an intermediate node, a credential-relevant node, a database node, or another supporting node. Fingerprint attributes may include ports, services, certificates, addresses, and other local features. Not all of these attributes are revealed by every observation.

Each true edge (v, w) has an edge state

$$z_{vw}^* = (p_{vw}^{\text{move},*}, c_{vw}^{\text{move},*}), \quad (4.3)$$

where $p_{vw}^{\text{move},*}$ is the true probability that a movement attempt along the edge succeeds and $c_{vw}^{\text{move},*}$ is the true or expected movement cost. The realised cost may include noise, so the same action can have slightly different execution costs in different trials.

At time t , the true execution state is

$$s_t^* = (G^*, \ell_t^*, \pi_t^*, \kappa_t^*, h_t^*), \quad (4.4)$$

where $\ell_t^* \in V^*$ is the true current location, π_t^* is the true privilege or capability state, $\kappa_t^* \in \{0, 1\}$ indicates whether the required credential condition is satisfied, and $h_t^* \in \{0, 1\}$ is the true task-success flag. The success flag becomes true only after a successful `query_db` action.

4.3. Internal Knowledge State

The planner acts on an internal knowledge state rather than on the true state. At time t , the internal state is

$$I_t = (\hat{G}_t, \hat{\ell}_t, \hat{\pi}_t, \hat{\kappa}_t, B_t, h_t), \quad (4.5)$$

where $\hat{G}_t = (\hat{V}_t, \hat{E}_t)$ is the current internal knowledge graph, $\hat{\ell}_t \in \hat{V}_t$ is the internal current location, $\hat{\pi}_t$ is the planner's current privilege estimate over known internal objects, $\hat{\kappa}_t \in \{0, 1\}$ is the internal credential flag, B_t is the remaining budget, and $h_t \in \{0, 1\}$ is the internal success flag.

The term *knowledge graph* is used here in a restricted planning sense. More generally, knowledge graphs represent entities and relations in graph-structured form (Hogan et al., 2021). In this thesis, the internal

4. Problem Definition and Environment Modelling

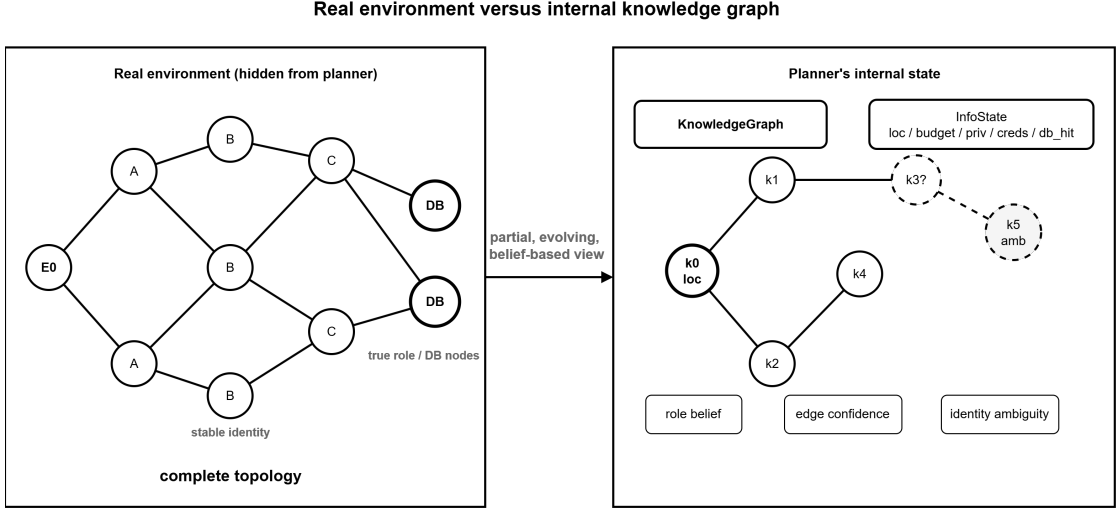


Figure 4.1.: Real environment versus internal planner representation. The simulator maintains the true environment graph with stable identities and true node roles, whereas the planner acts only on a partial and evolving knowledge graph together with runtime state variables. The internal representation may contain uncertain role beliefs, edge-confidence annotations, and identity ambiguity.

knowledge graph is the agent's evolving graph-structured belief state. It stores discovered objects, uncertain roles, temporary handles, confirmed stable identities, edge-confidence estimates, movement-cost estimates, and statistics used by local search and experience reuse. It is not assumed to be complete, correct, or globally consistent at every time step.

The separation between s_t^* and I_t is fundamental. The simulator updates s_t^* according to the true hidden environment and the stochastic outcome of actions. The planner updates I_t only through observations. Therefore, all local search, global aggregation, and experience reuse in this thesis operate on I_t , not on s_t^* .

Figure 4.1 illustrates this distinction.

4.3.1. Internal Node Representation

Each internal node $\hat{v}_i \in \hat{V}_t$ is associated with a local node state

$$x_i^t = (H_i^t, \zeta_i^t, F_i^t, b_i^t, d_i^t, \hat{\pi}_i^t, q_i^t). \quad (4.6)$$

The components are as follows:

- H_i^t is the set of temporary handles or local identifiers that have been associated with this internal node.

4.3. Internal Knowledge State

- ζ_i^t is the stable identity if it has been confirmed; otherwise $\zeta_i^t = \perp$.
- F_i^t is the accumulated fingerprint evidence, such as observed ports, services, certificate fragments, addresses, or other local attributes.
- b_i^t is a role-belief vector over a finite role set $\mathcal{R}$:

$$b_i^t(r) = \Pr(r_i = r \mid I_t), \quad r \in \mathcal{R}. \quad (4.7)$$

The role set may include entry, intermediate, credential-relevant, database-related, and other roles.

- $d_i^t \in [0, 1]$ is a derived database-potential or target-relevance score computed from role beliefs and local evidence.
- $\hat{\pi}_i^t$ is the planner's estimate of its current privilege or capability level at this internal node.
- q_i^t stores local statistics such as scan count, sensing count, exploit count, query count, visit count, last-visited step, and other search-relevant counters.

The node state is updated by observations. For example, **scan** may add new fingerprint evidence and neighbour handles; **traffic_sense** may update role beliefs through contact patterns; **exploit** may update privilege and credential-related fields; **confirm_identity** may set ζ_i^t and trigger merging of two internal nodes; and **query_db** may update query counters and database-role beliefs. The detailed action-specific updates are defined in Section 4.6.

4.3.2. Internal Edge Representation

Each internal edge $\hat{e}_{ij} \in \hat{E}_t$ is associated with an edge state

$$y_{ij}^t = (\rho_{ij}^t, \hat{c}_{ij}^t, \alpha_{ij}^t, \beta_{ij}^t, m_{ij}^t), \quad (4.8)$$

where $\rho_{ij}^t \in [0, 1]$ is the confidence that a traversable relation exists, $\hat{c}_{ij}^t$ is the estimated movement cost, $(\alpha_{ij}^t, \beta_{ij}^t)$ are Beta posterior parameters for movement reliability, and m_{ij}^t stores local edge statistics such as movement attempts, successes, failures, and recency.

The current estimated movement-success probability is

$$\hat{p}_{ij}^{\text{move}}(t) = \frac{\alpha_{ij}^t}{\alpha_{ij}^t + \beta_{ij}^t}. \quad (4.9)$$

After a realised movement attempt, the edge posterior and cost estimate are updated. If a move succeeds, α_{ij} is increased and the cost estimate is adjusted toward the realised cost. If a move fails or the target is found not to be adjacent, β_{ij} or the edge-confidence field ρ_{ij} is adjusted accordingly. These updates allow the planner to learn local movement reliability online.

4. Problem Definition and Environment Modelling

4.4. Observation Model and Identity Association

The observation returned after executing action a_t is denoted by o_{t+1} . Observations are heterogeneous: different actions produce different kinds of evidence. At an abstract level, an observation may contain

$$o_{t+1} = \left(N_{t+1}^{\text{obs}}, E_{t+1}^{\text{obs}}, T_{t+1}, C_{t+1}, X_{t+1}, Q_{t+1}, M_{t+1} \right), \quad (4.10)$$

where:

- N_{t+1}^{obs} contains newly observed node-level evidence. When a more detailed decomposition is needed, this field can be written as

$$N_{t+1}^{\text{obs}} = (H_{t+1}, F_{t+1}), \quad (4.11)$$

where H_{t+1} contains temporary handles or local identifiers, and F_{t+1} contains partial fingerprints such as observed ports, services, certificate fragments, addresses, or other local attributes.

- E_{t+1}^{obs} contains observed or hypothesised local relations.
- T_{t+1} contains traffic or contact statistics.
- C_{t+1} contains stable identity information revealed by `confirm_identity`.
- X_{t+1} contains exploit feedback, privilege feedback, or credential feedback.
- Q_{t+1} contains query feedback.
- M_{t+1} contains movement feedback, such as success, failure, non-adjacency, realised cost, and reached handle.

Not every field is present in every observation. For example, `scan` mainly returns node and edge evidence; `traffic_sense` mainly returns contact statistics; `move` mainly returns movement feedback; and `query_db` mainly returns query feedback.

A central complication is identity association. A newly observed handle or fingerprint does not necessarily correspond to a new real object. It may refer to an already known object under a different temporary handle. The planner therefore applies an association update:

$$\mathcal{M}_{t+1} = \text{Assoc}(\hat{G}_t, o_{t+1}), \quad (4.12)$$

where $\mathcal{M}_{t+1}$ is a set of create, associate, and merge decisions. This is related to the general problem of entity resolution or record linkage, in which multiple observed descriptions may refer to the same underlying entity (Fellegi and Sunter, 1969). In this thesis, the association process is used inside an online planning loop rather than as a standalone data-cleaning task.

For each newly observed entity description x , the planner chooses one of three abstract outcomes:

$$x \rightarrow \text{CREATE}, \quad x \rightarrow \text{ASSOCIATE}(\hat{v}_i), \quad x \rightarrow \text{MERGE}(\hat{v}_i, \hat{v}_j). \quad (4.13)$$

CREATE introduces a new internal node. ASSOCIATE attaches new evidence to an existing internal node. MERGE combines two internal nodes when stronger evidence indicates that they correspond to the same stable entity.

Temporary handles alone are not treated as stable identity evidence. Stable matching requires stronger signals such as a confirmed address, certificate, or another persistent identifier. Partial fingerprints, such as ports and services, can support similarity-based association, but they are treated as uncertain evidence. If two nodes are merged, their fingerprint evidence, role beliefs, edge incidences, cost estimates, reliability statistics, and visitation statistics are combined according to the knowledge-update rules described in the knowledge-representation chapter.

4.5. Action Space

The primitive action set consists of information-gathering actions, movement actions, capability-modifying actions, identity-confirmation actions, and the terminal query action:

$$\mathcal{A}_{\text{base}} = \{\text{scan}, \text{traffic_sense}, \text{exploit}, \text{confirm_identity}, \text{query_db}\}. \quad (4.14)$$

$$\mathcal{A}(I_t) = \mathcal{A}_{\text{base}} \cup \{\text{move}(\hat{u}) : \hat{u} \in \hat{V}_t^{\text{cand}}\}. \quad (4.15)$$

Here $\hat{V}_t^{\text{cand}} \subseteq \hat{V}_t$ is the set of currently considered internal movement targets. The argument $\hat{u}$ of **move** is an internal node, not a true node identifier. The simulator resolves it through the hidden correspondence between internal observations and true objects.

Each action has a precondition, a true environment effect, an observation, and an internal update. These are defined below. Let B_t denote the remaining budget. An action is executable only if B_t is sufficient to pay at least its minimal action cost. If an action is attempted and its realised cost is c_t , then

$$B_{t+1} = B_t - c_t. \quad (4.16)$$

If $B_{t+1} < 0$, the run is considered to have exhausted its budget.

4.6. Action Semantics

This section gives the action-level semantics required for the formal problem definition. For each action, the precondition specifies when the action may be selected, the true effect specifies how the hidden environment state may change, the observation specifies what information is returned to the planner, and the internal update specifies how the knowledge state is revised.

4.6.1. scan

Precondition. **scan** is executable when the planner has a current internal location $\hat{\ell}_t$ and sufficient remaining budget:

$$\text{Pre}(\text{scan}, I_t) \equiv (\hat{\ell}_t \in \hat{V}_t) \wedge (B_t \geq c_{\min}^{\text{scan}}). \quad (4.17)$$

4. Problem Definition and Environment Modelling

True effect. `scan` does not change the true location, privilege state, credential state, or success flag:

$$\ell_{t+1}^* = \ell_t^*, \quad \pi_{t+1}^* = \pi_t^*, \quad \kappa_{t+1}^* = \kappa_t^*, \quad h_{t+1}^* = h_t^*. \quad (4.18)$$

Observation. The observation reveals a partial local view around the true current location ℓ_t^* . It may include temporary handles for neighbouring objects, partial fingerprints of those objects, and local relation hints:

$$o_{t+1}^{\text{scan}} = \left(N_{t+1}^{\text{obs}}, E_{t+1}^{\text{obs}}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \right). \quad (4.19)$$

The scan observation does not reveal the full graph and does not directly reveal stable identities.

Internal update. The planner associates or creates internal nodes for observed handles and fingerprints, adds or updates internal edges for observed local relations, updates scan counters, and updates role beliefs using the newly observed fingerprints. Formally:

$$\hat{G}_{t+1} = U_{\text{scan}}(\hat{G}_t, o_{t+1}^{\text{scan}}). \quad (4.20)$$

The current internal location remains unchanged unless identity association or merging changes the representative internal node.

4.6.2. traffic_sense

Precondition. `traffic_sense` is executable when the planner has a current internal location and sufficient remaining budget:

$$\text{Pre}(\text{traffic_sense}, I_t) \equiv (\hat{\ell}_t \in \hat{V}_t) \wedge (B_t \geq c_{\min}^{\text{sense}}). \quad (4.21)$$

True effect. `traffic_sense` does not change the true execution state:

$$\ell_{t+1}^* = \ell_t^*, \quad \pi_{t+1}^* = \pi_t^*, \quad \kappa_{t+1}^* = \kappa_t^*, \quad h_{t+1}^* = h_t^*. \quad (4.22)$$

Observation. The observation returns traffic or contact statistics around the current true location:

$$o_{t+1}^{\text{sense}} = \left(N_{t+1}^{\text{obs}}, E_{t+1}^{\text{obs}}, T_{t+1}, \emptyset, \emptyset, \emptyset, \emptyset \right). \quad (4.23)$$

The contact statistics may provide weak structural evidence about neighbouring roles or likely downstream connectivity, but they do not reveal the full topology or stable identities.

Internal update. The planner updates contact statistics, role beliefs, and relation confidence. For example, repeated contact patterns may increase belief that a node is credential-relevant, database-adjacent, or structurally central. Abstractly:

$$\hat{G}_{t+1} = U_{\text{sense}}(\hat{G}_t, o_{t+1}^{\text{sense}}). \quad (4.24)$$

4.6.3. move()

Precondition. A movement action is parameterised by an internal target node $\hat{u} \in \hat{V}_t^{\text{cand}}$. It is executable when $\hat{u}$ is a current movement candidate and sufficient budget remains:

$$\text{Pre}(\text{move}(\hat{u}), I_t) \equiv (\hat{u} \in \hat{V}_t^{\text{cand}}) \wedge (B_t \geq c_{\min}^{\text{move}}). \quad (4.25)$$

True effect. Let the true current location be $\ell_t^* = v$. The internal target $\hat{u}$ is resolved by the simulator to a true candidate object w when such a correspondence is available. There are three cases.

1. If no valid true target can be resolved, or if $(v, w) \notin E^*$, the movement attempt is treated as invalid or non-adjacent:

$$\ell_{t+1}^* = v. \quad (4.26)$$

2. If $(v, w) \in E^*$, the movement succeeds with probability $p_{vw}^{\text{move},*}$:

$$\Pr(\ell_{t+1}^* = w \mid \ell_t^* = v, \text{move}(\hat{u})) = p_{vw}^{\text{move},*}. \quad (4.27)$$

3. If the movement fails, the true location remains unchanged:

$$\ell_{t+1}^* = v. \quad (4.28)$$

Movement does not directly change privilege, credentials, or task success:

$$\pi_{t+1}^* = \pi_t^*, \quad \kappa_{t+1}^* = \kappa_t^*, \quad h_{t+1}^* = h_t^*. \quad (4.29)$$

Observation. The movement observation reports whether the attempt succeeded, failed, or was non-adjacent, together with the realised movement cost and, if successful, the reached object's handle or fingerprint:

$$o_{t+1}^{\text{move}} = (N_{t+1}^{\text{obs}}, E_{t+1}^{\text{obs}}, \emptyset, \emptyset, \emptyset, \emptyset, M_{t+1}). \quad (4.30)$$

Internal update. If the move succeeds, the internal current location is updated to the associated internal node:

$$\hat{\ell}_{t+1} = \text{AssocLoc}(\hat{G}_t, M_{t+1}). \quad (4.31)$$

The corresponding internal edge reliability is updated by increasing the success posterior parameter and adjusting the movement-cost estimate. If the move fails, the failure posterior parameter is increased. If the observation indicates non-adjacency, the edge-confidence estimate ρ_{ij} is decreased. Abstractly:

$$y_{ij}^{t+1} = U_{\text{move}}(y_{ij}^t, M_{t+1}, c_t). \quad (4.32)$$

4.6.4. exploit

Precondition. `exploit` is executable when the planner has a current internal location and sufficient budget:

$$\text{Pre}(\text{exploit}, I_t) \equiv (\hat{\ell}_t \in \hat{V}_t) \wedge (B_t \geq c_{\min}^{\text{exploit}}). \quad (4.33)$$

4. Problem Definition and Environment Modelling

True effect. Let $\ell_t^* = v$. The exploit attempt succeeds with a probability determined by the true node attributes, security level, and current capability state:

$$\Pr(\text{exploit_success} = 1 \mid s_t^*) = p_{\text{exploit}}^*(v, \pi_t^*, \kappa_t^*). \quad (4.34)$$

On success, the true privilege state may increase:

$$\pi_{t+1}^* \geq \pi_t^*. \quad (4.35)$$

If the exploited node is credential-relevant, the credential flag may become true:

$$\kappa_{t+1}^* = 1. \quad (4.36)$$

On failure, privilege and credential state may remain unchanged. Exploit does not directly complete the task:

$$h_{t+1}^* = h_t^*. \quad (4.37)$$

Observation. The exploit observation reports whether privilege increased and whether credentials were obtained:

$$o_{t+1}^{\text{exploit}} = (\emptyset, \emptyset, \emptyset, \emptyset, X_{t+1}, \emptyset, \emptyset). \quad (4.38)$$

Internal update. The planner updates the privilege estimate for the current internal node, the global credential flag, exploit counters, and role beliefs. For example, obtaining credentials provides evidence that the current node is credential-relevant:

$$I_{t+1} = U_{\text{exploit}}(I_t, X_{t+1}, c_t). \quad (4.39)$$

4.6.5. confirm_identity

Precondition. `confirm_identity` is executable when the planner has a current internal location whose stable identity is unknown or uncertain, and sufficient budget remains:

$$\text{Pre}(\text{confirm_identity}, I_t) \equiv (\hat{\ell}_t \in \hat{V}_t) \wedge (\zeta_{\ell_t}^t = \perp \text{ or identity is uncertain}) \wedge (B_t \geq c_{\min}^{\text{confirm}}). \quad (4.40)$$

True effect. Identity confirmation does not change the true location, privilege state, credential state, or task-success flag:

$$\ell_{t+1}^* = \ell_t^*, \quad \pi_{t+1}^* = \pi_t^*, \quad \kappa_{t+1}^* = \kappa_t^*, \quad h_{t+1}^* = h_t^*. \quad (4.41)$$

Observation. The observation reveals stable identity evidence for the current true object, such as an address, certificate, or persistent identifier:

$$o_{t+1}^{\text{confirm}} = (\emptyset, \emptyset, \emptyset, C_{t+1}, \emptyset, \emptyset, \emptyset). \quad (4.42)$$

Internal update. The planner assigns the stable identity to the current internal node. If another internal node already has the same stable identity, the two nodes are merged:

$$\hat{G}_{t+1} = U_{\text{merge}}(\hat{G}_t, C_{t+1}). \quad (4.43)$$

Merging combines handles, fingerprint evidence, role beliefs, privilege estimates, incident edges, movement statistics, and visitation statistics. If the current internal location is merged into another representative node, $\hat{\ell}_{t+1}$ is updated to the representative.

4.6.6. query_db

Precondition. `query_db` is executable when the planner has a current internal location and sufficient budget:

$$\text{Pre}(\text{query_db}, I_t) \equiv (\hat{\ell}_t \in \hat{V}_t) \wedge (B_t \geq c_{\min}^{\text{query}}). \quad (4.44)$$

The action may be selected even when the planner is uncertain whether the current node is a true database node. This is important because the planner does not directly observe the true role.

True effect. Let $\ell_t^* = v$. The query succeeds only if the current true node is a database node and the required capability conditions are satisfied:

$$h_{t+1}^* = 1 \quad \text{iff} \quad r_v^* = \text{DB} \wedge \text{Capable}(\pi_t^*, \kappa_t^*) = 1 \wedge \text{QuerySuccess} = 1. \quad (4.45)$$

If any of these conditions fail, then $h_{t+1}^* = h_t^*$. The term `QuerySuccess` allows the simulator to include stochastic query success if desired; in a deterministic query setting it is always true when the role and capability conditions are met.

Observation. The query observation reports success or failure:

$$o_{t+1}^{\text{query}} = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, Q_{t+1}, \emptyset). \quad (4.46)$$

Internal update. If the query succeeds, the internal success flag becomes true:

$$h_{t+1} = 1. \quad (4.47)$$

If the query fails, the planner updates query counters and may revise role beliefs or capability-related beliefs. For example, a failed query at a node believed to be database-related may reduce the database-role belief, unless the failure is better explained by missing privilege or credentials:

$$I_{t+1} = U_{\text{query}}(I_t, Q_{t+1}, c_t). \quad (4.48)$$

4. Problem Definition and Environment Modelling

4.7. Internal State Update

Given the current internal state I_t , action a_t , observation o_{t+1} , and realised cost c_t , the general internal update is

$$I_{t+1} = U(I_t, a_t, o_{t+1}, c_t). \quad (4.49)$$

The update operator performs four kinds of operation:

1. **Budget update:** subtract the realised action cost from the remaining budget.
2. **Graph update:** create, associate, or merge internal nodes; add or revise edges; update node and edge statistics.
3. **Belief update:** revise role beliefs, edge confidence, movement reliability, privilege estimates, and credential state.
4. **Task update:** set the internal success flag if terminal query success is observed.

This update should not be understood as a single standard Bayesian belief update over a fixed hidden state space. Some subcomponents may be Bayesian or Bayesian-style. For example, movement reliability can be updated from observed movement successes and failures in a posterior-like manner. However, the full update operator is broader than belief revision alone: it also performs object creation, identity association, node merging, edge insertion, edge revision, budget accounting, and task-status updates.

A key difference from a standard fixed-state belief update is that the internal object set and edge set may themselves change:

$$\hat{V}_{t+1} \neq \hat{V}_t \quad \text{and} \quad \hat{E}_{t+1} \neq \hat{E}_t. \quad (4.50)$$

New objects may be added after `scan` or `traffic_sense`; edges may be added or revised after `scan`, `traffic_sense`, or `move`; and nodes may be merged after `confirm_identity`. In addition, the role-belief mechanism described in Chapter 5 is not implemented as a strict Bayesian posterior update from a prior distribution. It is better understood as an evidence-based recalculation and normalisation procedure over recorded local cues, such as observed services, traffic patterns, action outcomes, and structural context.

Thus, the operator U combines several update types: deterministic accounting updates, graph-structural updates, Bayesian-style statistical updates for some quantities, and heuristic evidence-aggregation updates for others. The present chapter defines the problem-level semantics of this update, while Chapter 5 describes the knowledge-representation and belief-update mechanisms used in the implementation.

4.8. Goal Condition, Budget Constraint, and Evaluation Criteria

Let an execution trajectory be

$$\tau = (I_0, a_0, o_1, c_0, I_1, a_1, o_2, c_1, \dots, I_T). \quad (4.51)$$

4.9. Internal Progress and Local Planning Criteria

The cumulative realised cost is

$$C(\tau) = \sum_{t=0}^{T-1} c_t. \quad (4.52)$$

A trajectory is budget-feasible if

$$C(\tau) \leq B_0. \quad (4.53)$$

The external task-success condition is

$$\text{Succ}(\tau) = \begin{cases} 1, & \text{if } h_t^* = 1 \text{ for some } t \leq T, \\ 0, & \text{otherwise.} \end{cases} \quad (4.54)$$

Thus, success is defined only by successful execution of `query_db` under the true role and capability conditions. Discovering a database-like node, reaching a high-belief node, or obtaining credentials is not itself terminal success.

The primary external objective is to maximise the probability of success under the budget constraint:

$$\max_{\pi} \Pr_{\pi}(\text{Succ}(\tau) = 1) \quad \text{subject to} \quad C(\tau) \leq B_0. \quad (4.55)$$

Cost is also used as an evaluation metric, especially through success step, remaining budget, and runtime. However, the task is not simply a shortest-path problem, because the graph is not fully known and many useful actions are information-gathering or capability-building actions.

4.9. Internal Progress and Local Planning Criteria

Terminal success is sparse: most actions do not immediately make `query_db` succeed. To guide local search before terminal success becomes visible, the planner uses an internal progress signal. This signal is not part of the external environment objective. It is an internal proxy used to evaluate whether a state or local option improves the prospect of future success.

Let

$$g(I_t) \in [0, 1] \quad (4.56)$$

denote the internal progress score. Conceptually, $g(I_t)$ may depend on four groups of evidence:

1. **Database potential:** whether discovered nodes have high database-related belief and are reachable or nearly reachable.
2. **Exploration progress:** whether new nodes or edges have been discovered.
3. **Uncertainty reduction:** whether role beliefs or identity ambiguity have become more informative.
4. **Capability progress:** whether privilege or credentials have improved.

4. Problem Definition and Environment Modelling

The exact formula for $g(I_t)$ and its relation to frontier scoring and global aggregation are given in Chapter 7. In this chapter, the important distinction is that terminal success and progress are different quantities. Terminal success is externally defined by `query_db`; progress is an internal shaping and diagnostic signal used to make local search less sparse.

A local option generated by the planner is therefore evaluated by three predicted quantities:

$$\hat{p}(o), \quad \hat{g}(o), \quad \hat{c}(o), \quad (4.57)$$

where $\hat{p}(o)$ is predicted terminal-success probability, $\hat{g}(o)$ is predicted progress, and $\hat{c}(o)$ is predicted cost. These quantities are used later for approximate non-dominated local option generation and global action aggregation. Their exact estimation procedure is defined in Chapter 7. This avoids prematurely compressing all local decisions into a single scalar value, which is important in multi-objective sequential decision problems (Roijers et al., 2013).

4.10. Formal Problem Statement

The problem addressed in this thesis can now be stated as follows.

Given:

- a hidden true environment graph G^* with node attributes, edge attributes, stochastic action outcomes, and a true current execution state s_t^* ;
- an initial internal knowledge state I_0 containing only partial information;
- a finite budget B_0 ;
- an action set $\mathcal{A}(I_t)$ containing information-gathering, movement, capability, identity-confirmation, and terminal query actions;
- heterogeneous observations generated by action execution; and
- an internal update operator U that can revise beliefs, create objects, associate evidence, and merge identities;

find an online policy

$$\pi : I_t \mapsto a_t \quad (4.58)$$

that selects actions from the current internal state so as to maximise the probability of terminal success before budget exhaustion:

$$\max_{\pi} \Pr_{\pi} \left(\exists t \leq T : h_t^* = 1 \wedge \sum_{k=0}^{t-1} c_k \leq B_0 \right). \quad (4.59)$$

The policy is online because it chooses each action after observing the current internal state rather than computing a complete plan over the hidden true graph. It is partially observable because s_t^* and G^* are not directly available. It is object-discovery-based because $\hat{V}_t$ and $\hat{E}_t$ can change during execution. It is identity-aware because temporary observations may later be associated or merged through stable

identity evidence. Finally, it is budget-constrained because every action consumes cost and the terminal objective must be achieved before the budget is exhausted.

4.11. Chapter Summary

This chapter defined the planning problem studied in the thesis. The key modelling distinction is between the hidden real environment state and the planner’s internal knowledge state. The internal state is a graph-structured, object-centric belief representation that can grow, be revised, and be reorganised through observations. The chapter also defined the primitive action space and, for each action, specified its preconditions, true effects, observations, and internal updates. Finally, it defined the external success condition, the budget constraint, and the role of internal progress as a proxy signal for local search. The following chapters build on this formalisation by defining the knowledge-update mechanisms, local subnet construction, approximate non-dominated option generation, and reusable policy-library abstraction.

4. Problem Definition and Environment Modelling

Table 4.1.: Correspondence between network-penetration concepts and the abstract planning model.

Network-penetration concept	Abstract model component	Interpretation in this thesis
Host or network asset	True node $v \in V^*$	A real environment object such as an entry host, intermediate host, credential-relevant host, or database-related host.
Observed host handle	Internal node $\hat{v} \in \hat{V}_t$	A planner-side object created from partial observations; it may later be associated or merged with another internal node.
IP address, certificate, stable identifier	Stable identity id_v^* or ζ_i^t	Persistent identity evidence revealed only by identity-confirmation observations.
Open ports, services, fingerprints	Fingerprint attributes ϕ_v^* and F_i^t	Local evidence used to update role beliefs and identity associations.
Network reachability	True edge $(v, w) \in E^*$	A traversable relation indicating that movement or lateral access from one object to another is possible.
Observed or hypothesised reachability	Internal edge $\hat{e}_{ij} \in \hat{E}_t$	The planner's current belief that two internal objects are connected or reachable.
Firewall, routing, segmentation, access uncertainty	Edge reliability and confidence	Captured abstractly by movement-success probability, edge confidence, and movement feedback.
Vulnerability or exploitable service	Security attribute sec_v^* and exploit	An abstract capability-improving opportunity; not modelled as a concrete CVE.
Successful exploitation	Privilege or credential update	May increase privilege π_t^* or set the credential flag $\kappa_t^* = 1$.
Credential discovery	Credential flag κ_t^* or $\hat{\kappa}_t$	Represents obtaining information required for later database access.
Reconnaissance or service scanning	scan	Reveals partial local topology, handles, and fingerprint evidence.
Traffic or contact analysis	traffic_sense	Provides weak structural evidence about neighbouring roles and downstream connectivity.
Lateral movement	move ($\hat{u}$)	Attempts to move from the current object to a selected internal target through a believed reachable relation.
Database server or target asset	DB role $r_v^* = \text{DB}$	A true target node at which successful terminal verification may be possible.
Target verification or database access	query_db	The terminal action; it succeeds only at a true database node when capability conditions are satisfied.
Operational cost or time budget	Budget B_t and realised action cost c_t	Limits how much information gathering, movement, exploitation, and querying can be attempted.

Internal State Representation: Knowledge Graph, Identity Association, and Role Beliefs

The previous chapter defined the formal problem setting, including the hidden true environment state, the internal knowledge state, the heterogeneous observation structure, and the primitive action semantics. In particular, Section 4.6 specified the preconditions, true effects, observations, and internal updates of the primitive actions. This chapter therefore does not repeat the full action model. Instead, it focuses on the structure and maintenance of the planner’s internal state.

Because the relevant objects, their attributes, and their relations are not fully visible at the outset, the planner cannot act directly on the hidden true environment graph. It must instead maintain an evolving internal approximation that is revised after each observation. This representation is object-centric rather than based on a fixed propositional state vector, and is therefore related to object-oriented and relational modelling approaches in decision making (Diuk et al., 2008; Boutilier et al., 2001). At the same time, it differs from standard finite-state belief-state models under partial observability, because the internal object set itself can grow, accumulate ambiguity, and later be reorganised through object merging (Kaelbling et al., 1998).

The internal representation used in this thesis is called a knowledge graph in a restricted planning sense. In general, knowledge graphs represent entities and relations in graph-structured form (Hogan et al., 2021). Here, the knowledge graph is not a general-purpose semantic knowledge base. It is the planner’s evolving graph-structured belief state. It stores discovered objects, temporary handles, confirmed stable identities, fingerprints, role beliefs, edge confidence, movement-cost estimates, movement-reliability estimates, and local statistics used by search and experience reuse.

The purpose of this chapter is to explain how this internal representation is organised and revised. Later chapters use this representation for local subnet construction, frontier and progress scoring, approximate non-dominated local option generation, and generalized policy-library retrieval.

5.1. Knowledge Graph as an Evolving Belief State

As defined in Chapter 4, the planner acts on an internal knowledge state

$$I_t = \left(\hat{G}_t, \hat{\ell}_t, \hat{\pi}_t, \hat{\kappa}_t, B_t, h_t \right), \quad (5.1)$$

5. Internal State Representation: Knowledge Graph, Identity Association, and Role Beliefs

where $\hat{G}_t$ is the internal knowledge graph, $\hat{\ell}_t$ is the current internal location, $\hat{\pi}_t$ is the planner's privilege estimate, $\hat{\kappa}_t$ is the internal credential flag, B_t is the remaining budget, and h_t is the internal success flag. This chapter focuses on the first component:

$$\hat{G}_t = (\hat{V}_t, \hat{E}_t). \quad (5.2)$$

The set $\hat{V}_t$ contains internal object hypotheses, and $\hat{E}_t$ contains internal relation hypotheses. An internal node is not assumed to be identical to a true environment node. Instead, it is the planner's current representation of an object that has been observed, inferred, or associated through previous interaction. In the ideal case, each internal node corresponds to exactly one true object. Under identity uncertainty, however, this correspondence can temporarily be imperfect:

- a true object may be represented by more than one internal node before identity confirmation;
- one internal node may accumulate evidence from several observations;
- two internal nodes may later be merged if stable identity evidence shows that they refer to the same true object.

For exposition, let

$$\mu_t : \hat{V}_t \rightarrow 2^{V^*} \quad (5.3)$$

denote the hidden correspondence between internal objects and true objects. The planner does not observe μ_t directly. It only maintains evidence that helps decide whether new observations should create new internal nodes, be associated with existing nodes, or trigger merging. Therefore, $\hat{G}_t$ should be understood as a revisable object-level belief structure, not as a copy of the true environment graph.

5.2. Internal Node Records

Each internal node $k \in \hat{V}_t$ is associated with a node record

$$x_k^t = (H_k^t, \zeta_k^t, F_k^t, b_k^t, d_k^t, s_k^t, \hat{\pi}_k^t, q_k^t, a_k^t). \quad (5.4)$$

The components are defined as follows.

- H_k^t is the set of temporary handles that have been associated with node k . A handle is useful for local reference, but it is not treated as stable identity evidence.
- ζ_k^t is the stable identity of the node if it has been confirmed; otherwise $\zeta_k^t = \perp$. Stable identity evidence may include persistent address- or certificate-level information.
- F_k^t is the accumulated fingerprint evidence, such as observed ports, services, certificate fragments, addresses, or other local attributes. Fingerprints may be partial and noisy.
- b_k^t is a role-belief distribution over a finite role set $\mathcal{R}$:

$$b_k^t(r) = \Pr(R_k = r \mid I_t), \quad r \in \mathcal{R}. \quad (5.5)$$

5.3. Internal Edge Records

- $d_k^t \in [0, 1]$ is a derived database-related or data-relevance belief. It is not an independent true label, but a scalar summary derived mainly from the database-related components of the role belief and task feedback.
- $s_k^t \in [0, 1]$ is a security or difficulty estimate associated with the node.
- $\hat{\pi}_k^t$ records the planner's current estimate of privilege or local capability at this node.
- q_k^t stores local interaction statistics, such as scan count, traffic-sensing count, exploit count, query count, visit count, last-visited step, and recency information.
- a_k^t stores identity-ambiguity and database-test flags, such as whether the node has been tested by `query_db`, whether it has been confirmed as a database node, and whether it is known not to be a database node under diagnostic feedback.

The important point is not the implementation name of each field, but its role in planning. These quantities allow the planner to reason about object identity, semantic role, local capability, uncertainty, and search history. They also provide the information used later for subnet construction, progress scoring, move-target selection, and policy-library retrieval.

5.3. Internal Edge Records

Each internal edge $(i, j) \in \hat{E}_t$ is associated with an edge record

$$y_{ij}^t = (\rho_{ij}^t, \hat{c}_{ij}^t, \alpha_{ij}^t, \beta_{ij}^t, m_{ij}^t, \xi_{ij}^t). \quad (5.6)$$

The components are defined as follows.

- $\rho_{ij}^t \in [0, 1]$ is the confidence that a traversable relation between i and j exists.
- $\hat{c}_{ij}^t$ is the estimated movement cost.
- $(\alpha_{ij}^t, \beta_{ij}^t)$ are Beta posterior parameters for movement-success probability.
- m_{ij}^t stores movement statistics, such as the number of attempts, successes, failures, and the most recent update time.
- ξ_{ij}^t stores blocking, non-adjacency, or topology-negation information.

The current estimated movement-success probability is

$$\hat{p}_{ij}^{\text{move}}(t) = \frac{\alpha_{ij}^t}{\alpha_{ij}^t + \beta_{ij}^t}. \quad (5.7)$$

The representation separates perceptual evidence from execution evidence. If an edge first enters the internal graph through sensing or scanning, the planner may initialise ρ_{ij}^t and $\hat{c}_{ij}^t$ from observation. However, successful traversability is not fully trusted until actual movement attempts provide evidence. After a successful `move`, the success posterior parameter is increased and the cost estimate is adjusted

5. Internal State Representation: Knowledge Graph, Identity Association, and Role Beliefs

Table 5.1.: Main internal knowledge-graph fields affected by each primitive action.

Action	Observation type	Main internal update
scan	Local handles, fingerprints, local relations	Create or associate nodes; add or revise edges; update fingerprints and scan counts.
traffic_sense	Contact statistics and weak structural evidence	Update contact statistics, role beliefs, edge confidence, and sensing counts.
move	Success, failure, non-adjacency, realised cost, reached handle	Update location, edge reliability, movement cost, visit counts, and blocked-edge evidence.
exploit	Privilege and credential feedback	Update privilege estimate, credential flag, exploit count, and role beliefs.
confirm_identity	Stable identity evidence	Set stable identity, associate confirmed evidence, and merge duplicate internal nodes.
query_db	Query success or failure feedback	Update success flag, query count, database belief, and database-test flags.

toward the realised cost. After a failed movement attempt, the failure posterior parameter is increased. If the observation indicates non-adjacency, the edge-confidence estimate may be reduced or the edge may be marked as blocked.

Thus, an internal edge is not merely a discovered connection. It is a belief-bearing relation whose existence, cost, and reliability can change through interaction. This distinction is important under partial observability, because sensing evidence may suggest a relation without guaranteeing that it is executable.

5.4. Observation-Induced Knowledge-Graph Updates

The action-level semantics were defined in Section 4.6. From the perspective of internal-state representation, the important question is how observations affect the knowledge graph. At a high level, the graph update can be written as

$$\hat{G}_{t+1} = U_{\hat{G}}(\hat{G}_t, a_t, o_{t+1}), \quad (5.8)$$

where a_t is the executed action and o_{t+1} is the returned observation.

Table 5.1 summarises the main internal fields affected by each primitive action. This table is not intended to repeat the full preconditions and effects of the actions. Instead, it shows how the observations defined in the previous chapter are used to revise the internal representation.

The resulting update is not monotonic in the simple set-inclusion sense. Although new scans and sensing actions can increase $\hat{V}_t$ and $\hat{E}_t$, identity confirmation can merge internal nodes, and movement feedback can weaken or block suspected edges. Therefore, the internal graph should be understood as a revisable belief structure rather than a monotonically growing discovered subgraph.

5.5. Identity Association

A key difference between the present problem and static graph search is that stable object identities are not directly exposed at early stages. The identity-observation constraint was stated in Chapter 4: actions such as `scan`, `traffic_sense`, `move`, `exploit`, and `query_db` may reveal temporary handles, local fingerprints, movement feedback, or task feedback, but they do not by themselves reveal stable object identities. Stable identity information is revealed only by `confirm_identity`.

Consequently, internal state update cannot be treated as a standard belief update over a fixed object set. It must also include object creation, object association, and object merging. This problem is related to entity resolution or record linkage, where multiple observed descriptions must be judged as referring to the same underlying entity or to different entities (Fellegi and Sunter, 1969). In this thesis, however, identity association is embedded inside an online planning loop: association errors can directly affect later search, movement, and experience reuse.

Let f be a newly observed fingerprint or entity description. The planner applies an association operator

$$A(f, \hat{G}_t) \rightarrow (k, \lambda), \quad (5.9)$$

where k is an internal node identifier and

$$\lambda \in \{\text{MATCHEDSTABLE}, \text{MATCHED}, \text{CREATED}, \text{CREATEDAMBIGUOUS}\} \quad (5.10)$$

is an association label.

The labels have the following meanings:

- **MATCHEDSTABLE**: the observation contains stable identity evidence that matches an existing internal node.
- **MATCHED**: the observation is sufficiently similar to an existing internal node, even though stable identity evidence is not available.
- **CREATED**: the observation is sufficiently dissimilar from all existing internal nodes, so a new internal node is created.
- **CREATEDAMBIGUOUS**: the evidence is inconclusive, so a new internal node is created while preserving ambiguity information rather than forcing an early merge.

5.5.1. Stable-Identity Matching

The association procedure first checks for stable identity evidence. If the observation contains a stable identifier such as a confirmed address or certificate, and an existing internal node has the same stable identifier, then the observation is associated with that node:

$$A(f, \hat{G}_t) = (k, \text{MATCHEDSTABLE}) \quad \text{if} \quad \zeta(f) = \zeta_k^t \neq \perp. \quad (5.11)$$

Stable-identity matching has priority over similarity matching. This is because stable identifiers are intended to represent persistent identity evidence, whereas ports, services, and temporary handles may be incomplete or ambiguous.

5. Internal State Representation: Knowledge Graph, Identity Association, and Role Beliefs

5.5.2. Similarity-Based Association

If stable identity evidence is not available, the planner uses similarity over partial fingerprints. Temporary handles are not treated as reliable identity evidence. Instead, similarity is computed from more persistent observable attributes such as ports and services.

Let $P(f)$ be the set of ports in fingerprint f , and let $S(f)$ be the set of services. The Jaccard coefficient between two finite sets is

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \quad (5.12)$$

with $J(A, B) = 0$ when both sets are empty. Jaccard similarity is a standard set-overlap measure (Jaccard, 1901). A fingerprint similarity score can then be written as

$$\text{sim}(f, F_k^t) = w_P J(P(f), P(F_k^t)) + w_S J(S(f), S(F_k^t)), \quad (5.13)$$

where $w_P, w_S \geq 0$ and $w_P + w_S = 1$.

Let

$$k^* = \arg \max_{k \in \hat{V}_t} \text{sim}(f, F_k^t). \quad (5.14)$$

Given high and low thresholds τ_{hi} and τ_{lo} , the association decision is

$$A(f, \hat{G}_t) = \begin{cases} (k^*, \text{MATCHED}), & \text{if } \text{sim}(f, F_{k^*}^t) \geq \tau_{\text{hi}}, \\ (k_{\text{new}}, \text{CREATED}), & \text{if } \text{sim}(f, F_{k^*}^t) \leq \tau_{\text{lo}}, \\ (k_{\text{new}}, \text{CREATEDAMBIGUOUS}), & \text{otherwise.} \end{cases} \quad (5.15)$$

The ambiguous case is important. Rather than forcing a merge when evidence is weak, the planner creates a new internal object and records the ambiguity. This reduces the risk of irreversible early merging errors. Later, `confirm_identity` may provide stable evidence that resolves the ambiguity.

5.6. Object Merging

Object merging occurs when stable identity evidence indicates that two internal nodes represent the same true object. Suppose that two internal nodes i and j have the same confirmed stable identity:

$$\zeta_i^t = \zeta_j^t \neq \perp. \quad (5.16)$$

The planner then merges them into a representative node:

$$\hat{v}_m = \text{Merge}(\hat{v}_i, \hat{v}_j). \quad (5.17)$$

When two nodes are merged, their internal states are combined as follows:

- temporary handles are combined by union:

$$H_m = H_i \cup H_j; \quad (5.18)$$

5.7. Role Beliefs

- stable identity is preserved:

$$\zeta_m = \zeta_i = \zeta_j; \quad (5.19)$$

- fingerprint evidence is merged by accumulating observed attributes:

$$F_m = F_i \cup F_j; \quad (5.20)$$

- role beliefs are combined and renormalised, for example by a weighted average:

$$b_m(r) = \frac{\omega_i b_i(r) + \omega_j b_j(r)}{\sum_{r' \in \mathcal{R}} (\omega_i b_i(r') + \omega_j b_j(r'))}; \quad (5.21)$$

- database-related belief, security estimate, privilege estimate, and local counters are combined according to their interpretation;
- incident edges are redirected to the representative node;
- duplicate edges are consolidated by combining edge confidence, cost estimates, movement statistics, and movement-reliability evidence.

For duplicate edges, if two internal edges represent the same relation after merging, their reliability evidence can be combined by adding pseudo-counts:

$$\alpha_m = \alpha_i + \alpha_j - 1, \quad \beta_m = \beta_i + \beta_j - 1, \quad (5.22)$$

where the subtraction avoids double-counting the initial prior if both edges were initialised from the same prior convention. Edge confidence may be combined using the maximum, a weighted average, or another conservative consolidation rule depending on the implementation setting.

The current internal location must also be updated after merging. If $\hat{\ell}_t$ is one of the merged nodes, then the representative node becomes the new internal current location:

$$\hat{\ell}_{t+1} = m. \quad (5.23)$$

This create–associate–merge mechanism is one of the main ways in which the present setting differs from a fixed-object POMDP. The internal object set $\hat{V}_t$ can expand as new observations arrive and can later contract when stable identity evidence resolves duplicate internal objects.

5.7. Role Beliefs

For each internal node $k \in \hat{V}_t$, the planner maintains a probability distribution over node roles:

$$b_k^t(r) = \Pr(R_k = r \mid I_t), \quad r \in \mathcal{R}. \quad (5.24)$$

In the simulated network domain, the role set is

$$\mathcal{R} = \{A, B, C, DB, O\}, \quad (5.25)$$

5. Internal State Representation: Knowledge Graph, Identity Association, and Role Beliefs

where A , B , and C denote domain-specific intermediate structural roles, DB denotes database-related nodes, and O denotes other nodes. These labels should not be interpreted as globally meaningful network classes. They are abstract role categories used by the planner to organise evidence and guide search in the simulated domain.

A newly created node is initialised with a prior distribution:

$$b_k^0(r) = p_0(r), \quad \sum_{r \in \mathcal{R}} p_0(r) = 1. \quad (5.26)$$

A uniform prior is used when no role-specific prior evidence is available, although other priors could be used if domain knowledge were available.

Role beliefs link raw observations to planning decisions. For example, a high DB belief increases the attractiveness of query-related actions and database-directed movement. A high credential-related belief increases the value of exploit or movement actions that may obtain credentials. Intermediate roles help estimate whether a node is likely to lead toward deeper network regions.

5.8. Role-Belief Update

Role beliefs are updated from multiple evidence sources. Let e_k^{t+1} denote the evidence received about node k at time $t + 1$. The evidence may include:

- fingerprint evidence, such as ports, services, or certificate fragments;
- structural evidence, such as newly observed neighbours or edge patterns;
- traffic evidence, such as contact counts, contact diversity, and contact modes;
- execution feedback, such as exploit success or credential acquisition;
- query feedback, such as successful database query or diagnostic query failure.

The update can be written abstractly as

$$\bar{b}_k^{t+1} = \Phi(b_k^t, e_k^{t+1}), \quad (5.27)$$

where Φ maps the previous belief and new evidence to a candidate posterior role distribution. The updated belief is then obtained by conservative fusion:

$$b_k^{t+1} = \text{Normalize}((1 - \eta)b_k^t + \eta\bar{b}_k^{t+1}), \quad (5.28)$$

where $\eta \in (0, 1]$ controls the strength of the new evidence.

A more explicit view is to assign evidence scores to each role:

$$z_k^{t+1}(r) = \lambda_f \phi_f(r; F_k^{t+1}) + \lambda_s \phi_s(r; \hat{G}_{t+1}) + \lambda_t \phi_t(r; T_{t+1}) + \lambda_x \phi_x(r; X_{t+1}) + \lambda_q \phi_q(r; Q_{t+1}), \quad (5.29)$$

5.9. Task Feedback and Database Belief

where ϕ_f is fingerprint-based evidence, ϕ_s is structural evidence, ϕ_t is traffic evidence, ϕ_x is exploit or capability evidence, and ϕ_q is query feedback. The candidate role distribution can be formed by normalising these scores:

$$\bar{b}_k^{t+1}(r) = \frac{\exp(z_k^{t+1}(r))}{\sum_{r' \in \mathcal{R}} \exp(z_k^{t+1}(r'))}. \quad (5.30)$$

This expression is intended as a conceptual description of evidence fusion. The implementation may use simpler rule-based score adjustments, but the semantic role of the update remains the same: each observation changes the planner's belief about the functional role of the internal object.

5.9. Task Feedback and Database Belief

The database-related quantity d_k^t is derived from the role belief and task feedback. A simple interpretation is

$$d_k^t \approx b_k^t(DB), \quad (5.31)$$

although the implementation may also include additional evidence from query feedback or local structure.

The action `query_db` is both a terminal action and an information source. If `query_db` succeeds at node k , the planner sets

$$b_k^{t+1}(DB) = 1, \quad b_k^{t+1}(r) = 0 \quad (r \neq DB), \quad (5.32)$$

and

$$d_k^{t+1} = 1. \quad (5.33)$$

The node is then treated as a confirmed database node.

If `query_db` fails, the interpretation depends on the reason for failure. A failure caused by explicit evidence that the node is not a database node should reduce $b_k(DB)$. However, a failure caused by missing credentials, insufficient privilege, or stochastic execution does not necessarily imply that the node is not database-related. Therefore, query feedback is handled diagnostically:

$$b_k^{t+1} = \Psi(b_k^t, Q_{t+1}), \quad (5.34)$$

where Ψ applies different updates for successful hits, explicit negative evidence, and non-diagnostic failure.

This distinction is important because overly aggressive negative updates would make the planner discard useful target candidates simply because the agent attempted a query before satisfying the required capability conditions.

5.10. Use of the Internal Representation in Later Planning Components

The internal knowledge graph is used by all later planning components:

5. *Internal State Representation: Knowledge Graph, Identity Association, and Role Beliefs*

- **Local subnet construction:** node role beliefs, edge confidence, movement-cost estimates, and current location are used to construct restricted local search views.
- **Frontier and progress scoring:** database belief, frontier structure, uncertainty, and capability state contribute to progress and frontier-related scores.
- **Local option generation:** local MCTS uses the internal graph and its estimated costs and reliabilities to predict success, progress, and cost.
- **Generalized policy retrieval:** structural signatures are extracted from the internal graph and role beliefs to retrieve reusable action templates.
- **Negative feedback and rebuilding:** mismatches between predicted and realised outcomes are used to update local statistics, revise edge reliability, and trigger local rebuilding when necessary.

The precise definitions of frontier scoring, progress scoring, local option values, and generalized policy-library signatures are given in later chapters. This chapter defines the representation on which those mechanisms operate.

5.11. Chapter Summary

This chapter defined the planner’s internal state representation without repeating the full problem-level action semantics already given in Chapter 4. The internal knowledge graph is an object-centric, revisable belief structure rather than a complete copy of the true environment. Internal nodes store temporary handles, stable identities when available, fingerprints, role beliefs, database-related belief, privilege estimates, and local statistics. Internal edges store existence confidence, movement-cost estimates, movement-reliability estimates, and blocking information. The chapter also explained how observations can create, associate, and merge internal objects, and how role beliefs are updated from fingerprint, structural, traffic, exploit, and query evidence. These mechanisms support the later local planning and experience-reuse components by providing a semantic, graph-structured, and dynamically revisable internal state.

Action Models and the Online Interaction Loop

The previous chapters defined the problem setting, the distinction between the hidden true environment and the planner’s internal knowledge state, and the internal knowledge graph used to represent discovered objects, identity evidence, role beliefs, and edge reliability. The formal semantics of the primitive actions were given in Section 4.6, where each action was specified in terms of its precondition, true effect, observation, and internal update.

This chapter does not repeat those definitions in full. Instead, it explains how the primitive actions are used operationally inside the planner’s online interaction loop. In particular, it describes how applicable actions are enumerated from the current internal state, how heterogeneous observations are interpreted, how the internal update pipeline combines graph revision and belief updates, and how the resulting closed loop supports local search, action aggregation, and experience reuse.

At time t , the planner selects an action a_t on the basis of its internal state I_t . The environment executes the action in the hidden true state s_t^* , returns a heterogeneous observation o_{t+1} and a realised execution cost c_t , and the planner applies the update operator

$$I_{t+1} = U(I_t, a_t, o_{t+1}, c_t). \quad (6.1)$$

The resulting system is therefore an online closed loop of

$$\text{decision} \rightarrow \text{execution} \rightarrow \text{observation} \rightarrow \text{revision}, \quad (6.2)$$

rather than a one-shot offline solution over a static graph. This view is closely related to online planning under partial observability (Kaelbling et al., 1998; Silver and Veness, 2010), while the network-domain instantiation follows the general motivation of penetration testing under uncertainty (Sarraute et al., 2012).

6.1. Role of This Chapter

The formal action definitions in Section 4.6 answer the question:

What does each primitive action mean at the level of the problem model?

The present chapter answers a different question:

6. Action Models and the Online Interaction Loop

How are those primitive actions integrated into the online planner?

This distinction is useful because the same action has two different roles. At the problem-model level, an action has a hidden true effect and produces an observation. At the planner level, the same action is also a source of internal revision: it may create or associate objects, revise role beliefs, update movement reliability, change privilege or credential state, or terminate the episode. The online planner therefore depends not only on the action semantics themselves, but also on how actions are enumerated, filtered, executed, and incorporated into the internal state.

6.2. Action Enumeration and Applicability

The primitive action space and action semantics were defined in Chapter 4. In particular, the planner operates over a state-dependent action set

$$\mathcal{A}(I_t) = \mathcal{A}_{\text{base}} \cup \{\text{move}(\hat{u}) : \hat{u} \in \hat{V}_t^{\text{cand}}\}, \quad (6.3)$$

where $\mathcal{A}_{\text{base}}$ contains the non-movement primitive actions and $\hat{V}_t^{\text{cand}} \subseteq \hat{V}_t$ is the set of currently considered internal movement targets, as defined in Section 4.5.

This chapter uses the same action set operationally inside the online interaction loop. The purpose here is not to redefine the primitive actions, but to specify how the planner enumerates actions that are currently usable under its internal knowledge state. Unlike a fixed action set in a fully specified MDP, the movement part of $\mathcal{A}(I_t)$ depends on the current internal graph. Newly discovered objects may become movement candidates, while suspected edges may later be weakened or blocked after failed movement feedback.

Not every syntactically valid action is useful or safe in every state. The planner therefore defines an applicability predicate

$$\chi(I_t, a) \in \{0, 1\}, \quad (6.4)$$

and the set of currently admissible actions is

$$\mathcal{A}_t^{\text{app}} = \{a \in \mathcal{A}(I_t) \mid \chi(I_t, a) = 1\}. \quad (6.5)$$

At the planner level, action applicability combines the formal preconditions from Section 4.6 with practical filtering rules used during online action enumeration:

1. the planner must have a valid current internal location;
2. the action-specific precondition must hold;
3. the estimated action cost must fit within the remaining budget;
4. repeated low-value actions may be filtered by node-level action limits;
5. movement actions must target currently usable internal candidates;
6. identity confirmation is considered only when identity is unknown or ambiguous.

6.3. Operational Roles of Primitive Actions

The estimated cost of an action is denoted by

$$\hat{c}(a \mid I_t). \quad (6.6)$$

Since realised costs may be noisy, budget feasibility is checked conservatively:

$$\lambda_{\text{safe}} \hat{c}(a \mid I_t) \leq B_t, \quad \lambda_{\text{safe}} \geq 1. \quad (6.7)$$

This conservative check does not mean that the true realised cost is known in advance. It only prevents the planner from selecting actions that are estimated to be unsafe under the remaining budget. After execution, the actual budget update uses the realised cost:

$$B_{t+1} = B_t - c_t. \quad (6.8)$$

The main internal fields affected by each primitive action were summarised in Table 5.1. In the online loop, these updates are applied through the general update operator $U(I_t, a_t, o_{t+1}, c_t)$ defined in Chapter 4.

6.3. Operational Roles of Primitive Actions

The primitive actions play different operational roles in the online loop. Their formal preconditions, effects, observations, and internal updates are defined in Section 4.6. Table 5.1 (page 40) summarises the main observation types and internal knowledge-graph fields affected by each primitive action. This section uses those action definitions as the basis for explaining how the planner enumerates, filters, and executes actions during online interaction.

This table should not be read as a substitute for the formal action definitions. Rather, it explains why the actions are needed by the planner. In particular, the actions are not merely different ways of moving through a graph. Some actions primarily expand the internal representation, some reduce semantic or identity uncertainty, some test the reliability of suspected edges, some modify capability state, and only one action formally verifies the terminal objective.

6.4. Heterogeneous Observation Handling

The environment returns heterogeneous observations, as defined in Equation 4.10. Different actions provide different kinds of evidence, so a single fixed-dimensional observation vector would obscure important semantic distinctions. In this chapter, the observation tuple is used operationally inside the online interaction loop.

When necessary, the node-observation field N_{t+1}^{obs} is unpacked as

$$N_{t+1}^{\text{obs}} = (H_{t+1}, F_{t+1}), \quad (6.9)$$

where H_{t+1} contains temporary handles or local identifiers, and F_{t+1} contains partial fingerprints such as observed ports, services, certificate fragments, or other attributes. The remaining fields follow the

6. Action Models and the Online Interaction Loop

notation of Chapter 4: E_{t+1}^{obs} stores observed or hypothesised relations, T_{t+1} stores traffic or contact statistics, C_{t+1} stores stable identity evidence, X_{t+1} stores exploit or credential feedback, Q_{t+1} stores query feedback, and M_{t+1} stores movement feedback.

Not all fields are present for every action. For example, `scan` mainly returns handles, fingerprints, and local relation evidence; `traffic_sense` mainly returns contact statistics; `move` returns movement feedback; `exploit` returns capability feedback; `confirm_identity` returns stable identity evidence; and `query_db` returns terminal-query feedback.

A key modelling constraint is preserved throughout the loop: early observations do not reveal stable identities. The actions `scan`, `traffic_sense`, `move`, `exploit`, and `query_db` may reveal temporary handles, partial fingerprints, movement feedback, or task feedback, but they do not by themselves reveal stable identity information. Stable identity evidence is returned only by `confirm_identity`. This constraint is what makes identity association a planning-relevant problem rather than a simple lookup.

6.5. Internal Update Pipeline

After each action execution, the planner applies the update operator

$$I_{t+1} = U(I_t, a_t, o_{t+1}, c_t). \quad (6.10)$$

Although this is written as a single operator, it consists of several conceptually distinct update stages:

1. **Budget update:** subtract the realised cost c_t from the remaining budget.
2. **Object update:** create new internal nodes, associate new evidence with existing nodes, or merge nodes if stable identity evidence is available.
3. **Edge update:** add newly observed relations, revise edge confidence, update movement cost estimates, and mark impossible or blocked relations when movement feedback contradicts the internal graph.
4. **Belief update:** revise role beliefs, database belief, uncertainty estimates, movement reliability, and capability-related beliefs.
5. **Runtime-state update:** update current location, privilege state, credential state, success flag, visit counts, and action counters.
6. **Planning-support update:** refresh local statistics used by later subnet construction, progress scoring, local option generation, and policy-library retrieval.

This update pipeline explains how action observations affect the internal knowledge graph introduced in Chapter 5. The numerical details of some updates, such as role-belief fusion, movement-reliability updates, and progress scoring, are described in the relevant representation and planning chapters. The important point here is that the online loop does not merely append observations to a log. It actively revises the planner’s internal model after every action.

Algorithm 1: Online interaction loop

Input: Initial internal state I_0 , hidden environment state s_0^* , initial budget B_0
Output: Execution trajectory τ and success indicator

```

1  $t \leftarrow 0$ ;
2  $\tau \leftarrow ()$ ;
3 while  $B_t > 0$  and  $h_t = 0$  do
4   Enumerate primitive actions  $\mathcal{A}(I_t)$  from the current internal graph;
5   Filter applicable actions:  $\mathcal{A}_t^{\text{app}} = \{a \in \mathcal{A}(I_t) : \chi(I_t, a) = 1\}$ ;
6   Select an action using the current planner:  $a_t \leftarrow \pi_{\text{plan}}(I_t, \mathcal{A}_t^{\text{app}})$ ;
7   Execute  $a_t$  in the hidden environment:  $(s_{t+1}^*, o_{t+1}, c_t) \sim P_{\text{env}}(\cdot \mid s_t^*, a_t)$ ;
8   Update the internal state:  $I_{t+1} \leftarrow U(I_t, a_t, o_{t+1}, c_t)$ ;
9   Append  $(I_t, a_t, o_{t+1}, c_t, I_{t+1})$  to  $\tau$ ;
10   $t \leftarrow t + 1$ ;
11 return  $\tau, h_t$ ;
```

6.6. Online Interaction Algorithm

The full online interaction loop can be written as Algorithm 1. The algorithm is intentionally expressed at the problem level rather than as program code. Its purpose is to show how action enumeration, planning, execution, observation, and internal revision are connected.

In this loop, π_{plan} is not a fixed closed-form policy. In later chapters, it is realised through local sub-net construction, budgeted MCTS, approximate non-dominated local option generation, policy-library retrieval, and global aggregation. The action model defined here provides the interface between those planning components and the environment.

6.7. Prediction, Execution Feedback, and Model Revision

A key feature of the interaction loop is that planner predictions can be corrected by execution feedback. The internal graph may contain a suspected edge, but an actual movement attempt can reveal failure or non-adjacency. A node may have high database belief, but a query attempt can provide diagnostic feedback. A node may appear only weakly credential-relevant, but a successful exploit can revise its role belief and capability value. These feedback effects are essential because the planner operates under incomplete information.

Thus, each primitive action contributes to at least one of the following correction mechanisms:

- **Structural correction:** movement feedback can confirm, weaken, or refute suspected relations.
- **Semantic correction:** traffic, exploit, and query feedback can revise role beliefs.
- **Identity correction:** identity confirmation can merge duplicated internal objects.

6. Action Models and the Online Interaction Loop

- **Capability correction:** exploit feedback can update privilege and credential state.
- **Terminal correction:** query feedback is the only source of confirmed task success.

These correction mechanisms are later used by the planner to detect prediction–execution mismatch, refresh local subnets, and update reusable action templates. Therefore, the action model is not only a way to generate transitions; it is also a source of online model revision.

6.8. Chapter Summary

This chapter explained how the primitive actions defined formally in Section 4.6 are integrated into the online interaction loop. Rather than repeating the full action semantics, it focused on action enumeration, applicability filtering, heterogeneous observation handling, internal update stages, and the decision–execution–revision cycle. The key modelling point is that every action is interpreted at two levels: it has a hidden true effect in the environment and an observable effect on the planner’s internal knowledge state. This distinction is essential for planning under partial observability, dynamic object discovery, and initially unstable identities.

The next chapters build on this loop. Local subnet construction uses the current internal graph and role beliefs; budgeted MCTS uses the admissible action set and action model to simulate local futures; approximate non-dominated option generation uses predicted success, progress, and cost derived from these local rollouts; and the generalized policy library stores reusable action templates over internal graph structure rather than over hidden true node identities.

Local Subnet Search and Approximate Non-Dominated Option Generation

The preceding chapters defined the planner’s internal state, action semantics, and online interaction loop. In particular, Chapter 4 defined the hidden true state, internal knowledge state, primitive action semantics, budget constraint, and internal progress signal, while Chapter 5 defined the knowledge graph, role beliefs, identity association, and edge-reliability estimates used by the planner. This chapter builds on those definitions and introduces the local search component of the framework.

If one were to perform a single global search directly over the entire internal knowledge graph $\hat{G}_t$, then as the number of discovered objects and relations grows, both the branching factor and the simulation cost would increase. Moreover, remote parts of the graph that are weakly relevant to the current decision may interfere with local reasoning. For this reason, the proposed framework does not apply a single global search over the full internal graph. Instead, it constructs restricted local subnets around selected anchor nodes and performs budgeted Monte Carlo Tree Search (MCTS) inside these subnets. The resulting local candidates are then passed to the global decision stage.

This design is related to localised approximation, hierarchical planning, and online Monte Carlo search under uncertainty (Silver and Veness, 2010; Ye et al., 2017; Bercher et al., 2019; Browne et al., 2012). However, the present framework differs from standard global MCTS and fixed hierarchical decomposition in two important respects. First, the local solving domain is not predefined, but is constructed dynamically from the current object-level knowledge graph. Second, local search results are not collapsed immediately into a single scalar action value. Instead, the local solver keeps an approximate set of non-dominated local options over predicted success, predicted progress, and predicted cost. This set is not an exact Pareto set in the formal multi-objective optimisation sense; it is a finite, sampled, and approximate set of locally non-dominated candidates generated under restricted subnet search.

7.1. Pure Global MCTS Baseline

As a reference point, Algorithm 2 summarises the pure global MCTS baseline used in the experiments. Unlike the proposed subnet-based framework, this baseline performs search directly on the current discovered graph without constructing a strict local subnet or maintaining an approximate non-dominated local option set.

7. Local Subnet Search and Approximate Non-Dominated Option Generation

Algorithm 2: Pure Global MCTS Baseline

Input: current internal state I_t ; planner parameters Θ
Output: primitive action a_t

```

1  $root \leftarrow \text{NODE}(I_t)$ 
2 for  $iter \leftarrow 1$  to  $\Theta.mcts\_iters$  do
3    $v \leftarrow root$ 
4    $\tau \leftarrow []$ 
   // Selection
5   while  $v$  is expanded and  $v$  is not terminal do
6      $v \leftarrow \text{SELECTUCT}(v)$ 
7     append  $v$  to  $\tau$ 
   // Expansion
8   if  $v$  is not terminal then
9      $\mathcal{A}(v) \leftarrow \text{PRIMITIVEACTIONS}(\text{state}(v), \Theta)$ 
10     $\text{EXPAND}(v, \mathcal{A}(v))$ 
11     $v \leftarrow \text{CHOOSEONECHILD}(v)$ 
12    append  $v$  to  $\tau$ 
   // Rollout on the current discovered graph
13    $(r_p, r_g, r_c) \leftarrow \text{ROLLOUT}(\text{state}(v), \Theta.rollout\_depth)$ 
   // Backup
14    $\text{BACKUP}(\tau, r_p, r_g, r_c)$ 
15  $v^* \leftarrow \text{BESTCHILD}(root)$ 
16  $a_t \leftarrow \text{INSTANTIATE}(v^*.first\_action, I_t)$ 
17 return  $a_t$ 

```

Algorithm 2 serves as a baseline planner without subnet restriction, action-template abstraction, or generalized policy reuse. It applies MCTS directly to the current internal graph and returns the first action associated with the highest-valued child of the root. This baseline is useful because it isolates the effect of local subnet restriction and option abstraction in later experimental comparisons.

7.2. Construction of Local Subnets

Recall from Chapter 4 that the planner does not have direct access to the hidden true graph $G^* = (V^*, E^*)$. Instead, local search operates on the current internal knowledge graph

$$\hat{G}_t = (\hat{V}_t, \hat{E}_t), \quad (7.1)$$

which contains only the objects, relations, costs, reliabilities, and role beliefs currently represented in the internal state I_t .

Given an anchor node $a \in \hat{V}_t$, the framework defines a local subnet centred on a as

$$S(a, t) = (V_{a,t}^{\text{sub}}, E_{a,t}^{\text{sub}}, F_{a,t}), \quad (7.2)$$

where $V_{a,t}^{\text{sub}} \subseteq \hat{V}_t$ is the subnet node set, $E_{a,t}^{\text{sub}} \subseteq \hat{E}_t$ is the subnet edge set, and $F_{a,t} \subseteq V_{a,t}^{\text{sub}}$ is the set of frontier nodes in the subnet.

7.3. Frontier Nodes and Frontier Score

Thus, $S(a, t)$ is not a subgraph of the hidden true environment G^* . It is a restricted local extract of the planner’s current internal graph $\hat{G}_t$. The action semantics and observation updates that create and revise $\hat{G}_t$ were defined in Chapter 4 and Chapter 5; this chapter uses the resulting internal graph as the input to local search.

The first stage of subnet construction is a bounded graph expansion from the anchor. Abstractly, one may write

$$V_{a,t}^{(0)} = \left\{ v \in \hat{V}_t : d_{\hat{G}_t}(a, v) \leq r_{\max} \right\}, \quad (7.3)$$

where $d_{\hat{G}_t}$ denotes hop distance in the current internal graph and $r_{\max}$ is a maximum expansion radius. In the implemented planner, this expansion is additionally constrained by maximum node count $N_{\max}$, maximum edge count $E_{\max}$, and edge-confidence thresholds. Thus, the subnet is not simply a radius ball in the true environment. It is a restricted extract of the currently known and currently trusted local neighbourhood.

The main subnet-size parameters can be summarised as

$$\Theta_{\text{sub}} = (r_{\max}, N_{\max}, E_{\max}, \rho_{\min}), \quad (7.4)$$

where $\rho_{\min}$ is the minimum edge-confidence threshold used when expanding through internal edges. These parameters determine the computational scope of local MCTS. They are treated as planner parameters rather than as fixed theoretical constants, and their sensitivity is evaluated in the experimental chapter.

7.3. Frontier Nodes and Frontier Score

A frontier node is an internal node in or near the boundary of the current local subnet that may lead to further discovery or deeper progress. The notion is related to frontier-based exploration, where the frontier marks the boundary between known and unknown regions (Yamauchi, 1997). In this thesis, the environment is a graph-structured network rather than a geometric map, so a frontier node should be understood as a planning-relevant boundary point in the currently discovered internal graph.

The planner assigns a node-level frontier score

$$\phi_{\text{frontier}}(v \mid I_t), \quad (7.5)$$

which estimates whether node v is a useful boundary point for local exploration or deeper movement. This score is not the same as the progress score used inside local MCTS. The frontier score is a *node-level subnet-construction and anchor-ranking quantity*; the progress score is a *state-level rollout evaluation quantity*. Their relationship is explained further in Section 7.8.

The frontier score can be interpreted as a tractable approximation to local value of information. This interpretation is related to value-of-information reasoning (Howard, 1966). Ideally, for a candidate frontier node v , one would evaluate the expected improvement in future planning value obtained by moving toward, sensing, or expanding the local search view around v :

$$\text{VOI}(v \mid I_t) = \mathbb{E}_{o \sim P(o \mid I_t, a_v)} [V(U(I_t, a_v, o))] - V(I_t) - \lambda_c \hat{C}(a_v), \quad (7.6)$$

7. Local Subnet Search and Approximate Non-Dominated Option Generation

where a_v denotes a local exploratory action or action sequence associated with v , o is the resulting observation, U is the internal update operator, V is the planner's value estimate over internal states, and $\hat{C}(a_v)$ is the estimated cost of reaching or observing the frontier node.

Computing this exact value of information is not feasible in the present setting. The planner does not have a complete observation model, the internal object set can change through discovery and identity merging, and the future value of an observation depends on later local search and global aggregation. Therefore, the framework uses a frontier score as a practical proxy for local value of information. The proxy decomposes the potential value of a frontier node into four computable components: task relevance, role uncertainty, graph-expansion potential, and novelty.

The first component is a novelty factor:

$$u(v) = \frac{1}{1 + \text{scanCount}(v)}. \quad (7.7)$$

This term favours less scanned nodes and models diminishing marginal information value from repeatedly observing the same node.

The second component is database-related potential:

$$d_{\text{DB}}(v) = \max \left(b_v(\text{DB}), \lambda_{\max} \max_{w \in N(v)} b_w(\text{DB}), \lambda_{\text{mean}} \text{mean}_{w \in N(v)} b_w(\text{DB}) \right). \quad (7.8)$$

where $\lambda_{\max}, \lambda_{\text{mean}} \in [0, 1]$ are planner parameters controlling the influence of the strongest neighbouring database belief and the average neighbouring database belief.

This term approximates task-relevant value of information. A frontier node is more valuable if it is itself database-like or if it is close to neighbours with high database belief, because exploring such a region is more likely to improve the chance of eventual terminal verification.

The third component is role-belief uncertainty. Let

$$H(b_v) = - \sum_{r \in \mathcal{R}} b_v(r) \log b_v(r) \quad (7.9)$$

be the entropy of the current role-belief distribution at node v . A normalised uncertainty term can be written as

$$h(v) = \frac{H(b_v)}{\log |\mathcal{R}|}. \quad (7.10)$$

This term approximates possible role information gain. If the role belief of a node is highly uncertain, further sensing or movement around that node may reduce uncertainty and improve later action selection.

The fourth component is local structural expansion potential:

$$\delta(v) = \min \left(1, \frac{\deg(v)}{d_0} \right). \quad (7.11)$$

where $d_0 > 0$ is a degree normalisation constant. It controls the degree level at which the structural-expansion term saturates.

This term gives higher value to nodes with greater local connectivity, since such nodes may expose additional movement options, downstream structure, or alternative routes. In practice, this structural

7.4. Frontier-Directed Path Expansion

value is most useful when combined with novelty, because a high-degree node that has already been fully explored should not remain indefinitely attractive.

A general frontier score can therefore be written as

$$\phi_{\text{frontier}}(v \mid I_t) = w_D d_{\text{DB}}(v) + w_R u(v) h(v) + w_G u(v) \delta(v) + w_N u(v). \quad (7.12)$$

Here w_D , w_R , w_G , and w_N are planner parameters controlling the relative importance of task-relevant database potential, role-uncertainty reduction, graph-expansion potential, and novelty. The multiplicative use of $u(v)$ in the uncertainty and graph-expansion terms reflects the intuition that repeated observations of the same node should have decreasing marginal value.

If the frontier score is used to rank nodes from a specific anchor a , a reachability discount may also be included:

$$\phi_{\text{frontier}}(v \mid I_t, a) = \rho_{\text{reach}}(a, v) [w_D d_{\text{DB}}(v) + w_R u(v) h(v) + w_G u(v) \delta(v) + w_N u(v)], \quad (7.13)$$

where $\rho_{\text{reach}}(a, v)$ decreases with the estimated distance or movement cost from anchor a to node v . In the implementation used in this thesis, a simplified additive version of this proxy is used for practical ranking, while cost and distance are also considered later during active-subnet selection and global aggregation.

The frontier score is therefore not intended to be a universal frontier metric or an exact value-of-information computation. It is a local, computable ranking signal that approximates which boundary nodes are likely to provide useful task-relevant information, reduce semantic uncertainty, expose additional graph structure, or support movement toward database-related regions. The numerical weights are planner parameters rather than part of the formal problem definition, and their sensitivity is evaluated experimentally.

7.4. Frontier-Directed Path Expansion

After the initial bounded expansion, the subnet may be enhanced by adding known paths from the anchor to selected high-frontier nodes. This mechanism is called *frontier-directed path expansion*. It is important to clarify what this means.

Frontier-directed path expansion does *not* create executable shortcut edges in the environment. It does not allow the agent to move directly between non-adjacent nodes, and it does not change the action semantics defined in Chapter 4. All movement actions are still executed through the true environment and must satisfy the movement semantics of `move`. Instead, frontier-directed path expansion only affects which already represented internal nodes and already represented internal edges are included in the local subnet given to MCTS.

The motivation is that a strict radius- or size-limited subnet can exclude intermediate nodes on a known path toward a high-value boundary region. In that case, local MCTS may remain confined to the immediate neighbourhood of the anchor and may fail to evaluate a direction that is promising but slightly outside the first local expansion. By explicitly including the currently known path toward selected high-frontier nodes, the subnet gives local MCTS a more coherent planning view of how to move

7. Local Subnet Search and Approximate Non-Dominated Option Generation

toward that boundary. This is a planning-domain construction mechanism, not an additional primitive action.

Formally, let $F_{a,t}^K$ be the set of top-ranked frontier nodes selected according to ϕ_{frontier} . For each $f \in F_{a,t}^K$, let

$$P_{\hat{G}_t}(a, f) \quad (7.14)$$

denote a shortest path from a to f in the current internal graph, using only edges whose confidence exceeds the expansion threshold. The frontier-directed expansion adds

$$V_{a,t}^{\text{path}} = \bigcup_{f \in F_{a,t}^K} P_{\hat{G}_t}(a, f) \quad (7.15)$$

to the subnet node set, subject to the same node and edge limits:

$$V_{a,t}^{\text{sub}} = V_{a,t}^{(0)} \cup V_{a,t}^{\text{path}}. \quad (7.16)$$

The expected benefit is to reduce local-search myopia under strict subnet limits. The mechanism is most useful when the current graph contains a known but multi-hop route toward a promising frontier region, while the immediate anchor neighbourhood alone does not expose enough of that route to local MCTS. Its effect is evaluated by ablation in the experimental chapter by comparing runs with and without frontier-directed path expansion.

7.5. Anchor Selection and the Active Subnet Set

Local subnets are not generated once and for all. Instead, they are constructed, cached, and reused around a dynamically selected set of anchor nodes. Recall from Section 7.2 that a local subnet $S(a, t)$ is centred on an internal anchor node $a \in \hat{V}_t$ and is constructed from the current internal knowledge graph $\hat{G}_t$, not from the hidden true graph.

Let the current internal state be I_t . The system first constructs a candidate anchor-node set

$$\mathcal{V}_t^{\text{anch}} \subseteq \hat{V}_t. \quad (7.17)$$

A simple candidate set consists of the current internal location and its known one-hop neighbours:

$$\mathcal{V}_t^{\text{anch}} = \{\hat{\ell}_t\} \cup N_{\hat{G}_t}(\hat{\ell}_t), \quad (7.18)$$

where $N_{\hat{G}_t}(\hat{\ell}_t)$ denotes the neighbour set of $\hat{\ell}_t$ in the current internal graph.

These candidates are ranked by an anchor score:

$$\psi_{\text{anchor}}(v \mid I_t) = \alpha_1 \phi_{\text{frontier}}(v \mid I_t) + \alpha_2 b_v(DB) + \alpha_3 \frac{H(b_v)}{H(b_v) + 1}. \quad (7.19)$$

Here $\phi_{\text{frontier}}(v \mid I_t)$ is the node-level frontier score defined in Section 7.3, $b_v(DB)$ is the planner's current belief that internal node v corresponds to a database-relevant node, and $H(b_v)$ is the entropy of the role-belief distribution at node v . The term

$$\frac{H(b_v)}{H(b_v) + 1} \quad (7.20)$$

7.6. Strict Subnet Views and Restricted Solving Domains

is a bounded uncertainty term: it increases with role uncertainty but remains below one.

The coefficients $\alpha_1, \alpha_2, \alpha_3 \geq 0$ are planner parameters. They determine the relative importance of frontier value, database-related belief, and semantic uncertainty when choosing where local solving should be centred. They are not theoretical constants. Their purpose is to provide a practical ranking signal for selecting anchor nodes that may support discovery, target approach, or useful uncertainty reduction.

Given a maximum number of newly constructed anchor subnets K_{anch} , the selected anchor set may be written as

$$\mathcal{V}_t^{\text{sel}} = \text{TopK}_{v \in \mathcal{V}_t^{\text{anch}}} (\psi_{\text{anchor}}(v \mid I_t), K_{\text{anch}}). \quad (7.21)$$

This notation means that the planner selects at most K_{anch} candidate anchors with the highest anchor scores. Ties may be broken by distance to the current internal location or by insertion order.

The framework also maintains a bounded subnet cache. A cached subnet stores its anchor, the local graph extract, a structural fingerprint of the local context, and the approximate non-dominated local options generated by previous local solving. When the cache reaches its capacity limit, older or less relevant entries may be discarded. Thus, before each global decision, the planner may reuse existing cached subnets, rebuild stale subnets, or add new subnets around selected anchors.

Let $d_t(\hat{\ell}_t, a)$ be the current internal distance estimate from the current location $\hat{\ell}_t$ to anchor a . For an anchor a with a cached or newly constructed subnet, the active-subnet ranking score is

$$\psi_{\text{active}}(a \mid I_t) = \beta_1 \frac{1}{1 + d_t(\hat{\ell}_t, a)} + \beta_2 \phi_{\text{frontier}}(a \mid I_t) + \beta_3 b_a(DB). \quad (7.22)$$

The first term favours subnets whose anchors are close to the current internal location. The second term favours subnets centred on frontier-relevant nodes. The third term favours subnets with database-related belief. As above, the coefficients $\beta_1, \beta_2, \beta_3 \geq 0$ are planner parameters controlling the relative importance of proximity, frontier value, and database relevance.

Let $\mathcal{C}_t^{\text{sub}}$ denote the set of currently cached and newly constructed subnet anchors. Given an active-subnet limit K_{active} , the active subnet-anchor set is

$$\mathcal{V}_t^{\text{active}} = \text{TopK}_{a \in \mathcal{C}_t^{\text{sub}}} (\psi_{\text{active}}(a \mid I_t), K_{\text{active}}). \quad (7.23)$$

The active subnet set is then

$$\mathcal{S}_t^{\text{active}} = \{S(a, t) : a \in \mathcal{V}_t^{\text{active}}\}. \quad (7.24)$$

Thus, the active subnet set is not the set of all cached subnets. It is a small subset of currently relevant local regions, selected according to proximity, frontier value, and database-related belief. Only options from active subnets are passed forward to the global action aggregator. This keeps global decision making focused on local regions that are reachable or strategically promising under the current internal state.

7.6. Strict Subnet Views and Restricted Solving Domains

A central design choice of the framework is that local MCTS operates on a strict subnet view. Let

$$S(a, t) = (V_{a,t}^{\text{sub}}, E_{a,t}^{\text{sub}}, F_{a,t}) \quad (7.25)$$

7. Local Subnet Search and Approximate Non-Dominated Option Generation

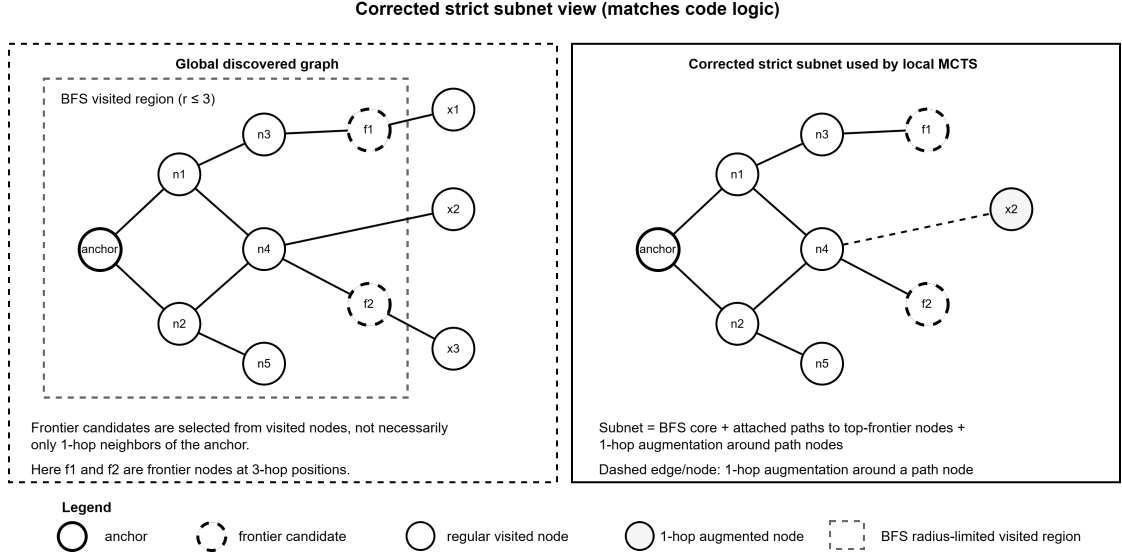


Figure 7.1.: Illustration of a strict subnet view. Left: a local subnet extracted around the current anchor from the discovered graph, including internal subnet nodes and frontier nodes. Right: the restricted view used by local solving. Nodes outside the subnet are excluded from local MCTS, so option generation is performed only within the subnet-induced search domain.

be a subnet. The corresponding strict subnet view is the restricted graph

$$\hat{G}_t^{(a)} = \left(V_{a,t}^{\text{sub}}, E_{a,t}^{\text{sub}} \right). \quad (7.26)$$

During local solving, the rollout model is restricted to this graph. Actions, observations, and graph updates that would involve objects outside the subnet are filtered or ignored in the local simulation.

Figure 7.1 illustrates the strict subnet view. The left-hand side shows a subnet extracted from the discovered internal graph, including nearby subnet nodes and selected frontier nodes. The right-hand side shows the restricted view used by local MCTS. Nodes and edges outside the subnet are excluded from local rollout.

This restriction has two consequences. First, it prevents remote and weakly relevant parts of the discovered graph from diluting local rollout statistics. Second, it makes local options more comparable across episodes and network instances: an option generated for a subnet is associated with a local structural context rather than with the entire global graph. This is one of the reasons why local search outputs can later be abstracted into action templates and stored in the generalized policy library.

The strict subnet view should therefore be understood as a local domain-defining mechanism. It specifies what local MCTS is allowed to see, update, and evaluate. It is not merely an attention mask over the full graph.

7.7. Budgeted MCTS Within a Subnet

Given a strict subnet view $\hat{G}_t^{(a)}$ and a corresponding local state $I_t^{(a)}$, the framework performs budget-constrained MCTS inside this restricted solving domain. MCTS is a sampling-based search method for sequential decision problems and has been widely used in large planning domains (Kocsis and Szepesvári, 2006; Browne et al., 2012). In the present setting, the purpose of local MCTS is not to compute a complete policy over the full environment. Rather, it estimates the value of locally available first actions under a restricted view and a finite simulation budget.

Let $B \leq B_t$ be the local budget under consideration. A single local solve returns a first action

$$a^*(B) \tag{7.27}$$

together with three predicted quantities

$$\hat{p}(B), \quad \hat{g}(B), \quad \hat{c}(B). \tag{7.28}$$

These quantities correspond to predicted terminal-success value, predicted progress value, and predicted cumulative cost. They are the local-search estimates of the more abstract quantities introduced in Chapter 4.

More explicitly, for a local option ω produced under budget B , the values can be understood as empirical estimates:

$$\hat{p}(\omega) \approx \frac{1}{N_\omega} \sum_{n=1}^{N_\omega} \mathbf{1}\{\text{Succ}(\tau_\omega^{(n)}) = 1\}, \tag{7.29}$$

$$\hat{g}(\omega) \approx \frac{1}{N_\omega} \sum_{n=1}^{N_\omega} \Psi_g(\tau_\omega^{(n)}), \tag{7.30}$$

$$\hat{c}(\omega) \approx \frac{1}{N_\omega} \sum_{n=1}^{N_\omega} C(\tau_\omega^{(n)}), \tag{7.31}$$

where $\tau_\omega^{(n)}$ is the n -th local rollout beginning with option ω , N_ω is the number of sampled rollouts associated with that option, Ψ_g is the progress-evaluation functional, and $C(\tau_\omega^{(n)})$ is the cumulative local rollout cost. These are not exact values under the hidden true environment. They are empirical predictions under the planner's current internal model and strict subnet view.

During MCTS, each search node maintains visit counts and accumulated estimates of success, progress, and cost. A rollout produces a success indicator

$$r_{\text{succ}} = \mathbf{1}\{\text{rollout reaches successful query_db}\}, \tag{7.32}$$

a progress value

$$r_{\text{prog}} = g(I_{\text{roll}}), \tag{7.33}$$

and an accumulated rollout cost. The backup step aggregates these quantities so that each root child can be evaluated by predicted success, predicted progress, and predicted cost.

The primitive action set used inside local MCTS is inherited from the action model defined in Chapter 4, but filtered by the strict subnet view and action-applicability checks. Therefore, local search does not simulate arbitrary actions over the full discovered graph. It simulates only those actions that are admissible in the current internal state and compatible with the restricted local solving domain.

7.8. Progress Score as a Belief-State Potential

The external task objective is terminal success under a budget constraint: the episode succeeds only when `query_db` succeeds at a true database node with the required capability conditions. However, this terminal signal is sparse. In many early local rollouts, no successful database query is reached, even though the rollout may still reveal useful structure, improve role beliefs, obtain capability, or move the agent closer to database-relevant regions. To provide denser guidance for local search, the planner uses an internal progress score

$$g(I_t) \in [0, 1]. \quad (7.34)$$

The progress score should be interpreted as a belief-state potential rather than as an additional external task objective. It estimates how favourable the current internal state is for future success, given the planner’s current knowledge graph, role beliefs, capability state, and discovered local structure. This idea is related to potential-based reward shaping, where an auxiliary potential function can provide denser learning or planning feedback without redefining the original task objective (Ng et al., 1999). In this thesis, the use of progress is more heuristic: because the setting involves partial observability, dynamic object discovery, and finite-budget local MCTS estimates, no policy-invariance guarantee is claimed. Instead, progress is used as a practical belief-state value approximation for guiding local search and global action aggregation.

In the implementation, the progress score combines four groups of internal evidence:

$$g(I_t) = w_{\text{pot}}g_{\text{pot}}(I_t) + w_{\text{exp}}g_{\text{exp}}(I_t) + w_{\text{unc}}g_{\text{unc}}(I_t) + w_{\text{cap}}g_{\text{cap}}(I_t), \quad (7.35)$$

where g_{pot} measures database-related potential, g_{exp} measures discovered-structure progress, g_{unc} measures local semantic uncertainty, and g_{cap} measures capability progress. The weights are planner parameters, and the final value is clipped to $[0, 1]$. The implementation-level definitions of these four components are given in Appendix A.1.

The database-potential term is based on the database-role belief of reachable internal nodes, discounted by estimated movement distance from the current internal location. The discovered-structure term is a saturating function of the number of discovered internal nodes and edges, so that structure discovery gives useful early guidance but cannot grow without bound. The uncertainty term gives a small information-potential signal for semantically uncertain current locations, with a penalty for repeated scanning at the same node. The capability term reflects whether the planner has obtained credentials or privilege that make a successful `query_db` more likely. These components are all computed from the internal state I_t , not from the hidden true environment graph.

It is important to distinguish this progress score from the frontier score in Section 7.3. The frontier score $\phi_{\text{frontier}}(v \mid I_t)$ is a node-level ranking signal used for subnet construction, anchor selection, and frontier-directed path expansion. By contrast, $g(I_t)$ is a state-level potential used in local rollout evaluation, option prediction, global aggregation, and execution feedback. The two scores share some evidence sources, such as role beliefs, uncertainty, and database potential, but they serve different roles in the planner. A high-frontier node may contribute to progress if moving toward it improves future success potential, but frontier score and progress score are not the same quantity.

7.9. ϵ -Sweep and Approximate Non-Dominated Local Options

The progress weights are not claimed to be globally optimal. Their role is to provide a practical value approximation when terminal success feedback is sparse. The sensitivity experiments in Chapter 11 evaluate the effect of changing the progress weight in the global aggregation stage. The results show that removing progress can produce fast successful runs in favourable cases, but it reduces robustness and increases the risk of failure, supporting the interpretation of progress as a stabilising shaping signal.

7.9. ϵ -Sweep and Approximate Non-Dominated Local Options

Given a subnet $S(a, t)$, the local solver does not invoke MCTS only once. Instead, it constructs a budget grid

$$\mathcal{B}_t = \{B_1, \dots, B_K\}, \quad 0 < B_1 < \dots < B_K \leq B_t, \quad (7.36)$$

and runs budgeted MCTS once for each budget value B_k . This is inspired by the ϵ -constraint idea in multi-objective optimisation, where one explores trade-offs by optimising under different constraints (Mavrotas, 2009). In this thesis, the budget grid is used pragmatically to produce local candidates under different resource assumptions.

For each B_k , the local solver returns a first action and its predicted values:

$$\omega_k = (a_k, \hat{p}_k, \hat{g}_k, \hat{c}_k). \quad (7.37)$$

Let

$$\Omega_t^{(a)} = \{\omega_1, \dots, \omega_K\} \quad (7.38)$$

be the finite candidate set generated for subnet anchor a .

The solver then applies dominance filtering to this finite sampled set. The result should not be called a complete Pareto set. A complete Pareto set would require reasoning over the full feasible decision space, whereas the local solver only samples a finite number of candidates using finite-budget MCTS inside a restricted subnet. Therefore, the output is an approximate non-dominated local option set.

The filtering mode depends on whether local success estimates appear informative. Let

$$p_{\max} = \max_{\omega_k \in \Omega_t^{(a)}} \hat{p}_k. \quad (7.39)$$

The threshold p_{gate} determines whether the local solver treats terminal-success predictions as sufficiently informative:

$$\text{mode} = \begin{cases} \text{success,} & \text{if } p_{\max} > p_{\text{gate}}, \\ \text{progress,} & \text{otherwise.} \end{cases} \quad (7.40)$$

In success mode, the primary value is predicted success:

$$s_k = \hat{p}_k. \quad (7.41)$$

In progress mode, the primary value is predicted progress:

$$s_k = \hat{g}_k. \quad (7.42)$$

7. Local Subnet Search and Approximate Non-Dominated Option Generation

For two local options ω_i and ω_j , option ω_j dominates ω_i if

$$s_j \geq s_i, \quad \hat{c}_j \leq \hat{c}_i, \quad (7.43)$$

and at least one inequality is strict. The retained set is therefore

$$\text{ND}(\Omega_t^{(a)}) = \left\{ \omega_i \in \Omega_t^{(a)} : \nexists \omega_j \in \Omega_t^{(a)} \text{ such that } \omega_j \succ \omega_i \right\}. \quad (7.44)$$

The purpose of p_{gate} is to prevent weak or noisy terminal-success estimates from dominating early local search. If all predicted success values are near zero, the local solver relies on progress-oriented filtering so that exploratory actions can remain available. Once at least one local option has a sufficiently high predicted terminal-success value, filtering shifts toward success-oriented selection. Thus, p_{gate} controls the transition between early progress-guided search and later success-guided search. Its sensitivity is evaluated in the experimental chapter.

7.10. From Local Options to Global Decision Making

The output of subnet solving is not a final primitive action. It is a set of local option specifications. Each local option contains:

- a first primitive action;
- predicted success $\hat{p}$;
- predicted progress $\hat{g}$;
- predicted cost $\hat{c}$;
- the anchor and subnet context from which it was generated;
- structural descriptors used later for action-template abstraction and policy-library retrieval.

Before each global decision, the system selects active anchors, ensures that the corresponding subnets are available, rebuilds stale subnets when their structural fingerprints have changed, and then collects the approximate non-dominated local options from the active subnets. These candidates are later combined with rule-based actions and candidates retrieved from the generalized policy library.

This design explains why the framework uses subnets plus approximate non-dominated local options rather than pure global MCTS alone. Pure global MCTS repeatedly faces the full discovered graph and full action space as they expand. The subnet-based framework instead transforms local search into reusable structured candidates. These candidates can be cached, compared, abstracted into action templates, and reused across related contexts. This is what makes later generalized policy-library reuse possible.

7.11. Chapter Summary

This chapter introduced local subnet search and approximate non-dominated local option generation. A local subnet is a restricted graph extracted from the planner’s current internal knowledge graph around an anchor node. The strict subnet view defines the local solving domain for budgeted MCTS. Frontier score is used to identify promising boundary nodes for subnet construction, while progress score is used as a state-level proxy reward inside local rollouts. Frontier-directed path expansion does not add executable shortcut edges; it only includes currently known paths toward selected high-frontier nodes in the local solving view. Finally, the ϵ -sweep produces a finite set of local candidates under different budget values, and dominance filtering retains an approximate non-dominated local option set over predicted success, predicted progress, and predicted cost. These local options are passed to the global decision stage, where they are combined with rule-based actions and reusable policy-library candidates.

Action-Template Abstraction and the Generalized Policy Library

Chapter 7 introduced the local subnet solver. Given a strict subnet and a finite MCTS budget, the solver produces a finite set of sampled local candidates and then retains an approximate non-dominated subset over predicted success, predicted progress, and predicted cost. As discussed in Chapter 7, this set should not be called a complete Pareto set: it is not guaranteed to contain all Pareto-optimal solutions over the full feasible decision space. Rather, it is a finite and approximate set of locally non-dominated options generated under restricted subnet search and finite rollout budget (Ehrgott, 2005).

If these local options were stored only as concrete actions applied to concrete internal node identifiers, they would be useful only within the current episode and the current graph instance. For example, the action `move(17)` has no direct meaning in another network instance whose internal node identifiers are different. To make local experience reusable, this chapter introduces *action-template abstraction* and a *generalized policy library*. The objective is not to learn a complete universal policy, but to compress locally useful first-action experience into structured units that are independent of instance-specific node identifiers and can be reused across episodes.

This chapter addresses three questions:

1. How is a concrete local action abstracted into a categorical action template?
2. How is a movement template re-instantiated in a new internal graph?
3. How are templates indexed, retrieved, updated, and forgotten in the generalized policy library?

The mechanism is related in spirit to temporal abstraction and reusable options in sequential decision making (Sutton et al., 1999). However, the objects stored here are not full temporally extended option policies with initiation sets and termination conditions. They are abstract first-action priors, indexed by structural signatures and updated from execution feedback.

8.1. From Local Options to Action Templates

Let a local option produced by the subnet solver be denoted by

$$\omega = (a, \hat{p}, \hat{g}, \hat{c}, \xi), \quad (8.1)$$

where a is the first primitive action proposed by local MCTS, $\hat{p}$ is its predicted terminal-success value, $\hat{g}$ its predicted progress value, $\hat{c}$ its predicted cost, and ξ denotes local context information such as the subnet anchor and structural descriptors. The quantities $\hat{p}$, $\hat{g}$, and $\hat{c}$ were defined in Chapter 7; they are empirical local predictions, not exact values under the hidden true environment.

The abstraction function

$$\Gamma(a, I_t) = \tau \quad (8.2)$$

maps a concrete first action a in internal state I_t to an action template τ . The purpose of Γ is to remove dependence on concrete node identifiers while preserving the semantic role of the action.

For non-movement actions, the template contains only the primitive action type:

$$\tau = (\text{type}), \quad \text{type} \in \{\text{scan}, \text{traffic_sense}, \text{exploit}, \text{confirm_identity}, \text{query_db}\}. \quad (8.3)$$

This is sufficient because these actions are interpreted relative to the current internal location.

For movement actions, a concrete action has the form $\text{move}(u)$, where u is an internal target node. Since the identifier u is not reusable across graph instances, the movement template stores categorical descriptors:

$$\tau = (\text{move}, \sigma_{\text{sel}}, \sigma_{\text{tar}}, \sigma_{\text{goal}}). \quad (8.4)$$

The components are:

- $\sigma_{\text{sel}} \in \mathcal{S}_{\text{sel}}$ is a categorical selector name, such as **db_potential**, **frontier**, **credential**, **info_gain**, **low_cost**, or **lateral**.
- $\sigma_{\text{tar}} \in \mathcal{B}_{\text{role}}$ is a categorical target-role bucket. In the simulated network domain,

$$\mathcal{B}_{\text{role}} = \{A, B, C, DB, O, UNK, ANY\}. \quad (8.5)$$

The value **UNK** denotes an uncertain or weakly classified target, while **ANY** denotes that no target-role restriction is imposed.

- $\sigma_{\text{goal}} \in \mathcal{G}_{\text{move}}$ is a categorical movement-goal label:

$$\mathcal{G}_{\text{move}} = \{\text{db}, \text{cred}, \text{lateral}, \text{ANY}\}. \quad (8.6)$$

It records whether the movement is primarily interpreted as database-directed movement, credential/capability preparation, lateral exploration, or unrestricted movement.

Thus, all template components are enumerated categorical values. A movement template does not store a concrete target identifier. Instead, it stores a rule for how a target should be selected in a future context.

8.2. Movement-Goal and Target-Bucket Abstraction

The target-role bucket of a node u is computed from its current role belief:

$$\text{bucket}(u) = \begin{cases} \arg \max_{r \in \mathcal{R}} b_u(r), & \text{if } \max_{r \in \mathcal{R}} b_u(r) \geq \theta_{\text{role}}, \\ \text{UNK}, & \text{otherwise.} \end{cases} \quad (8.7)$$

Here θ_{role} is a confidence threshold. This rule prevents weak role beliefs from being forced into an overconfident categorical label.

The movement-goal label is computed from the current internal state and the target node:

$$g_{\text{move}}(I_t, u) \in \{\text{db}, \text{cred}, \text{lateral}\}. \quad (8.8)$$

A representative decision rule is

$$g_{\text{move}}(I_t, u) = \begin{cases} \text{db}, & \text{if } d_u^t \geq \theta_{\text{db}} \text{ or } b_u^t(DB) \geq \theta_{\text{db}}, \\ \text{cred}, & \text{if } \hat{\kappa}_t = 0 \text{ and } b_u^t(C) \geq \theta_{\text{cred}}, \\ \text{lateral}, & \text{otherwise.} \end{cases} \quad (8.9)$$

The first case labels moves toward database-related candidates; the second labels moves toward likely credential- or capability-enabling candidates when credentials are not yet available; the final case covers movement that is mainly exploratory, connective, or lateral.

When abstracting a concrete movement action $\text{move}(u)$, the framework therefore stores

$$\Gamma(\text{move}(u), I_t) = (\text{move}, \sigma_{\text{sel}}, \text{bucket}(u), g_{\text{move}}(I_t, u)). \quad (8.10)$$

The selector component σ_{sel} records how the target was selected, while the target bucket and movement-goal components record what kind of target the movement was directed toward.

8.3. Template Instantiation in a New Context

When a library item is retrieved in a new internal state I_t , its action template must be converted back into a concrete primitive action. This process is called *template instantiation*. It is important because the template does not contain a concrete node identifier. Instead, it must select an appropriate target from the current internal graph.

For non-movement templates, instantiation is direct:

$$\text{Inst}((\text{type}), I_t) = \text{type}, \quad (8.11)$$

provided that the corresponding primitive action is applicable in I_t . If the action is not applicable, the template produces no executable candidate.

For movement templates

$$\tau = (\text{move}, \sigma_{\text{sel}}, \sigma_{\text{tar}}, \sigma_{\text{goal}}), \quad (8.12)$$

instantiation proceeds in two stages: candidate filtering and selector evaluation.

8. Action-Template Abstraction and the Generalized Policy Library

First, the planner constructs the candidate set

$$\mathcal{C}_\tau(I_t) = \left\{ u \in \mathcal{N}_t^{\text{cand}} : \chi(I_t, \text{move}(u)) = 1 \wedge R_{\text{tar}}(u, \sigma_{\text{tar}}) \wedge R_{\text{goal}}(I_t, u, \sigma_{\text{goal}}) \right\}. \quad (8.13)$$

Here $\mathcal{N}_t^{\text{cand}}$ is the current set of movement candidates, and χ is the action-applicability predicate defined in the action model. The target-bucket predicate is

$$R_{\text{tar}}(u, \sigma_{\text{tar}}) = \begin{cases} 1, & \text{if } \sigma_{\text{tar}} = \text{ANY}, \\ 1, & \text{if } \text{bucket}(u) = \sigma_{\text{tar}}, \\ 0, & \text{otherwise.} \end{cases} \quad (8.14)$$

The movement-goal predicate is

$$R_{\text{goal}}(I_t, u, \sigma_{\text{goal}}) = \begin{cases} 1, & \text{if } \sigma_{\text{goal}} = \text{ANY}, \\ 1, & \text{if } g_{\text{move}}(I_t, u) = \sigma_{\text{goal}}, \\ 0, & \text{otherwise.} \end{cases} \quad (8.15)$$

Second, the selector name σ_{sel} defines a scoring function over the filtered candidates:

$$S_{\sigma_{\text{sel}}}(u \mid I_t). \quad (8.16)$$

The instantiated target is

$$u^* = \arg \max_{u \in \mathcal{C}_\tau(I_t)} S_{\sigma_{\text{sel}}}(u \mid I_t). \quad (8.17)$$

The resulting concrete action is

$$\text{Inst}(\tau, I_t) = \text{move}(u^*). \quad (8.18)$$

If $\mathcal{C}_\tau(I_t) = \emptyset$, the template cannot be instantiated in the current context and is ignored by the global aggregator.

Representative selector functions include:

$$S_{\text{db.potential}}(u \mid I_t) = d_u^t + \lambda_\rho \rho_{\hat{\ell}_u}^t - \lambda_c \hat{c}_{\hat{\ell}_u}^t, \quad (8.19)$$

$$S_{\text{frontier}}(u \mid I_t) = \phi_{\text{frontier}}(u \mid I_t) + \lambda_\rho \rho_{\hat{\ell}_u}^t - \lambda_c \hat{c}_{\hat{\ell}_u}^t, \quad (8.20)$$

$$S_{\text{credential}}(u \mid I_t) = (1 - \hat{\kappa}_t) b_u^t(C) + \lambda_\rho \rho_{\hat{\ell}_u}^t - \lambda_c \hat{c}_{\hat{\ell}_u}^t, \quad (8.21)$$

$$S_{\text{info.gain}}(u \mid I_t) = \frac{H(b_u^t)}{H(b_u^t) + 1} + \frac{1}{1 + \text{scanCount}(u)} - \lambda_c \hat{c}_{\hat{\ell}_u}^t, \quad (8.22)$$

$$S_{\text{low.cost}}(u \mid I_t) = -\hat{c}_{\hat{\ell}_u}^t + \lambda_\rho \rho_{\hat{\ell}_u}^t, \quad (8.23)$$

$$S_{\text{lateral}}(u \mid I_t) = \lambda_\rho \rho_{\hat{\ell}_u}^t + \lambda_{\text{deg}} \min \left(1, \frac{\text{deg}(u)}{d_0} \right) - \lambda_d d_u^t - \lambda_c \hat{c}_{\hat{\ell}_u}^t. \quad (8.24)$$

Here $\rho_{\hat{\ell}_u}^t$ denotes the current internal reliability or reachability estimate from the current location $\hat{\ell}_t$ to candidate target u , and $\hat{c}_{\hat{\ell}_u}^t$ denotes the estimated movement cost. The term d_u^t denotes the current distance or depth-related estimate used by the selector, depending on the selector context. The coefficients $\lambda_\rho, \lambda_c, \lambda_{\text{deg}}, \lambda_d \geq 0$ are planner parameters controlling the influence of reliability, cost, local degree, and distance penalties. These selector scores do not define new primitive actions; they only rank currently admissible movement targets when instantiating a stored movement template.

These formulas are not intended to define new primitive actions. They define how a stored movement template is evaluated against currently available movement targets. Thus, the selector is a context-dependent target-selection rule, not a pointer to the original target node.

8.4. Structural Signatures and Local Context Indexing

Action templates alone are insufficient for reuse. The same template may be useful in one local context and harmful in another. Therefore, each template is stored together with a structural signature that describes the local situation in which the template was learned.

A signature is a finite tuple of categorical values:

$$\sigma_t \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_m. \quad (8.25)$$

Each component is obtained by discretising a local feature of the current internal state. The library does not store continuous belief vectors or raw graph fragments directly; it stores bucketed structural descriptors.

To avoid defining unused components, each retrieval mode defines only the components that are actually used by that mode. In the coarse retrieval mode, the signature is

$$\sigma_t^{\text{coarse}} = (q_{DB}, q_H, q_{\text{deg}}, q_F, q_\pi, q_{\text{sec}}), \quad (8.26)$$

where

- q_{DB} is a bucketed database-belief value at the current node;
- q_H is a bucketed role-belief entropy value;
- q_{deg} is a bucketed local degree value;
- q_F is a bucketed frontier-score value;
- q_π is a bucketed privilege value;
- q_{sec} is a bucketed security or difficulty value.

For example, a value $x \in [0, 1]$ can be discretised into L levels by

$$q_L(x) = \min(L - 1, \lfloor Lx \rfloor). \quad (8.27)$$

Degree and privilege are discretised by capped integer buckets.

In the hybrid retrieval mode, the signature is instead

$$\sigma_t^{\text{hyb}} = (q_{\text{loc}}, q_{\text{sat}}, q_{\text{phase}}, q_\pi, q_\kappa, q_{\text{curDB}}, q_{\text{rf}}). \quad (8.28)$$

The components have the following meanings:

- q_{loc} is a categorical local-position tier;
- q_{sat} is an exploration-saturation bucket;
- q_{phase} is a directional phase bucket, such as early exploration, credential seeking, or database seeking;
- q_π is the privilege bucket;

8. Action-Template Abstraction and the Generalized Policy Library

- q_κ is the credential flag bucket;
- q_{curDB} is the current-node database-potential bucket;
- q_{rf} is a combined risk/frontier bucket.

The system uses one retrieval mode consistently when storing and retrieving library items. Thus, if the library is configured to use σ^{coarse} , the hybrid components are not part of the key; if it is configured to use σ^{hyb} , the coarse signature is not used as the retrieval key. This avoids the ambiguity of defining a larger signature and then discarding one of its components.

This structural compression is conceptually related to generalized planning, where features and abstract actions are used to represent classes of structurally similar situations rather than single fully instantiated states (Bonet and Geffner, 2018; Bonet et al., 2019). The difference is that this thesis does not compute a complete generalized policy. It stores local action priors indexed by discrete structural signatures.

8.5. Subnet Fingerprints and Cache-Level Structure

In addition to the current-location signature, the framework may compute a node-identifier-independent fingerprint for a subnet. This fingerprint is used for cache-level reuse and staleness checking rather than as the main generalized-policy retrieval key.

Let S be a subnet and let σ_t be the current retrieval signature. A subnet fingerprint can be written abstractly as

$$\Phi_{\text{sub}}(S, \sigma_t) = \text{Hash}(\sigma_t, \mathcal{M}_V(S), \mathcal{M}_E(S)), \quad (8.29)$$

where $\mathcal{M}_V(S)$ is a multiset of bucketed node features and $\mathcal{M}_E(S)$ is a multiset of bucketed edge features. Node features may include database-belief bucket, security bucket, scan-count bucket, degree bucket, handle availability, and stable-identity availability. Edge features may include edge-confidence bucket, movement-success bucket, and movement-cost bucket.

Because these features are stored as sorted multisets and do not contain absolute node identifiers, two subnets with different concrete node names but similar bucketed structure can receive the same or similar fingerprint. This supports structure-level caching, while the generalized policy library itself relies on the signature-template key described above.

8.6. Library Items and Retrieval

A generalized policy-library item is defined as

$$m = (\sigma, \tau, \theta), \quad (8.30)$$

where σ is a structural signature, τ is an action template, and θ is the associated experience record. The experience record contains predicted and observed statistics:

$$\theta = (p_{\text{pred}}, g_{\text{pred}}, c_{\text{pred}}, g_{\text{ewma}}, \rho_c, n, e_{\text{use}}, e_{\text{upd}}). \quad (8.31)$$

8.7. Online Update, Successful Replay, and Slow Forgetting

Here p_{pred} is the predicted success value, g_{pred} the predicted progress value, c_{pred} the predicted cost, g_{ewma} the exponentially weighted moving average of observed progress, ρ_c the moving average of observed cost ratio, n the effective sample count, and e_{use} and e_{upd} the most recent use and update episode indices.

The logical identity of a library item is determined by the pair

$$(\sigma, \tau). \quad (8.32)$$

When new experience is written to the library with the same signature and template, it updates the existing item rather than creating a long-lived duplicate.

At retrieval time, the current signature σ_t is compared with stored signatures using Hamming distance:

$$d_H(\sigma_t, \sigma_m) = \sum_i \mathbb{I}[\sigma_t^{(i)} \neq \sigma_m^{(i)}]. \quad (8.33)$$

The retrieved set is

$$R(\sigma_t) = \text{TopN}_{m \in \mathcal{M}}(-d_H(\sigma_t, \sigma_m), p_m, g_m, e_m^{\text{use}}), \quad (8.34)$$

where $\mathcal{M}$ is the current library and the ranking favours small signature distance, high predicted success, high observed progress, and more recent use.

For each retrieved item, the stored template τ_m is instantiated in the current state using the instantiation rule in Section 8.3. If the template cannot be instantiated, for example because no current movement candidate satisfies its target-bucket and movement-goal predicates, the item does not produce an executable action candidate. This prevents the library from forcing outdated or context-incompatible actions into the global aggregator.

8.7. Online Update, Successful Replay, and Slow Forgetting

The generalized policy library is not a static database. It is updated continuously from execution feedback.

Let

$$m = (\sigma, \tau, \theta) \quad (8.35)$$

be an existing item. When the planner executes an action corresponding to (σ, τ) , the item is updated from observed feedback. If the item does not yet exist, it is initialised from the current local prediction and observation. If it already exists, the effective sample count is increased and the predicted or observed statistics are smoothed.

For predicted success, the update can be written as

$$p'_{\text{pred}} = (1 - \alpha_p)p_{\text{pred}} + \alpha_p \cdot \text{hit}, \quad (8.36)$$

where $\text{hit} \in \{0, 1\}$ is direct terminal-success feedback when available. Since terminal success is sparse, this update is mainly informative for query-related or late-stage actions.

For observed progress and cost ratio, the library uses exponentially weighted moving averages:

$$g'_{\text{ewma}} = (1 - \beta_g)g_{\text{ewma}} + \beta_g g_{\text{obs}}, \quad (8.37)$$

8. Action-Template Abstraction and the Generalized Policy Library

and

$$\rho'_c = (1 - \beta_c)\rho_c + \beta_c \frac{c_{\text{actual}}}{c_{\text{pred}} + \epsilon_c}. \quad (8.38)$$

Here $\epsilon_c > 0$ prevents division by zero.

The library receives positive information from successful replay. At the end of a successful episode, the planner does not store the entire trajectory verbatim. Instead, it extracts a key subsequence:

$$\Xi^* \subseteq \Xi, \quad (8.39)$$

where Ξ is the full executed trajectory. Key steps include successful or diagnostic `query_db` actions, exploits that produced privilege or credentials, information actions that opened new structure, and movements corresponding to first entry into strategically important phases. This prevents the library from being dominated by low-value intermediate actions.

The library also receives negative or weak feedback online. If a retrieved general-policy candidate is executed and produces no structural discovery, no capability gain, no progress increase, and no terminal success, the associated item is updated with weak progress or stagnation evidence. Thus, the library learns not only from successful episodes but also from poor general guidance.

Finally, unused items are slowly forgotten. For an unused item, the update may take the form

$$p \leftarrow 0.5 + \gamma_p(p - 0.5), \quad g \leftarrow \gamma_g g, \quad \rho_c \leftarrow 1 + \gamma_c(\rho_c - 1), \quad n \leftarrow \gamma_n n, \quad (8.40)$$

where $\gamma_p, \gamma_g, \gamma_c, \gamma_n \in (0, 1)$ are decay parameters. Items whose effective sample count becomes too small or whose quality remains poor may be pruned. This mechanism prevents early low-quality experience from occupying library capacity indefinitely.

8.8. State-Bucket-to-Template Strategy Abstraction

The generalized policy library should not be interpreted as a memory of concrete trajectories or node-specific actions. Its role is to store reusable local strategy priors at the level of abstract structural states and action templates.

Given an internal state I_t , the retrieval-signature function maps the current local context to a structural state bucket:

$$\sigma_t = \Sigma(I_t). \quad (8.41)$$

This signature does not encode the full internal graph or the concrete identities of all nodes. Instead, it summarises planning-relevant properties of the current local context, such as role-belief patterns, database-related evidence, capability state, frontier structure, and local progress indicators.

A concrete action a_t is similarly abstracted into an action template:

$$\tau_t = \Gamma(a_t, I_t). \quad (8.42)$$

For non-movement actions such as `scan`, `traffic_sense`, `exploit`, `confirm_identity`, and `query_db`, the template may simply record the action type. For movement actions, however, the template does not

8.9. What the Generalized Policy Library Provides

store the concrete target node. Instead, it records a selector, a target bucket, and a movement goal, such as

$$\text{move}(\text{InfoGain}, \text{DB}, \text{db}), \quad (8.43)$$

which means moving toward a database-relevant target selected by an information-gain criterion.

The library therefore stores mappings of the form

$$(\sigma, \tau) \mapsto (\hat{p}, \hat{g}, \hat{c}, n), \quad (8.44)$$

where σ is a structural state bucket, τ is an abstract action template, $\hat{p}$ is the predicted success value, $\hat{g}$ is the predicted progress value, $\hat{c}$ is the predicted cost, and n is the number of times this entry has been observed or updated.

This gives the policy library a predicate-level interpretation. Each entry can be read as a local rule of the following form:

In this class of local belief states, this type of abstract action is likely to be useful.

For example, an under-observed local context may map to `scan` or `traffic_sense`; a context with possible capability gain may map to `exploit`; a context with database-related evidence may map to `move(InfoGain, DB, db)` or `move(DB_seeking, DB, db)`; and a late-stage context with strong database evidence and sufficient capability may map to `query_db`.

Thus, the generalized policy library performs a state-bucket-to-template abstraction. It converts concrete execution experience into reusable local first-action priors. This is weaker than learning a complete generalized policy, but stronger than simply caching previous actions. It allows experience from one episode to guide planning in a new episode whenever the new local context falls into a similar structural bucket.

8.9. What the Generalized Policy Library Provides

The generalized policy library stores experience units of the form:

$$\text{local structural signature} + \text{action template} + \text{success/progress/cost statistics}. \quad (8.45)$$

This means that the library reuses local decision priors, not concrete node identifiers and not full policies.

The mechanism provides three advantages. First, it allows experience from previous subnet searches to influence later decisions without rerunning the same local search from scratch. Second, it makes experience less dependent on instance-specific node names. Third, it allows positive and negative feedback to gradually reshape the stored values.

At the same time, the mechanism has clear limitations. The structural signature is a coarse categorical abstraction, so different local contexts can collide in the same bucket. The action template represents only a first-action prior, not a full multi-step policy. Selector-based instantiation can fail if no suitable current target satisfies the stored categorical constraints. Therefore, the generalized policy library should not

8. *Action-Template Abstraction and the Generalized Policy Library*

be interpreted as a complete generalized planner. It is better understood as a transferable, updateable, and forgettable library of local action priors.

This interpretation is consistent with the experimental results reported later. Same-family transfer is expected to be more reliable because local structural signatures and action-template meanings are more likely to match. Cross-type transfer is more difficult because the same categorical signature may correspond to different graph dynamics or role distributions in a different network family.

8.10. Chapter Summary

This chapter defined the action-template abstraction and generalized policy library. The local solver outputs an approximate non-dominated set of local options, not a complete Pareto set. Each concrete action is abstracted into a categorical template; for movement actions, the template stores a selector name, target-role bucket, and movement-goal label rather than a concrete node identifier. The chapter also defined how movement templates are instantiated in a new context by filtering current movement candidates and applying a selector-specific scoring function. Structural signatures were defined only in terms of components actually used by the selected retrieval mode, avoiding unused signature dimensions. Finally, the chapter described how library items are retrieved, updated through successful replay and online feedback, and slowly forgotten over time. The resulting mechanism is a reusable library of local first-action priors rather than a complete generalized policy.

Global Decision Making and Triggered Rebuilding

The previous chapters introduced the local subnet solver and the mechanism for action-template abstraction and generalized policy reuse. However, neither the local options produced by subnet solving nor the template-based candidates retrieved from the generalized policy library are themselves necessarily the final executed action. A complete online planner still requires a global decision layer that compares rule-based candidates, local subnet candidates, and library-based candidates under the current internal state, and that updates its reliance on these sources in response to execution feedback.

At a high level, the global decision layer implements the following closed loop:

$$\begin{aligned} \text{local solving} &\rightarrow \text{candidate instantiation} \rightarrow \text{global aggregation} \\ &\rightarrow \text{execution feedback} \rightarrow \text{selective rebuilding.} \end{aligned} \tag{9.1}$$

This chapter defines this loop. Its purpose is not to introduce a new action model; the primitive action semantics were defined in Chapter 4, and the online interaction loop was described in Chapter 6. Instead, this chapter explains how the planner combines multiple candidate sources, chooses one executable primitive action, monitors progress, and rebuilds local subnets when previous local predictions appear stale.

Conceptually, this layer is related to online search under uncertainty, hierarchical decomposition, and adaptive approximate planning (Kaelbling et al., 1998; Silver and Veness, 2010; Ye et al., 2017; Bercher et al., 2019). In the present thesis, however, its specific role is to integrate strict-subnet local solving, reusable action templates, rule-based fallback candidates, online feedback, and triggered rebuilding into a single decision process.

Figure 9.1 summarises the overall online planning pipeline of the proposed framework, including internal-state updating, local subnet solving, generalized policy retrieval, global action aggregation, execution feedback, and trigger-based rebuilding.

9.1. The Global Decision Loop

As discussed in Chapter 7, the local subnet solver generates approximate non-dominated local options under a strict subnet view and finite MCTS budget. As discussed in Chapter 8, the generalized policy

9. Global Decision Making and Triggered Rebuilding

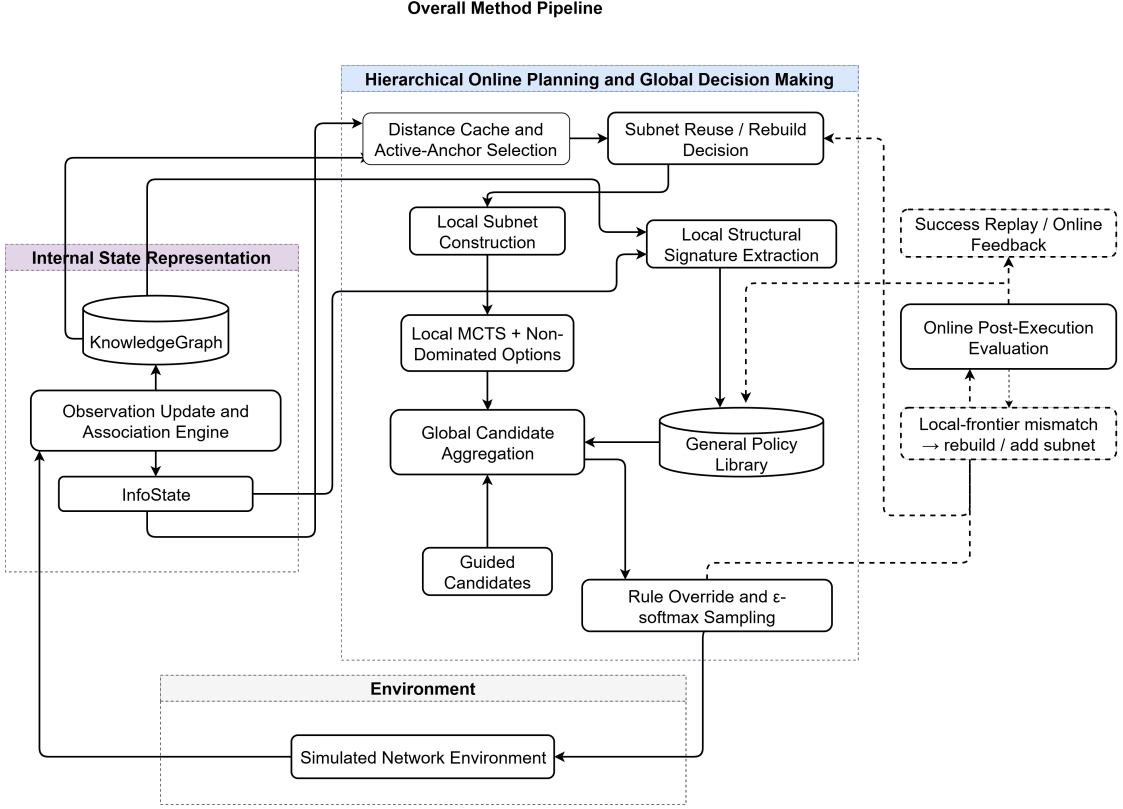


Figure 9.1.: Overall online planning pipeline of the proposed framework.

library retrieves action templates indexed by structural signatures. The global decision layer combines these with rule-based fallback candidates and chooses a single primitive action to execute.

Algorithm 3 on page 87 summarises one decision step of the complete planner. The algorithm should be read as the operational integration of earlier components, not as a separate new model. The local subnet solver corresponds to Chapter 7; the template-instantiation procedure corresponds to Chapter 8; and the primitive action execution and internal update are those defined in Chapter 6.

A single step of global decision making can be written as the following multi-stage mapping:

$$\begin{aligned}
 I_t &\xrightarrow{\text{active anchor selection}} \mathcal{A}_t^{\text{anch}} \\
 &\xrightarrow{\text{subnet solving or cache reuse}} \Omega_t^{\text{local}} \\
 &\xrightarrow{\text{candidate instantiation}} \mathcal{C}_t \\
 &\xrightarrow{\text{selection and execution}} (a_t, o_{t+1}, c_t) \\
 &\xrightarrow{\text{online update and possible rebuild}} I_{t+1}.
 \end{aligned} \tag{9.2}$$

Here $\mathcal{A}_t^{\text{anch}}$ is the active anchor set, Ω_t^{local} is the set of local options collected from active subnets, and $\mathcal{C}_t$ is the set of executable candidates entering global comparison.

9.2. Candidate Instantiation and Unified Candidate Sets

The active anchors used here are the same concept defined in Section 7.5. They should not be understood as coming only from a permanent subnet store. Instead, the active anchor set is selected from both newly generated anchor candidates and anchors associated with cached subnets. For each active anchor, the planner either reuses a cached subnet option set if the corresponding structural fingerprint is still valid, or rebuilds the strict subnet and recomputes its local options.

Let the j -th active anchor be a_j , and let its current subnet fingerprint be $\Phi_{\text{sub}}(a_j, I_t)$. The subnet maintenance logic can be written as

$$\text{EnsureSubnet}(a_j, t) = \begin{cases} \Omega_{a_j}^{\text{cached}}, & \text{if } \Phi_{\text{sub}}(a_j, I_t) = \Phi_{\text{sub}}^{\text{cache}}(a_j), \\ \Omega_{a_j}^{\text{new}}, & \text{otherwise.} \end{cases} \quad (9.3)$$

Thus, each decision step does not replan the entire problem from scratch. Instead, it selectively combines cached local experience, newly solved subnet options, library-based templates, and rule-based fallback candidates.

9.2. Candidate Instantiation and Unified Candidate Sets

The global aggregator compares executable candidates, not raw local options or raw policy-library templates. This section defines how instantiated candidates are derived from the three sources used by the planner.

At time t , the complete candidate set is

$$\mathcal{C}_t = \mathcal{C}_t^{\text{rule}} \cup \mathcal{C}_t^{\text{local}} \cup \mathcal{C}_t^{\text{general}}, \quad (9.4)$$

where $\mathcal{C}_t^{\text{rule}}$ contains rule-based fallback candidates, $\mathcal{C}_t^{\text{local}}$ contains candidates instantiated from local subnet options, and $\mathcal{C}_t^{\text{general}}$ contains candidates instantiated from generalized policy-library items.

A candidate is represented as

$$c = (a, \hat{p}, \hat{g}, \hat{r}, \hat{d}, \hat{c}, \text{src}, \text{meta}), \quad (9.5)$$

where a is an executable primitive action, $\hat{p}$ is predicted terminal-success value, $\hat{g}$ is predicted progress value, $\hat{r}$ is a risk estimate, $\hat{d}$ is a distance or reachability estimate, $\hat{c}$ is predicted execution cost, src records the source of the candidate, and meta stores source-specific information needed for online feedback.

9.2.1. Instantiation from Local Options

A local option has the form

$$\omega = (a_\omega, \hat{p}_\omega, \hat{g}_\omega, \hat{c}_\omega, a_{\text{anch}}, \xi), \quad (9.6)$$

where a_ω is the first primitive action proposed by the local subnet solver, $\hat{p}_\omega$, $\hat{g}_\omega$, and $\hat{c}_\omega$ are empirical predictions from local MCTS, a_{anch} is the subnet anchor, and ξ contains local context descriptors.

9. Global Decision Making and Triggered Rebuilding

The instantiation rule for a local option is direct. If the first action a_ω is applicable in the current internal state, then it becomes an executable candidate:

$$\text{Inst}_{\text{local}}(\omega, I_t) = \begin{cases} (a_\omega, \hat{p}_\omega, \hat{g}_\omega, \hat{r}_\omega, \hat{d}_\omega, \hat{c}_\omega, \text{local}, \text{meta}_\omega), & \text{if } \chi(I_t, a_\omega) = 1, \\ \perp, & \text{otherwise.} \end{cases} \quad (9.7)$$

Here χ is the action-applicability predicate from the action model. The risk and distance terms are computed from the current internal graph and the candidate action. If the option's first action is no longer applicable, for example because the relevant edge was blocked or the target was merged into another node, the option produces no candidate.

9.2.2. Instantiation from the Generalized Policy Library

A retrieved library item has the form

$$m = (\sigma, \tau, \theta), \quad (9.8)$$

where σ is the stored structural signature, τ is an action template, and θ is the associated experience record. Unlike a local option, a template may not directly contain a concrete executable action. It must first be instantiated in the current internal state using the template-instantiation rule defined in Section 8.3.

For a non-movement template, instantiation simply checks whether the corresponding primitive action is applicable. For a movement template, instantiation constructs a candidate target set, filters it by the stored target-role bucket and movement-goal label, and then applies the stored selector function to choose a concrete movement target. This process was defined in Section 8.3; here it is used to turn retrieved library items into executable candidates:

$$\text{Inst}_{\text{general}}(m, I_t) = \begin{cases} (a_m, \hat{p}_m, \hat{g}_m, \hat{r}_m, \hat{d}_m, \hat{c}_m, \text{general}, \text{meta}_m), & \text{if } \text{Inst}(\tau, I_t) = a_m, \\ \perp, & \text{if } \tau \text{ cannot be instantiated in } I_t. \end{cases} \quad (9.9)$$

If no current movement target satisfies the template's categorical constraints, the item is ignored for the current decision step. This prevents structurally incompatible library items from forcing inappropriate actions into the global comparison.

9.2.3. Rule-Based Fallback Candidates

Rule-based candidates are generated directly from the current internal state. They are not intended to replace local search or generalized policy retrieval. Their role is to ensure that a small number of high-confidence or safety-critical primitive actions remain visible to the global aggregator even when local options are stale, too narrow, or temporarily dominated by noisy predicted scores.

For example, if the current node has high credential-relevant belief and credentials have not yet been obtained, `exploit` may be added as a rule-based candidate. If the current node has high database belief and the required capability conditions are satisfied, `query_db` may be added as a rule-based candidate. Information-gathering actions such as `scan`, `traffic_sense`, or `confirm_identity` may also be added when the current node is under-observed or identity remains ambiguous.

9.3. Global Aggregation Score

Thus, rule-based candidates are fallback candidates, not the only source of exploration. Local subnet options can themselves be exploratory, and generalized templates may also instantiate to exploratory actions. The rule-based layer simply guarantees that basic information-gathering, capability-critical, and terminal-verification actions are not lost due to local scoring noise.

9.3. Global Aggregation Score

The system does not perform a hard switch between local and general sources. Instead, all instantiated candidates are mapped into a common comparison space and evaluated jointly.

Let candidate $c \in \mathcal{C}_t$ have predicted statistics

$$\left(\hat{p}(c), \hat{g}(c), \hat{r}(c), \hat{d}(c), \hat{c}(c) \right), \quad (9.10)$$

where $\hat{p}(c)$ is the predicted terminal-success value, $\hat{g}(c)$ is the predicted progress value, $\hat{r}(c)$ is a risk estimate, $\hat{d}(c)$ is a distance or reachability-cost estimate, and $\hat{c}(c)$ is the predicted execution cost. These quantities may come from different sources. For local candidates, $\hat{p}$, $\hat{g}$, and $\hat{c}$ are inherited from local subnet MCTS estimates; for generalized candidates, they are taken from the policy-library experience record; for rule-based candidates, they are assigned from the current internal state and action type.

The base global score is

$$S_{\text{base}}(c \mid I_t) = w_p \hat{p}(c) + w_g \hat{g}(c) - w_r \hat{r}(c) - w_d \hat{d}(c) - w_c \hat{c}(c). \quad (9.11)$$

Here $w_p, w_g, w_r, w_d, w_c \geq 0$ are aggregation weights. The positive weights w_p and w_g increase the score of candidates with higher predicted terminal success and higher predicted progress. The weights w_r , w_d , and w_c penalise risk, distance or reachability cost, and execution cost respectively. These weights are planner parameters, not part of the formal problem definition. They determine how the global decision layer balances terminal success, intermediate progress, risk avoidance, reachability, and budget efficiency.

The final score augments the base score with structured adjustment terms:

$$\begin{aligned} \text{Score}(c \mid I_t) &= S_{\text{base}}(c \mid I_t) + \Delta_{\text{src}}(c) + \Delta_{\text{stage}}(c, I_t) \\ &\quad - L_{\text{loop}}(c, I_t) + B_{\text{novel}}(c, I_t). \end{aligned} \quad (9.12)$$

The term $\Delta_{\text{src}}(c)$ is a source-prior adjustment. It represents a small preference or penalty based on whether the candidate comes from local subnet solving, the generalized policy library, or the rule-based fallback layer. In the current framework, generalized candidates receive a mild penalty:

$$\Delta_{\text{src}}(c) = \begin{cases} -\lambda_{\text{gen}}, & c \in \mathcal{C}_t^{\text{general}}, \\ 0, & c \in \mathcal{C}_t^{\text{local}} \cup \mathcal{C}_t^{\text{rule}}, \end{cases} \quad (9.13)$$

where $\lambda_{\text{gen}} \geq 0$ is a planner parameter controlling how strongly generalized candidates are down-weighted relative to current-instance local and rule-based candidates. This does not mean that generalized library candidates are assumed to be unreliable. Rather, it prevents early over-reliance on cross-episode priors when the current internal graph is still incomplete.

9. Global Decision Making and Triggered Rebuilding

The term $\Delta_{\text{stage}}(c, I_t)$ is a stage-dependent adjustment. It captures corrections that depend on the current capability state, credential state, and estimated planning phase. For example, Δ_{stage} may increase the score of **exploit** when credentials are unavailable and the current node has high credential-relevant belief. It may also increase the score of **query_db** when the current node has high database belief and the required capability conditions are satisfied. Conversely, it may decrease the score of premature movement toward database-like nodes if required credentials are still missing. Thus, Δ_{stage} encodes high-level phase information that is not fully captured by the linear base score.

For any candidate c , let $u(c)$ denote the node whose repetition or novelty is evaluated. If c is a movement candidate, then $u(c)$ is its target node. If c is a non-movement action such as **scan**, **traffic_sense**, **exploit**, **confirm_identity**, or **query_db**, then $u(c) = \hat{\ell}_t$, the current internal location.

The term $L_{\text{loop}}(c, I_t)$ is a loop penalty. It discourages repeated low-value revisits or repeated actions that have recently failed to produce progress. For a movement candidate whose target node is u , one representative form is

$$L_{\text{loop}}(c, I_t) = \lambda_L \cdot \mathbb{I}[u(c) \in R_K(t)] + \lambda_V \frac{\text{visitCount}(u(c))}{1 + \text{visitCount}(u(c))}. \quad (9.14)$$

Here $R_K(t)$ is the set of internal nodes visited during the last K steps, $\text{visitCount}(u)$ is the number of previous visits to node u , and $\lambda_L, \lambda_V \geq 0$ are planner parameters controlling the strength of recency-based and frequency-based loop penalties. For non-movement actions, this penalty is zero unless the action is a repeated information-gathering action at the same node.

The term $B_{\text{novel}}(c, I_t)$ is a novelty bias. It gives a small bonus to candidates that are likely to reveal new local information. For a movement candidate targeting node u , or an information-gathering action applied at node u , a representative form is

$$B_{\text{novel}}(c, I_t) = \lambda_N \frac{1}{1 + \text{visitCount}(u(c))} + \lambda_S \frac{1}{1 + \text{scanCount}(u(c))}. \quad (9.15)$$

Here $\text{scanCount}(u)$ is the number of previous scan operations associated with node u , and $\lambda_N, \lambda_S \geq 0$ control the strength of visit-based and scan-based novelty bonuses. If a candidate has no explicit target node, u is taken to be the current internal location $\hat{\ell}_t$.

The loop penalty and novelty bias are adjustment terms in the global scoring function. They do not introduce new primitive actions and do not change the environment dynamics. Their role is to make action selection less likely to waste budget on repeated low-value revisits and more likely to preserve useful information-gathering behaviour under partial observability.

The global score is therefore not merely a fixed linear scalarisation. It is a linear base score over predicted success, progress, risk, distance, and cost, augmented by source priors, stage-dependent corrections, loop penalties, and novelty bias. This is consistent with the view that sequential decision making under multiple competing criteria often requires adaptive trade-off management rather than a single universal scalar objective (Rojers et al., 2013; Ehrgott, 2005).

9.4. Stochastic Selection and Rule-Based Overrides

Although all candidates receive a common score, the system does not always select the top-scoring candidate deterministically. Instead, it combines score-based ranking with stochastic selection in order to avoid premature lock-in when local predictions are uncertain.

Let the score of candidate c_i be $s_i = \text{Score}(c_i \mid I_t)$. The softmax sampling probability is

$$\Pr(c_i \mid I_t) = \frac{\exp(s_i/T_t)}{\sum_j \exp(s_j/T_t)}, \quad (9.16)$$

where $T_t > 0$ is a temperature parameter. A higher temperature makes the distribution flatter and increases the chance of selecting a lower-scoring candidate. In addition, with probability ϵ_t , the system samples from a broader top- k candidate pool rather than selecting the current best candidate. This ϵ -mixing mechanism should not be understood as a separate exploratory action source. It is a selection-level mechanism applied to the candidate set $\mathcal{C}_t$, which already contains local, library-based, and rule-based candidates.

The exploration parameters can be adjusted when the planner detects stagnation. If there are several consecutive steps without a progress event, the planner can increase ϵ_t , increase T_t , or widen the sampled top- k pool. This makes the action-selection process responsive to local stagnation rather than tied to a fixed exploration schedule.

In addition to stochastic selection, the framework retains a small rule-based override mechanism. This mechanism applies only in high-confidence states where an action is structurally obvious, such as querying a highly likely database node when capability conditions are satisfied, or exploiting a likely credential-relevant node when credentials are still missing. The idea is related in spirit to move-ordering and forcing heuristics in game-tree search, such as the killer heuristic, where highly promising moves are kept visible or prioritised so that search does not miss them due to ordering or scoring noise (Akl and Newborn, 1977). In this thesis, the rule layer is not claimed as a novel search principle; it is a practical safety mechanism for high-confidence terminal or capability-critical states.

Let a_t^{agg} be the action selected by the global aggregator. The final selected action is

$$a_t = \begin{cases} a_t^{\text{rule}}, & \text{if } \rho(I_t) = 1, \\ a_t^{\text{agg}}, & \text{otherwise,} \end{cases} \quad (9.17)$$

where $\rho(I_t)$ is the rule-trigger condition and a_t^{rule} is the corresponding high-confidence rule action. This mechanism does not replace local search or policy-library reuse. It only prevents a small number of obvious actions from being missed when the statistical score is noisy.

9.5. Progress Monitoring and Anti-Stagnation Mechanisms

Recall from Chapter 4 that the internal state is

$$I_t = (\hat{G}_t, \hat{\ell}_t, \hat{\pi}_t, \hat{\kappa}_t, B_t, h_t), \quad (9.18)$$

9. Global Decision Making and Triggered Rebuilding

where $\hat{G}_t$ is the internal knowledge graph, $\hat{\ell}_t$ is the current internal location, $\hat{\pi}_t$ is the privilege estimate, $\hat{\kappa}_t$ is the credential flag, B_t is the remaining budget, and h_t is the internal success flag. The progress function $g(I_t)$ was introduced in Chapter 4 and defined in detail in Section 7.8 as a belief-state potential used to provide dense local guidance when terminal success is sparse.

In a partially observable environment, successful execution of a primitive action is not necessarily meaningful progress toward the task objective. For example, a `move` action may succeed in reaching a node while still failing to reveal structure, improve capability, or increase future success potential. The main loop therefore defines a progress-event predicate.

Let

$$\Delta \text{priv}_t = \text{priv}(I_{t+1}) - \text{priv}(I_t), \quad (9.19)$$

$$\Delta \hat{\kappa}_t = \hat{\kappa}_{t+1} - \hat{\kappa}_t, \quad (9.20)$$

$$\Delta |\hat{V}|_t = |\hat{V}_{t+1}| - |\hat{V}_t|, \quad (9.21)$$

$$\Delta g_t = g(I_{t+1}) - g(I_t). \quad (9.22)$$

Here $\text{priv}(I_t)$ denotes the current privilege estimate in internal state I_t , $\hat{\kappa}_t$ is the credential flag, $|\hat{V}_t|$ is the number of internal nodes in the knowledge graph, and $g(I_t)$ is the progress potential. If edge discovery is used explicitly, one can also define

$$\Delta |\hat{E}|_t = |\hat{E}_{t+1}| - |\hat{E}_t|. \quad (9.23)$$

The progress-event predicate is then

$$\begin{aligned} \text{ProgEvt}_t = \mathbb{I} \Big[& h_{t+1} = 1 \vee \Delta \text{priv}_t > 0 \vee \Delta \hat{\kappa}_t > 0 \\ & \vee \Delta |\hat{V}|_t > 0 \vee \Delta |\hat{E}|_t > 0 \\ & \vee g(I_{t+1}) > \max\{g(I_t), g_{\max}\} + \varepsilon \Big], \end{aligned} \quad (9.24)$$

where $g_{\max}$ is the best progress value observed so far in the episode and $\varepsilon > 0$ is a small tolerance. The condition $h_{t+1} = 1$ already covers successful `query_db`, so it is not necessary to include a separate duplicate query-success term.

The main loop uses this predicate to maintain variables such as `best_prog`, `no_prog_steps`, `no_prog_moves`, and `last_progress_step`. If no progress event occurs, `no_prog_steps` is incremented. If the action was a movement action and no progress event occurs, `no_prog_moves` is also incremented. These counters affect later decisions by increasing stochastic exploration and by strengthening loop penalties or novelty bias terms in the global score.

This mechanism has two functions. First, it provides the global decision layer with a dense diagnostic signal of whether recent action choices are useful, instead of relying only on rare terminal success. Second, together with node-level visit statistics, it supports anti-loop and anti-stagnation behaviour. The mechanism does not alter the external task definition. It only helps the planner detect when budget is being spent without improving the internal conditions for future success.

9.6. Online Feedback to Local and General Sources

After a global action has been executed, the framework updates not only the internal state but also the source-specific statistics associated with the selected candidate. If the selected candidate originates from a local option, then the local option statistics are updated using the realised hit signal, realised cost, and observed progress. If the selected candidate originates from the generalized policy library, then feedback is written back to the corresponding library item.

Here, a failure or stall is not limited to environment-level action failure. It refers to an executed step that produces none of the following: terminal success, privilege gain, credential gain, structural discovery, or meaningful progress increase. Using the notation in Section 9.5, this can be written as

$$\text{FailOrStall}_t = 1 - \text{ProgEvt}_t. \quad (9.25)$$

This definition is intentionally tied to progress monitoring: an action may succeed operationally but still count as a stall if it does not improve the internal state in any task-relevant way.

Thus, the global decision layer is not merely a selector of actions. It also evaluates which candidate source is reliable in the current instance. If local options repeatedly exhibit a mismatch between predicted and realised outcomes, this provides evidence of local model mismatch. If generalized candidates repeatedly fail or stall in the current structural context, the corresponding generalized items are down-weighted. In this way, the subnet solver, generalized library, and global decision layer form a cross-level feedback loop rather than a one-way pipeline.

9.7. Trigger 2: Rebuilding Driven by Local Mismatch

Because the knowledge graph evolves continually through object discovery, identity merging, edge-posterior updates, and belief revision, a previously constructed subnet and its associated local option set may become stale. For this reason, the framework uses triggered rebuilding at the global stage. Its purpose is to add or rebuild subnets when existing local experience appears inconsistent with recent execution feedback.

For local options, the local library maintains three mismatch streaks: success-prediction mismatch, cost-prediction mismatch, and progress-prediction mismatch. Let a local option ω have predicted values $(\hat{p}_\omega, \hat{g}_\omega, \hat{c}_\omega)$. After execution, define observed quantities p_t^{obs} , g_t^{obs} , and c_t^{obs} . A mismatch event can be defined as

$$M_p(\omega, t) = \mathbb{I}\left[p_t^{\text{obs}} < \hat{p}_\omega - \delta_p\right], \quad (9.26)$$

$$M_g(\omega, t) = \mathbb{I}\left[g_t^{\text{obs}} < \hat{g}_\omega - \delta_g\right], \quad (9.27)$$

$$M_c(\omega, t) = \mathbb{I}\left[c_t^{\text{obs}} > \hat{c}_\omega + \delta_c\right], \quad (9.28)$$

where $\delta_p, \delta_g, \delta_c$ are tolerance parameters. The corresponding streaks are incremented when mismatch occurs and reset or weakened when the prediction is consistent with the observation.

9. Global Decision Making and Triggered Rebuilding

Trigger 2 is activated when any mismatch streak exceeds a threshold:

$$\text{Trigger2}(\omega) = \mathbb{I}[\text{streak}_p(\omega) \geq K_{\text{trig}} \vee \text{streak}_c(\omega) \geq K_{\text{trig}} \vee \text{streak}_g(\omega) \geq K_{\text{trig}}]. \quad (9.29)$$

In the implementation, these tolerances and streak lengths correspond to parameters such as `trigger2.delta.p`, `trigger2.delta.c`, `trigger2.delta.g`, and `trigger2.streak`.

Even if no individual local option has triggered Trigger 2, the system may still request additional subnet construction or rebuilding from global-level signals. Let $\hat{h}_t$ denote the rolling hit rate, and let $p_{\text{frontier_best}}$ denote the best current frontier-level success prediction. The rebuild condition can be written as

$$\text{NeedAdd}_t = \mathbb{I}[|\Omega_t^{\text{active}}| < M_{\min} \vee \text{Trigger2} = 1 \vee \hat{h}_t < \gamma p_{\text{frontier_best}} \vee \text{ForceAdd}_t], \quad (9.30)$$

where $M_{\min}$ is the minimum desired number of active local options, $\gamma \in (0, 1)$ is a tolerance ratio, and ForceAdd_t represents explicit initialisation or control-phase rebuilding. Whenever $\text{NeedAdd}_t = 1$, the planner selects anchors again, builds new strict subnets from the latest internal knowledge graph, and recomputes approximate non-dominated local option sets.

The significance of this mechanism is that local subnets and historical local options are treated as approximations relative to the current internal model, not as permanent truths. As edge posteriors, role beliefs, identity associations, and local graph structure evolve, a previously useful local option may become stale. Triggered rebuilding is the mechanism by which the planner balances reuse of local experience with the need to refresh the local solving domain.

9.8. Chapter Summary

This chapter defined the global decision layer of the proposed framework. It explained how active anchors are selected from both newly generated candidates and cached subnet anchors, how local options and generalized policy-library items are instantiated into executable candidates, how rule-based fallback candidates are added, and how all candidates are compared using a common scoring function. It also defined stochastic selection, rule-based overrides, progress monitoring, loop penalties, novelty biases, online feedback, and Trigger-2-based rebuilding.

The global decision layer is therefore not a binary selector between local and generalized guidance. It is a unified online control layer that integrates local subnet candidates, generalized policy-library candidates, and rule-based fallback candidates within a single revisable decision process. Its main role is to connect local solving, cross-episode experience reuse, execution feedback, and adaptive rebuilding into one continually updated planning loop.

Algorithm 3: One Decision Step of the Subnet-Based Planner with Policy Reuse

Input: current internal state I_t ; subnet cache $\mathcal{S}$; generalized policy library $\mathcal{G}$; planner parameters Θ

Output: primitive action a_t

```

1 BUILDDISTCACHE( $I_t$ )
2  $\mathcal{A}_t^{\text{anch}} \leftarrow \text{CHOOSEACTIVEANCHORS}(I_t, \mathcal{S}, \Theta)$ 
3  $\Omega_t^{\text{local}} \leftarrow []$ 
4 foreach  $a \in \mathcal{A}_t^{\text{anch}}$  do
5   if REUSABLESUBNET( $a, I_t, \mathcal{S}$ ) then
6      $\Omega_t^a \leftarrow \text{LOADCACHEDOPTIONS}(a, \mathcal{S})$ 
7   else
8      $S(a, t) \leftarrow \text{BUILDSTRICTSUBNET}(I_t, a, \Theta)$ 
9      $\hat{G}_t^{(a)} \leftarrow \text{MAKESTRICTSUBNETVIEW}(I_t, S(a, t))$ 
10     $\Omega_t^a \leftarrow []$ 
11    foreach  $B_k \in \text{BUDGETGRID}(I_t, \Theta)$  do
12       $(a_k, \hat{p}_k, \hat{g}_k, \hat{c}_k) \leftarrow \text{BUDGETEDMCTS}(\hat{G}_t^{(a)}, I_t, B_k, \Theta)$ 
13       $\omega_k \leftarrow \text{MAKELOCALOPTION}(a_k, \hat{p}_k, \hat{g}_k, \hat{c}_k, a, I_t)$ 
14      append  $\omega_k$  to  $\Omega_t^a$ 
15    if  $\max_{\omega \in \Omega_t^a} \hat{p}(\omega) > \Theta.p\_gate$  then
16       $\Omega_t^a \leftarrow \text{NONDOMINATEDPRUNE}(\Omega_t^a, \text{mode} = \hat{p})$ 
17    else
18       $\Omega_t^a \leftarrow \text{NONDOMINATEDPRUNE}(\Omega_t^a, \text{mode} = \hat{g})$ 
19    UPDATESUBNETCACHE( $\mathcal{S}, a, S(a, t), \Omega_t^a$ )
20  append all options in  $\Omega_t^a$  to  $\Omega_t^{\text{local}}$ 
21  $\sigma_t \leftarrow \text{RETRIEVALSIGNATURE}(I_t, \Theta)$ 
22  $\mathcal{M}_t \leftarrow \text{RETRIEVELIBRARYITEMS}(\mathcal{G}, \sigma_t, \Theta)$ 
23  $\mathcal{C}_t^{\text{local}} \leftarrow \text{INSTANTIATELOCALOPTIONS}(\Omega_t^{\text{local}}, I_t, \Theta)$ 
24  $\mathcal{C}_t^{\text{general}} \leftarrow \text{INSTANTIATELIBRARYITEMS}(\mathcal{M}_t, I_t, \Theta)$ 
25  $\mathcal{C}_t^{\text{rule}} \leftarrow \text{RULECANDIDATES}(I_t, \Theta)$ 
26  $(a_t, \text{src}, \text{meta}) \leftarrow \text{AGGREGATEANDCHOOSEACTION}(I_t, \mathcal{C}_t^{\text{rule}}, \mathcal{C}_t^{\text{local}}, \mathcal{C}_t^{\text{general}}, \Theta)$ 
27  $(I_{t+1}, \text{obs}, c_t, \text{terminal}) \leftarrow \text{EXECUTEACTION}(I_t, a_t)$ 
28  $\text{ONLINEUPDATE}(I_t, I_{t+1}, a_t, \text{src}, \text{meta}, \text{obs}, c_t, \mathcal{S}, \mathcal{G}, \Theta)$ 
29 if NEEDADDOREBUILDSUBNET( $I_t, I_{t+1}, \text{meta}, \Theta$ ) then
30   TRIGGERSUBNETREBUILD( $I_{t+1}, \mathcal{S}, \Theta$ )
31 if  $h_{t+1} = 1$  then
32   REPLAYKEYSTEPS(TOLIBRARY( $\mathcal{G}$ ))
33 return  $a_t$ 

```

Experimental Design

This chapter presents the experimental protocol used to evaluate the proposed framework. The goal of the evaluation is to assess the effectiveness, robustness, transferability, and parameter sensitivity of the combined use of strict subnet solving, approximate non-dominated local option generation, action-template abstraction, generalized policy-library reuse, and online model revision in the partially observable network-penetration domain considered in this thesis. The penetration-testing domain is used here as a structured evaluation environment rather than as the only intended application domain, which is consistent with earlier work that treats penetration testing as planning under uncertainty rather than as static attack-path analysis (Sarraute et al., 2012).

The experimental study is organised around four groups of comparisons. The first compares the proposed subnet-based framework with a pure global MCTS baseline, in order to evaluate the contribution of restricted-view local solving. The second examines same-type transfer, in which a generalized policy library is learned and reused within the same structural bucket. The third examines cross-type transfer, in which policy experience learned from `small_sparse` and `medium_medium` networks is transferred to `large_dense` target networks. The fourth consists of ablation and sensitivity experiments, including frontier-directed path expansion, subnet size, MCTS rollout budget, progress weight, probability-gating threshold, and aggregation-weight profiles.

10.1. Experimental Objectives

The empirical evaluation pursues five main objectives.

First, it evaluates whether the subnet-based framework yields competitive planning behaviour relative to pure global MCTS. This comparison tests whether strict local solving improves interaction-level efficiency in environments with partial observability, dynamic object discovery, and identity uncertainty.

Second, it examines whether a generalized policy library learned from successful replay on networks of the same structural type can improve subsequent search performance. This evaluates within-bucket transfer, where the source and target networks share broad structural characteristics but differ in concrete node identities and topology.

10. Experimental Design

Third, it assesses whether policy-library experience can transfer across structural types. In particular, experience learned from `small_sparse` and `medium_medium` networks is transferred to `large_dense` networks. This tests whether the current structural signatures and action templates provide useful guidance beyond the source distribution.

Fourth, it evaluates whether important design choices are robust under local parameter changes. The sensitivity experiments do not aim to perform a full hyperparameter optimisation. Instead, they test whether the main conclusions depend on a single fragile parameter setting.

Fifth, it analyses behavioural differences using stage-specific indicators rather than terminal success alone. Since the task is not a shortest-path problem on a known graph, final success rate is not sufficient. The experiments therefore also report success step, remaining budget, first entry into the *C* layer, first entry into the database layer, and total runtime.

10.2. Environment and Task Setting

All experiments are conducted in the same simulated network-penetration environment. The true network is defined by a JSON graph instance. The simulator maintains the full topology, node attributes, true node roles, database-node set, edge movement costs, and edge movement-success probabilities, together with runtime quantities such as current location, privilege state, credential availability, and task-success status.

The planner does not observe this full state. It begins from an entry node with only limited local information and progressively constructs an internal knowledge graph through interaction. A key modelling constraint is that early actions such as `scan`, `traffic_sense`, `move`, `exploit`, and `query_db` may reveal handles, fingerprints, movement feedback, or task feedback, but they do not directly reveal stable identities. Stable identity evidence is revealed only through `confirm_identity`. Thus, the experiments evaluate online planning under partial observability and evolving object identity, rather than offline search on a fully known graph.

10.3. Network Buckets and Instances

The evaluation uses three structural buckets:

$$\text{small_sparse}, \quad \text{medium_medium}, \quad \text{large_dense}. \quad (10.1)$$

Each bucket contains five graph instances. In the main baseline comparison, each method is evaluated on all five instances in each bucket, and each instance is run under five random seeds. Thus, each method–bucket pair in the main comparison contains

$$5 \text{ instances} \times 5 \text{ seeds} = 25 \text{ runs}. \quad (10.2)$$

The `small_sparse` bucket represents comparatively shallow and weakly connected networks. The `medium_medium` bucket provides an intermediate regime. The `large_dense` bucket represents the most ambiguous and

difficult setting, where many local alternatives can produce lateral wandering, misleading frontier estimates, or policy-library mismatch. This organisation allows the experiments to analyse both absolute performance and the stability of transfer as structural complexity increases.

A single run is identified by the tuple

$$(\text{method}, \text{bucket}, \text{instance}, \text{seed}). \quad (10.3)$$

The random seed controls stochastic execution outcomes, local MCTS rollout sampling, and stochastic selection or tie-breaking in the global action-selection layer. When two methods are compared on a matched subset, the same instance–seed pairs are used where possible. This reduces the chance that observed differences are caused only by different random execution realisations.

10.4. Compared Methods

The following method names are used throughout the evaluation.

MCTS-Pure. This is the pure global MCTS baseline. It does not use strict subnet construction, approximate non-dominated local option generation, or the generalized policy library. It is used to evaluate performance without local abstraction and without cross-episode experience reuse.

Subnet-Empty. This setting uses the internal knowledge graph, strict subnet construction, budgeted local MCTS, approximate non-dominated local option generation, and global action aggregation, but uses an empty generalized policy library. It evaluates the contribution of the subnet-based planning framework itself, before any generalized policy-library reuse is introduced.

Subnet-SameTransfer. This setting augments **Subnet-Empty** with a generalized policy library learned from networks of the same structural type. It is evaluated on held-out target instances from the same bucket. This setting tests whether local planning experience can be abstracted and reused within structurally similar network families, rather than merely memorising the original training instances.

Subnet-CrossTransfer. This setting uses a generalized policy library learned from `small_sparse` and `medium_medium` networks and evaluates it on `large_dense` target instances. This setting is reported separately because its purpose is to evaluate cross-type transfer rather than same-family reuse.

10.5. Ablation and Sensitivity Settings

In addition to the main comparison settings, the thesis reports several ablation and sensitivity experiments. These experiments use matched subsets of instances and seeds where possible. They should therefore be interpreted as diagnostic evidence about design behaviour and local robustness, rather than as full hyperparameter optimisation.

10. Experimental Design

Table 10.1.: Ablation and sensitivity experiments. The default setting is the **Subnet-Empty** configuration, which serves as the medium or reference condition.

Experiment	Settings	Purpose
Frontier expansion	default / no expansion	Test whether frontier-directed path inclusion improves deep exploration.
Subnet size	small / default / large	Test sensitivity to local solving-domain size.
MCTS budget	low / default / high	Test sensitivity to rollout count and rollout depth.
Progress weight	$w_g = 0$ / default / higher w_g	Test whether progress guidance improves robustness.
Probability gate	$p_{\text{gate}} = 0$ / default / higher p_{gate}	Test the transition between progress-guided and success-guided filtering.
Aggregation profile	progress-aware / conservative / success-oriented	Test different global action-ranking profiles.

For these sensitivity studies, the default **Subnet-Empty** configuration is treated as the medium or reference condition. This default configuration uses the internal knowledge graph, strict subnet construction, budgeted local MCTS, approximate non-dominated local option generation, and global action aggregation, but does not use a generalized policy library. It therefore provides the reference point for evaluating how changes to subnet size, MCTS budget, progress weight, probability gate, and frontier-directed path expansion affect the subnet-based framework itself.

In the aggregation-weight-profile experiment, the default configuration is interpreted as the *progress-aware* profile, because it includes an explicit progress term and balances predicted success with progress, risk, distance, and cost. The conservative and success-oriented profiles modify this global ranking behaviour in order to test whether the observed results depend strongly on the default aggregation weights.

The sensitivity experiments are not intended to identify globally optimal parameter values. The framework contains many coupled parameters, including subnet-size limits, rollout budget, progress weight, probability gate, aggregation weights, and candidate-adjustment terms. Exhaustive optimisation over all combinations would require a separate hyperparameter-optimisation study. The purpose here is more limited: to test whether the reported conclusions remain stable under representative local changes to key parameters.

10.6. Evaluation Metrics

The evaluation reports the following metrics.

Success Rate. The proportion of runs for which `db_hit=True`. This is the primary terminal-performance metric. It is computed over all available runs in the corresponding setting.

Success Step. The interaction step at which `query.db` first succeeds. This metric is averaged only over successful runs, because failed runs do not have a meaningful success step.

Budget Remaining. The remaining budget at successful termination. This measures resource efficiency among successful runs and is averaged only over successful runs.

First-*C* Step. The first step at which a *C*-layer node is reached. This measures entry into a key intermediate structural layer. It is averaged over successful runs for which the event is observed.

First-DB Step. The first step at which a database-layer node is reached. This measures stagewise progress toward terminally relevant regions. It is averaged over successful runs for which the event is observed.

Total Time. The total wall-clock runtime of an episode. This captures planning overhead and execution overhead together. Unlike success step and remaining budget, total time is averaged over all available runs, because failed runs also consume computational time.

General Step Ratio. For transfer-based methods, this measures the proportion of executed actions selected from generalized policy-library candidates. It indicates how strongly the library influences actual execution.

10.7. Reporting Protocol

The tables report both the number of available runs and the number of expected runs. The notation

$$m/n \tag{10.4}$$

means that m valid runs were available out of n expected runs for that setting. For the main baseline comparison, all settings use the balanced design of five instances and five seeds per bucket, giving 25 runs per method–bucket pair. Same-type transfer uses held-out target instances and reports 10 runs per bucket. Cross-type transfer is evaluated on the `large_dense` bucket and reports 25 runs.

For all tables, success is reported as both a count and a percentage, for example 20/25 (80.00%). This makes explicit how the success rate is computed. Continuous metrics are reported as mean $\pm$ sample standard deviation. Success step, budget remaining, first-*C*, and first-DB are averaged over successful runs unless otherwise stated. Total runtime is averaged over all available runs.

Because the number of runs is finite, the reported means should be interpreted as empirical estimates rather than exact population-level performance values. The standard deviations are included to make

10. Experimental Design

variability visible. The results chapter therefore reports distribution plots, including boxplots for success step, budget remaining, and runtime, to complement the tabular averages.

10.8. Chapter Summary

This chapter described the experimental design used to evaluate the proposed framework. It specified the task setting, network buckets, instance and seed structure, compared methods, evaluation metrics, reporting protocol, and the ablation and sensitivity experiments used to answer concerns about design choices and parameter robustness. The next chapter presents the results and analyses them in terms of robustness, interaction efficiency, transferability, and sensitivity to key planner parameters.

Experimental Results and Analysis

This chapter reports and analyses the experimental results with respect to the objectives defined in Chapter 10. The reported values are empirical summaries over the runs described in Section 10.3 and Section 10.7. Success rates are computed over all available runs. Unless otherwise stated, success step, remaining budget, first- C , and first-DB are averaged over successful runs; total runtime is averaged over all available runs.

The main experimental summaries are reported directly in this chapter, including success counts, success step, remaining budget, stage indicators, and runtime. Tables report mean $\pm$ sample standard deviation for continuous metrics, and distribution plots are included to show run-level variability in the main comparisons.

Overall, the results show a consistent behavioural pattern. Pure MCTS is the strongest method in raw success robustness and wall-clock runtime. The subnet-based framework is less robust in some settings, but successful subnet-based runs usually reach the goal in fewer action steps and preserve more budget. Same-type policy-library transfer produces the strongest and most stable improvement. Cross-type transfer is useful but unstable. The ablation and sensitivity experiments provide diagnostic evidence that frontier-directed path expansion, progress-aware aggregation, the default probability gate, and the default local-search budget improve robustness under the tested settings. These sensitivity experiments should not be interpreted as exhaustive hyperparameter optimisation.

11.1. Reporting and Variability

The experiments involve stochastic execution outcomes, stochastic rollout sampling inside MCTS, and stochastic or tie-breaking behaviour in action selection. To control for this randomness, the main comparison evaluates each method on multiple graph instances and multiple random seeds. In the main baseline comparison, each bucket contains five graph instances, and each instance is evaluated under five random seeds, giving 25 runs per method–bucket pair.

For matched comparisons, the same instance–seed pairs are used where possible. This makes the comparison less dependent on a particular random execution realisation. However, the reported averages should still be interpreted as empirical summaries, not as exact population-level estimates over all possible network structures. For this reason, the tables report success counts and sample standard deviations, and the figures show the distribution of run-level outcomes.

11. Experimental Results and Analysis

Table 11.1.: Full comparison between pure MCTS and the empty-library subnet framework, including stage indicators. Continuous values are mean $\pm$ sample standard deviation. Step, budget, first- C , and first-DB are computed over successful runs; time is computed over all runs.

Setting	Success	Step	Budget	First- C	First-DB	Time
SS	25/25	29.60 ± 5.39	15.92 ± 8.43	6.80 ± 2.31	14.04 ± 7.40	71.81 ± 20.21
MCTS-Pure	(100.00%)					
SS Subnet-Empty	24/25	24.29 ± 7.85	33.21 ± 12.01	15.92 ± 7.86	20.54 ± 7.57	849.32 ± 367.76
	(96.00%)					
MM	25/25	32.24 ± 5.43	10.36 ± 8.08	5.28 ± 1.49	13.32 ± 7.29	107.24 ± 28.21
MCTS-Pure	(100.00%)					
MM Subnet-Empty	23/25	29.91 ± 7.93	25.28 ± 12.24	20.13 ± 8.23	26.52 ± 8.10	1392.56 ± 619.08
	(92.00%)					
LD	25/25	33.80 ± 4.83	9.84 ± 9.20	6.60 ± 2.53	17.40 ± 7.33	176.62 ± 34.82
MCTS-Pure	(100.00%)					
LD Subnet-Empty	20/25	26.65 ± 10.12	33.60 ± 13.69	18.90 ± 10.05	23.35 ± 10.22	2272.05 ± 1416.45
	(80.00%)					

SS = `small_sparse`, MM = `medium_medium`, LD = `large_dense`.

Unless otherwise stated, success step and remaining budget are computed only over successful runs, because failed runs do not have a meaningful success step or terminal remaining-budget value. Total runtime is computed over all available runs, because failed episodes also consume planning time.

11.2. Pure MCTS versus the Empty-Library Subnet Framework

Table 11.1 compares pure global MCTS with **Subnet-Empty**. Each method-bucket pair contains 25 runs, corresponding to five graph instances and five random seeds per instance. The table reports both terminal metrics and stage indicators, including first- C and first-DB timing.

Pure MCTS achieves 25/25 successes in all three buckets. By contrast, **Subnet-Empty** achieves 24/25, 23/25, and 20/25 successes on `small_sparse`, `medium_medium`, and `large_dense`, respectively. Thus, pure MCTS remains more robust in terminal success rate.

Among successful episodes, however, **Subnet-Empty** is more interaction-efficient. In all three buckets, successful **Subnet-Empty** runs use fewer action steps and leave more remaining budget than pure MCTS. For example, in `large_dense`, mean success step decreases from 33.80 ± 4.83 to 26.65 ± 10.12 , and mean remaining budget increases from 9.84 ± 9.20 to 33.60 ± 13.69 . This indicates that the subnet framework is not simply searching faster or more broadly; rather, when it succeeds, it tends to use interaction resources more economically.

The stage indicators reveal a further behavioural difference. Pure MCTS reaches C -layer and database-layer nodes earlier than **Subnet-Empty** in all three buckets. This suggests that pure MCTS is effective

11.2. Pure MCTS versus the Empty-Library Subnet Framework

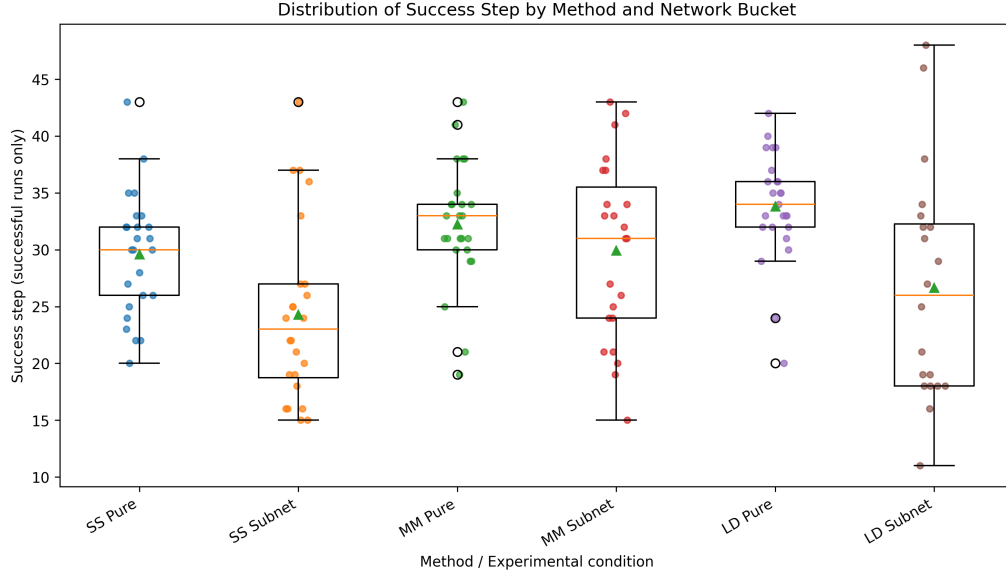


Figure 11.1.: Distribution of success step for pure MCTS and **Subnet-Empty** across the three network buckets. Each point denotes one successful run, while the boxplots summarise the distribution of successful-run outcomes. Lower values indicate that the goal is reached in fewer interaction steps.

at quickly reaching deeper regions of the graph. The subnet framework reaches these regions later, but when it succeeds, it tends to complete the task with fewer total steps and substantially more remaining budget. Thus, the subnet framework appears to improve interaction efficiency after useful local structure has been identified, rather than improving early reachability.

The runtime comparison is less favourable to the proposed framework. Pure MCTS has substantially smaller wall-clock time in all three buckets. Therefore, the current framework should be interpreted primarily as improving *interaction efficiency* and *budget efficiency*, not wall-clock efficiency.

Figures 11.1–11.3 show the run-level distributions behind the averages in Table 11.1. Figure 11.1 shows that successful **Subnet-Empty** runs generally require fewer interaction steps than pure MCTS, especially in the **large_dense** bucket. Figure 11.2 shows the corresponding budget effect: successful subnet-based runs tend to terminate with substantially more remaining budget. At the same time, Figure 11.3 shows the opposite pattern for wall-clock time. Even on a logarithmic scale, the subnet-based method is consistently slower than pure MCTS. These distributions support the central interpretation of this comparison: the proposed framework improves interaction efficiency when it succeeds, but it does not currently improve computational efficiency.

11. Experimental Results and Analysis

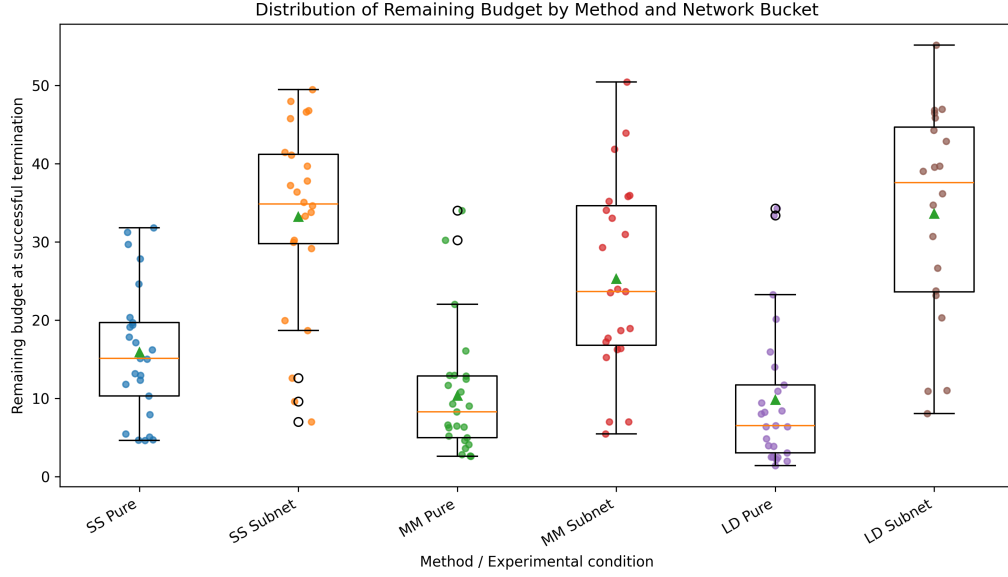


Figure 11.2.: Distribution of remaining budget for pure MCTS and **Subnet-Empty** across the three network buckets. Only successful runs are included. Higher values indicate that the planner reaches the goal more economically.

11.3. Policy-Library Transfer

The transfer experiments evaluate whether local planning experience can be abstracted and reused through the generalized policy library. The same-type transfer experiment tests reuse within the same structural bucket, while the cross-type transfer experiment tests whether experience learned from `small_sparse` and `medium_medium` networks can guide planning in `large_dense` networks.

11.3.1. Same-Type Transfer

Table 11.2 reports same-type transfer results on held-out instances. The empty-library baseline is evaluated on the same held-out instances as the transfer condition, making this a matched comparison. The table includes both final performance metrics and stage indicators.

The same-type library improves performance in the reported held-out runs. On `small_sparse`, success remains 10/10, while success step decreases and remaining budget increases. On `medium_medium`, the same pattern holds. On `large_dense`, same-type transfer increases success from 7/10 to 10/10, while also reducing success step.

This provides the strongest empirical support for the policy-library mechanism. The library does not store concrete node-specific paths. Instead, it stores state-bucket-to-template local strategy priors that can guide planning in structurally similar local contexts.

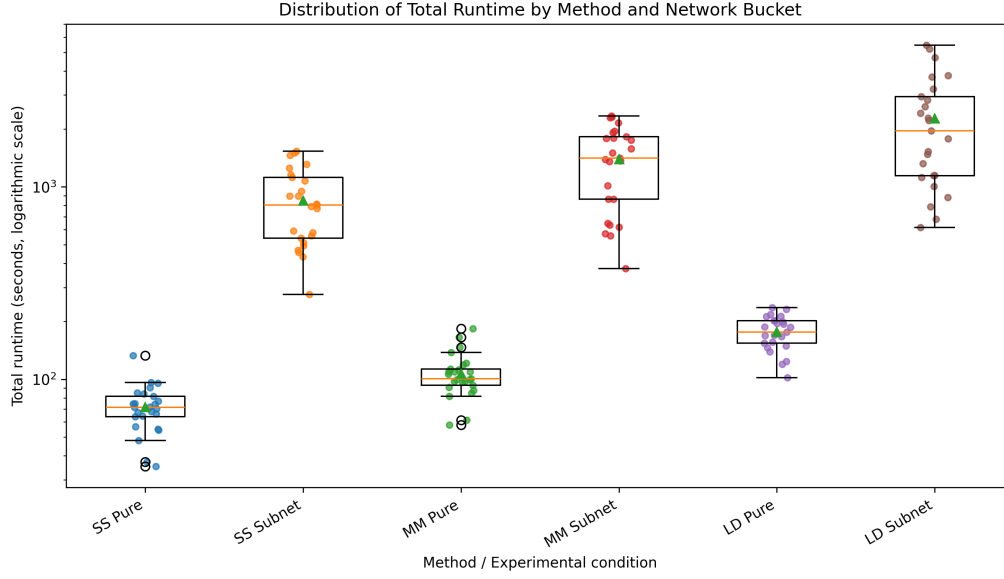


Figure 11.3.: Distribution of total runtime for pure MCTS and **Subnet-Empty** across the three network buckets. All available runs are included. The vertical axis is shown on a logarithmic scale because the runtime of the subnet-based method is substantially larger than that of pure MCTS.

Figures 11.4 and 11.5 make the same-type transfer effect visible at the run level. The success-step distribution shifts downward under same-type transfer in all three buckets, indicating that the generalized policy library helps the planner reach successful termination in fewer actions. The remaining-budget distribution also shifts upward, showing that the improvement is not only a reduction in step count but also a better use of the execution budget. This distributional evidence supports the claim that the library provides reusable local decision priors in structurally similar contexts, rather than merely improving a small number of average values.

The stage indicators show that the generalized policy library affects more than the final query phase. In all three buckets, same-type transfer improves first- C and first-DB timing. For example, in **large_dense**, first- C improves from 15.86 ± 10.46 to 8.10 ± 2.02 , and first-DB improves from 20.43 ± 10.55 to 15.20 ± 8.27 . This suggests that the library provides useful directional guidance during intermediate movement and capability-building stages, not only during terminal database querying.

11.3.2. Cross-Type Transfer

The cross-type transfer experiment uses a library learned from **small_sparse** and **medium_medium**, and evaluates it on **large_dense**. Table 11.3 compares the cross-transfer condition with the **Subnet-Empty** baseline on **large_dense**.

11. Experimental Results and Analysis

Table 11.2.: Full held-out same-type transfer results, including stage indicators. Continuous values are mean $\pm$ sample standard deviation. Step, budget, first- C , and first-DB are computed over successful runs; time is computed over all runs.

Setting	Success	Step	Budget	First- C	First-DB	Time
SS Empty-heldout	10/10 (100.00%)	27.70 ± 7.66	27.66 ± 12.79	19.30 ± 7.93	23.40 ± 7.72	1004.17 ± 364.52
SS Same-Transfer	10/10 (100.00%)	21.00 ± 5.93	35.20 ± 14.38	9.40 ± 2.63	17.40 ± 6.11	737.57 ± 289.04
MM Empty-heldout	10/10 (100.00%)	30.50 ± 8.72	23.90 ± 14.45	20.40 ± 8.54	27.20 ± 9.07	1496.88 ± 690.18
MM Same-Transfer	10/10 (100.00%)	23.70 ± 11.76	34.44 ± 17.79	12.80 ± 10.62	19.50 ± 11.07	680.62 ± 224.47
LD Empty-heldout	7/10 (70.00%)	23.43 ± 10.49	38.40 ± 12.85	15.86 ± 10.46	20.43 ± 10.55	2354.83 ± 1728.99
LD Same-Transfer	10/10 (100.00%)	19.20 ± 7.48	39.16 ± 15.41	8.10 ± 2.02	15.20 ± 8.27	985.48 ± 498.11

SS = `small_sparse`, MM = `medium_medium`, LD = `large_dense`.

Cross-type transfer does not improve the overall success rate: both settings achieve 20/25 successes. However, it slightly improves success step among successful runs and reduces total runtime.

The stage indicators in Table 11.3 show that cross-type transfer improves first- C from 18.90 ± 10.05 to 12.90 ± 9.50 , and first-DB from 23.35 ± 10.22 to 19.75 ± 10.33 . Therefore, although cross-type transfer does not improve terminal success rate, it does provide earlier directional movement toward capability-relevant and database-relevant regions. The correct interpretation is therefore mixed. Cross-type transfer provides useful directional bias and reduces some planning overhead, but it is not sufficiently reliable to improve overall success probability. This is consistent with the limitations of the current structural signatures: a signature learned in smaller or sparser networks can still point toward useful intermediate actions, but it may not always preserve the correct meaning in denser networks.

11.4. Ablation and Sensitivity Analysis

The full mean $\pm$ sample standard deviation summaries for the ablation and sensitivity settings are reported directly in this chapter. The analysis below focuses on the main behavioural patterns supported by these tables.

11.4.1. Frontier-Directed Path Expansion

The frontier-directed path expansion ablation directly addresses whether this mechanism helps deep exploration under strict subnet limits. Table 11.4 compares the default configuration with the version

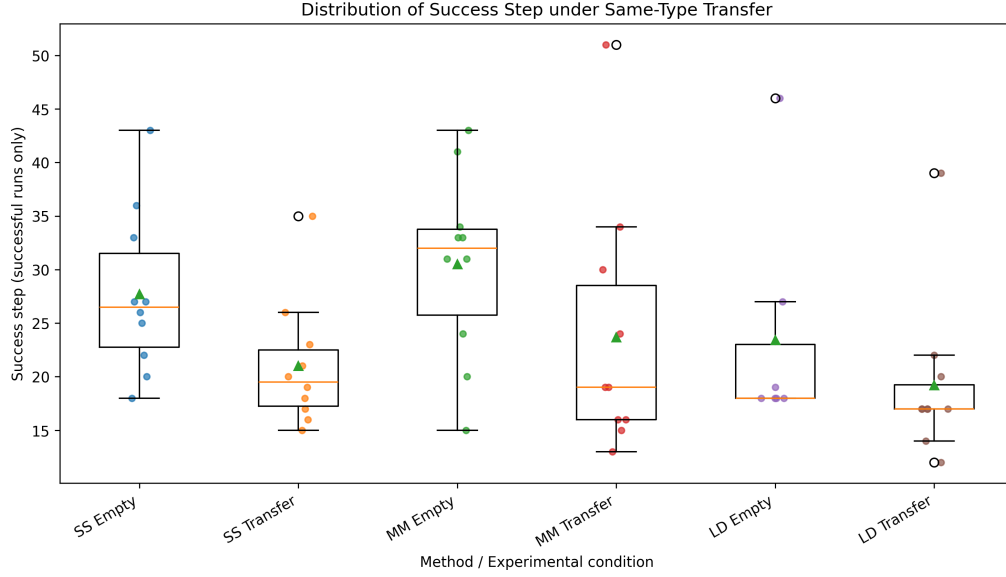


Figure 11.4.: Distribution of success step for the held-out empty-library baseline and the same-type transfer condition. Each point denotes one successful run. Lower values indicate more efficient task completion.

that disables frontier-directed path expansion. Because this mechanism is intended to support deeper exploration under strict subnet limits, both first- C and first-DB are reported in addition to terminal and efficiency metrics.

Removing frontier-directed path expansion reduces success from 23/27 to 20/27, increases success step from 25.48 ± 8.08 to 31.65 ± 10.98 , decreases remaining budget from 32.33 ± 12.31 to 23.15 ± 15.96 , and delays first-DB from 21.61 ± 8.02 to 27.80 ± 11.76 . First- C is also delayed, from 16.61 ± 7.75 to 19.65 ± 9.70 . These results support the design motivation of frontier-directed path expansion.

The mechanism does not create executable shortcut edges in the environment. Instead, it includes currently known paths toward high-frontier nodes in the local solving view. The ablation results suggest that this inclusion helps local MCTS maintain deeper exploration direction under strict subnet limits.

Figures 11.6 and 11.7 show that the effect of frontier-directed path expansion is visible at the run level. Without frontier-directed path expansion, the success-step distribution shifts upward and shows larger observed variability in the tested runs. The first-DB distribution also shifts later, which is consistent with the intended role of the mechanism: it helps local search preserve directions toward deeper database-relevant regions under strict subnet limits. Thus, the ablation provides direct empirical support for keeping frontier-directed path expansion in the framework.

11. Experimental Results and Analysis

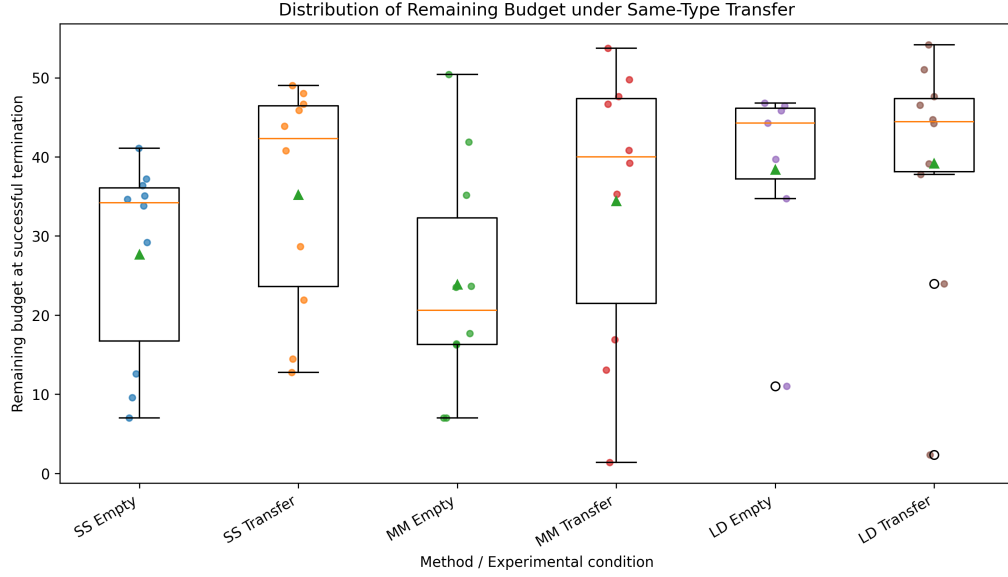


Figure 11.5.: Distribution of remaining budget for the held-out empty-library baseline and the same-type transfer condition. Only successful runs are included. Higher values indicate better budget efficiency.

11.4.2. Other Sensitivity Results

Tables 11.5 and 11.6 report the remaining sensitivity experiments. Table 11.5 reports success, success step, remaining budget, and runtime, while Table 11.6 reports first- C and first-DB timing.

Together, Tables 11.5 and 11.6 show both final outcome changes and stagewise behavioural changes under each tested parameter variation.

The subnet-size results suggest that the planner is not simply improved by increasing the local search domain. In the tested problem setting and network scales, the default subnet size performs better than the larger setting in success rate and runtime, and better than the smaller setting in step and budget efficiency. This indicates that subnet size introduces a trade-off: a larger subnet can increase branching and local ambiguity, while an overly small subnet can become locally myopic. Therefore, the default setting should be interpreted as a reasonable configuration for the tested experimental regime, rather than as a generally optimal subnet size.

The MCTS-budget results show a reliability–runtime trade-off. The low-budget setting is cheaper but less robust, while the high-budget setting improves some successful-run metrics at higher computational cost. The default budget is therefore a practical compromise in the tested range.

The progress-weight and aggregation-profile experiments support the role of progress-aware decision making. Removing progress guidance lowers success, even though successful runs can sometimes be efficient. This suggests that progress acts as a stabilising signal before reliable terminal-success estimates are available. Conversely, over-emphasising progress can delay terminal completion.

11.5. Summary of Findings

Table 11.3.: Full cross-type transfer results on `large_dense`, including stage indicators. Continuous values are mean $\pm$ sample standard deviation. Step, budget, first- C , and first-DB are computed over successful runs; time is computed over all available runtime records. The cross-transfer runtime uses 24 non-empty runtime records.

Setting	Success	Step	Budget	First- C	First-DB	Time
Subnet-Empty	20/25 (80.00%)	26.65 $\pm$ 10.12	33.60 $\pm$ 13.69	18.90 $\pm$ 10.05	23.35 $\pm$ 10.22	2272.05 $\pm$ 1416.45
Subnet-CrossTransfer	20/25 (80.00%)	24.85 $\pm$ 10.15	32.29 $\pm$ 16.98	12.90 $\pm$ 9.50	19.75 $\pm$ 10.33	1520.95 $\pm$ 1120.15

Table 11.4.: Full frontier-directed path expansion ablation results. Continuous values are mean $\pm$ sample standard deviation. Step, budget, first- C , and first-DB are computed over successful runs; time is computed over all available runs.

Setting	Success	Step	Budget	First- C	First-DB	Time
With expansion/default	23/27 (85.19%)	25.48 $\pm$ 8.08	32.33 $\pm$ 12.31	16.61 $\pm$ 7.75	21.61 $\pm$ 8.02	1536.85 $\pm$ 1073.85
Without expansion	20/27 (74.07%)	31.65 $\pm$ 10.98	23.15 $\pm$ 15.96	19.65 $\pm$ 9.70	27.80 $\pm$ 11.76	1766.46 $\pm$ 1204.13

The probability-gate experiment shows that both lower and higher alternatives perform worse than the default in the tested subset. This supports the intended role of the gate: it delays success-oriented filtering when success estimates are weak, but allows success-oriented selection once a sufficiently promising local opportunity appears.

Figure 11.8 summarises the success-rate patterns across the sensitivity experiments. The default settings are not claimed to be globally optimal, but they are consistently among the most robust settings in terms of success rate. The clearest drops occur when progress guidance is removed, when the probability gate is set too low or too high, and when the aggregation profile becomes purely success-oriented. This supports the interpretation that progress-aware aggregation and probability gating are not arbitrary decorative components; they stabilise the planner when terminal-success estimates are still uncertain.

11.5. Summary of Findings

The experimental results support five main findings.

First, pure MCTS remains the strongest baseline in success robustness and wall-clock runtime. The proposed framework does not currently outperform pure MCTS as a low-overhead planner.

11. Experimental Results and Analysis

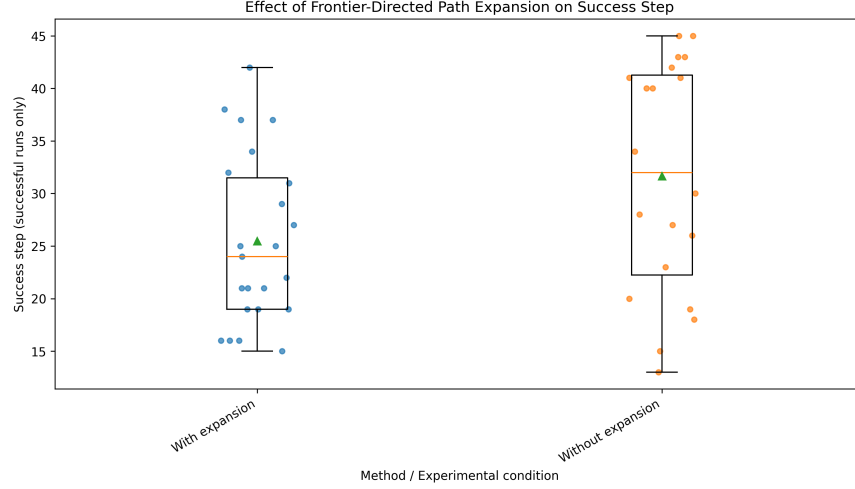


Figure 11.6.: Effect of frontier-directed path expansion on the distribution of success step. Each point denotes one successful run. Lower values indicate that deep progress is achieved more efficiently.

Second, the subnet framework improves interaction efficiency when it succeeds. It generally completes successful episodes in fewer action steps and preserves more budget.

Third, same-type generalized policy reuse provides the strongest evidence for useful abstraction. It improves success step, remaining budget, and runtime, and in **large_dense** also improves success rate.

Fourth, cross-type transfer is partially effective but unstable. It improves some stage indicators and runtime in **large_dense**, but it does not improve success rate over the empty-library baseline.

Fifth, the ablation and sensitivity experiments provide diagnostic support for several design choices. Frontier-directed path expansion improves deep exploration robustness; the default subnet size balances local focus and search coverage; the default MCTS budget balances reliability and runtime; progress-aware aggregation stabilises decisions; and the default probability gate provides a useful transition between progress-guided and success-guided local filtering. These findings should be interpreted as empirical evidence under the tested settings, not as proof of globally optimal parameter choices.

11.5. Summary of Findings

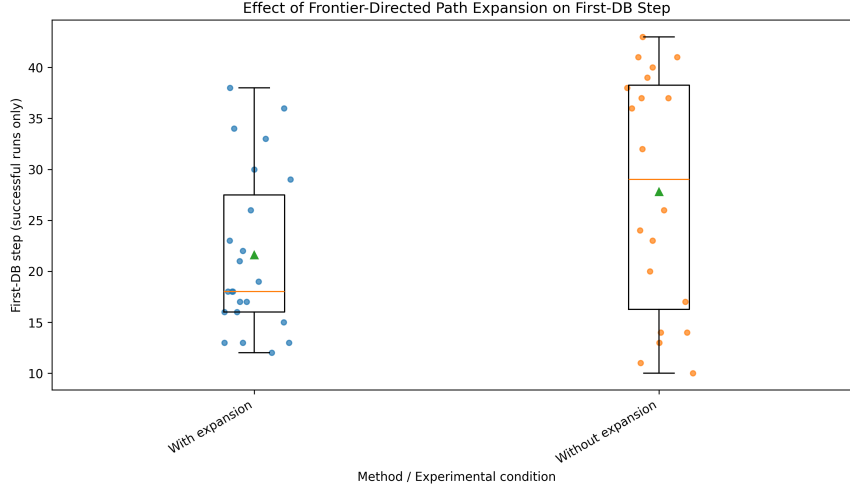


Figure 11.7.: Effect of frontier-directed path expansion on the distribution of first-DB step. Only successful runs are included. Lower values indicate earlier arrival at database-relevant regions.

Table 11.5.: Full sensitivity results for success and efficiency metrics. The default setting is taken from matched **Subnet-Empty** baseline rows. Step and budget are computed over successful runs; time is computed over all available runs.

Setting	Success	Step	Budget	Time
Subnet small	12/12 (100.00%)	29.25 ± 10.35	26.00 ± 16.46	1250.65 ± 639.27
Subnet default	12/12 (100.00%)	24.88 ± 8.22	32.63 ± 14.12	1210.14 ± 703.18
Subnet large	10/12 (83.33%)	26.90 ± 8.99	29.23 ± 13.31	1719.40 ± 743.48
MCTS low	10/14 (71.43%)	25.50 ± 9.73	30.13 ± 14.95	890.41 ± 665.93
MCTS default	12/12 (100.00%)	25.50 ± 8.46	31.77 ± 14.27	1301.70 ± 816.89
MCTS high	11/12 (91.67%)	22.45 ± 5.52	37.04 ± 9.15	1709.37 ± 1175.48
$w_g = 0.00$	9/12 (75.00%)	20.11 ± 7.15	40.97 ± 12.52	1418.91 ± 1237.35
$w_g = 0.08$ default	12/12 (100.00%)	25.50 ± 8.46	31.77 ± 14.27	1301.70 ± 816.89
$w_g = 0.16$	10/12 (83.33%)	26.80 ± 7.39	31.00 ± 10.78	1862.86 ± 1020.46
$p_{\text{gate}} = 0.00$	7/12 (58.33%)	26.29 ± 8.52	32.32 ± 9.85	1842.57 ± 870.29
$p_{\text{gate}} = 0.05$ default	12/12 (100.00%)	25.50 ± 8.46	31.77 ± 14.27	1301.70 ± 816.89
$p_{\text{gate}} = 0.10$	8/12 (66.67%)	30.25 ± 12.52	23.01 ± 17.37	2381.06 ± 1174.92
Aggregation progress/default	14/16 (87.50%)	24.43 ± 8.24	33.90 ± 14.19	1672.45 ± 1332.98
Aggregation conservative	12/16 (75.00%)	29.08 ± 10.34	25.48 ± 16.64	2245.75 ± 1541.45
Aggregation success-oriented	11/16 (68.75%)	32.36 ± 10.33	22.82 ± 13.30	2284.74 ± 1067.31

11. Experimental Results and Analysis

Table 11.6.: Full sensitivity results for stage indicators. First- C and first-DB are averaged over successful runs.

Setting	First- C	First-DB
Subnet small	16.42 ± 9.70	23.25 ± 9.96
Subnet default	15.54 ± 7.63	20.67 ± 7.76
Subnet large	18.30 ± 8.54	22.80 ± 8.40
MCTS low	13.20 ± 7.51	20.40 ± 9.94
MCTS default	16.17 ± 8.00	21.25 ± 8.11
MCTS high	13.45 ± 5.73	18.45 ± 5.30
$w_g = 0.00$	12.89 ± 6.99	17.33 ± 7.42
$w_g = 0.08$ default	16.17 ± 8.00	21.25 ± 8.11
$w_g = 0.16$	18.80 ± 6.63	23.60 ± 7.00
$p_{\text{gate}} = 0.00$	15.71 ± 9.59	22.71 ± 8.83
$p_{\text{gate}} = 0.05$ default	16.17 ± 8.00	21.25 ± 8.11
$p_{\text{gate}} = 0.10$	15.00 ± 9.41	25.12 ± 13.54
Aggregation progress/default	15.36 ± 7.64	20.36 ± 7.80
Aggregation conservative	16.83 ± 10.00	24.92 ± 10.26
Aggregation success-oriented	22.09 ± 11.85	28.45 ± 10.95

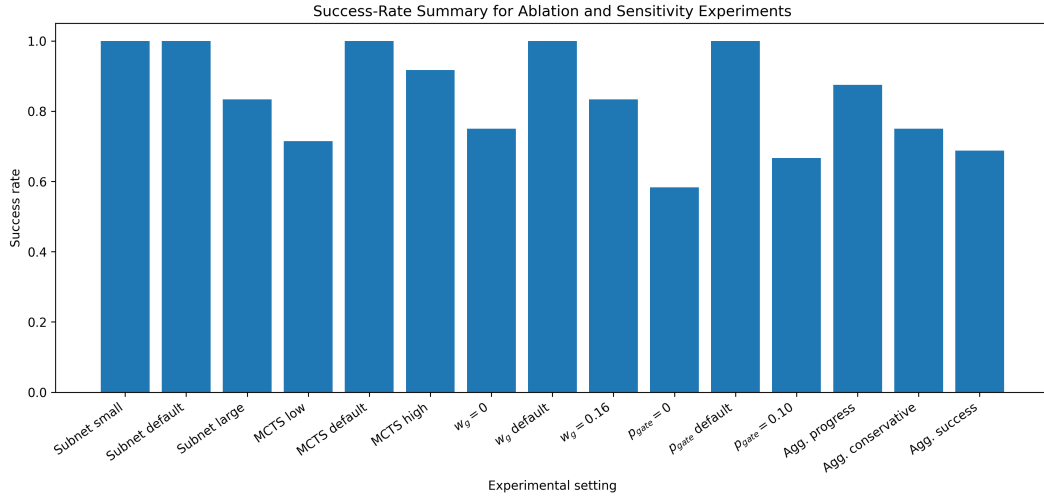


Figure 11.8.: Success-rate summary for the ablation and sensitivity experiments. Each bar reports the empirical success rate under one tested setting. The figure provides a compact overview of the robustness of the main design choices.

Discussion

This chapter interprets the experimental results reported in Chapter 11. The purpose is not to repeat every numerical comparison, but to clarify what the results do and do not support about the proposed framework. The discussion therefore distinguishes between empirical findings, plausible interpretations, and limitations of the current implementation.

The central result is not that the proposed framework is a uniformly better replacement for pure MCTS. Rather, the results show a trade-off. Pure MCTS remains stronger in raw success robustness and wall-clock runtime, while the subnet-based framework tends to use fewer interaction steps and preserve more budget in successful runs. Same-type policy-library transfer provides the strongest evidence for reusable local strategic experience, whereas cross-type transfer provides useful but less reliable guidance. The contribution should therefore be interpreted as a structured online planning framework that improves interaction-level efficiency and enables reusable local decision priors under certain conditions, not as a generally faster or universally more robust planner.

12.1. Trade-Off Relative to Pure MCTS

The main comparison shows that pure MCTS remains the strongest baseline in terms of terminal success robustness and computational time. It achieves 25/25 successes in all three main buckets and has much lower wall-clock runtime than the subnet-based framework. This is an important negative result for the proposed method: the framework should not be described as a faster or uniformly stronger replacement for pure MCTS.

The advantage of the subnet-based framework lies instead in interaction-level efficiency among successful runs. In all three buckets, successful **Subnet-Empty** runs reach terminal success in fewer action steps and preserve more remaining budget than pure MCTS. For example, in **large_dense**, pure MCTS succeeds in 25/25 runs with a mean success step of 33.80 ± 4.83 and mean remaining budget of 9.84 ± 9.20 , while **Subnet-Empty** succeeds in 20/25 runs with a mean success step of 26.65 ± 10.12 and mean remaining budget of 33.60 ± 13.69 . Thus, the subnet framework is less robust in terminal success rate, but successful subnet-based runs are substantially more economical in their use of the interaction budget.

This distinction is central to the interpretation of the thesis contribution. The proposed framework improves the quality of online interaction when it succeeds, but it does not yet improve computational efficiency. In settings where interaction cost, execution budget, or the cost of failed actions is more important than planning time, this trade-off may be acceptable. In settings where wall-clock response

12. Discussion

time is the dominant requirement, the current implementation would require significant optimisation before it could be competitive.

The distribution plots in Chapter 11 reinforce this interpretation. The success-step and remaining-budget distributions show that the interaction-efficiency advantage is not only a single average-value effect. At the same time, the log-scale runtime distribution shows that the subnet-based method is consistently slower than pure MCTS. The current framework therefore trades computational overhead for structured local reasoning and budget-efficient successful interaction.

12.2. Role of Strict Subnet Solving

The results suggest that strict subnet solving acts primarily as a local decision-quality mechanism. The subnet framework constrains local reasoning to a restricted view of the internal knowledge graph. This restriction can slow early exploration or reduce robustness if the selected local view excludes relevant alternatives. However, when the selected subnet contains useful structure, it can reduce interference from remote or weakly relevant parts of the discovered graph and produce more focused local options.

This interpretation is consistent with the main comparison. **Subnet-Empty** is less robust than pure MCTS in success rate, but successful subnet-based runs require fewer action steps and preserve more budget. This suggests that the benefit of strict subnet solving is not broad search coverage. Instead, its benefit is more selective local reasoning once the planner is operating in a relevant part of the internal graph.

The subnet-size sensitivity results support a cautious version of this conclusion. Under the tested problem setting and network scales, the default subnet size performs better than the larger setting in success rate and runtime, and better than the smaller setting in step and budget efficiency. This does not imply that the default subnet size is generally optimal. Rather, it suggests that subnet size introduces a trade-off. A larger subnet can reintroduce branching and local ambiguity, while an overly small subnet can become locally myopic. The default setting should therefore be interpreted as a reasonable configuration for the tested experimental regime, not as a universal subnet-size choice.

Thus, strict subnet solving should be understood as a biasing mechanism rather than as a universally beneficial reduction of the search space. It focuses computation on a local region that is expected to be useful under the current internal state. This can improve local decision quality when the selected subnet is appropriate, but it can also exclude useful alternatives if the subnet is too small, poorly anchored, or based on inaccurate internal beliefs. The effectiveness of the subnet framework therefore depends not only on subnet construction, but also on anchor selection, active-subnet selection, frontier handling, and the mechanisms that reuse or rebuild subnets during online interaction.

12.3. Role of Frontier-Directed Path Expansion

The frontier-directed path expansion ablation provides empirical support for this design under the tested configurations. Removing the mechanism reduces success from 23/27 to 20/27, increases mean success

12.4. Progress-Aware Decision Making

step from 25.48 ± 8.08 to 31.65 ± 10.98 , decreases remaining budget from 32.33 ± 12.31 to 23.15 ± 15.96 , and delays first-DB from 21.61 ± 8.02 to 27.80 ± 11.76 . These changes are consistent with the intended role of the mechanism: to reduce local-search myopia under strict subnet limits.

It is important to state precisely what this mechanism does. Frontier-directed path expansion does not add executable shortcut edges to the environment and does not change the primitive action semantics. The agent still moves through the environment using ordinary `move` actions and must satisfy the usual adjacency, stochastic movement, and budget conditions. The mechanism only affects the local solving view by including known paths toward high-frontier nodes. In other words, it changes which internal structure is visible to local MCTS, not what the agent can physically execute.

This distinction addresses the concern that frontier nodes may already be reachable in the internal graph. The issue is not reachability in the full internal knowledge graph, but visibility inside a bounded local subnet. A high-value frontier node may be reachable from the current location while still being excluded from a strict local subnet due to size limits. Frontier-directed path expansion helps preserve directions toward deeper or more informative regions without modifying the real environment dynamics.

At the same time, the ablation should not be overstated. The results support the usefulness of the current mechanism under the tested settings, but they do not prove that the particular frontier score, path-inclusion rule, or threshold choices are optimal. A more complete study could compare alternative frontier scores, different path-depth limits, and different ways of combining frontier value with reachability and database belief.

12.4. Progress-Aware Decision Making

The progress-weight and aggregation-profile sensitivity experiments support the role of progress-aware decision making. When the progress term is removed, successful runs can still be fast and budget-efficient, but the success rate drops from 12/12 under the default setting to 9/12 for $w_g = 0.00$. This pattern is informative. It suggests that progress is not required for every successful episode, but it improves robustness across stochastic runs by providing guidance before terminal-success estimates become reliable.

The successful $w_g = 0.00$ runs have strong step and budget values, which shows that removing progress does not necessarily make the planner inefficient in favourable cases. However, the lower success rate indicates that without progress guidance the planner is more vulnerable to early misleading decisions. The progress score should therefore be interpreted as a stabilising belief-state potential, not as an additional task objective and not as a guarantee of better performance in every run.

The higher-progress setting, $w_g = 0.16$, also performs worse than the default in success rate. This suggests that progress guidance is useful only when balanced against predicted success, cost, risk, and reachability. If progress is over-emphasised, the planner may continue to value intermediate improvements even when terminal completion should become the priority. The role of progress is therefore phase-dependent: it is most useful when terminal success is sparse or poorly estimated, but it should not dominate once a credible terminal opportunity appears.

12. Discussion

The aggregation-profile experiment gives a compatible result. The progress-aware default profile achieves 14/16 successes, while the conservative and success-oriented profiles achieve 12/16 and 11/16 successes respectively. The success-oriented profile performs worst in this tested subset. This suggests that predicted terminal success can be misleading when the planner has not yet resolved enough structure, role belief, or capability information. Progress-aware aggregation helps avoid premature commitment to terminal-looking options whose success estimates may still be unreliable.

Overall, these results support progress as a practical dense guidance signal. They do not establish a theoretical optimality guarantee, and they do not show that the exact progress weight is globally optimal. They do show that, in the tested experimental regime, progress-aware decision making improves robustness relative to both removing the progress term and using a more success-oriented aggregation profile.

12.5. Probability Gate and Search-Phase Transition

The probability-gate sensitivity experiment shows that the default p_{gate} value is important in the tested subset. The default setting succeeds in 12/12 runs, while $p_{\text{gate}} = 0.00$ succeeds in 7/12 runs and $p_{\text{gate}} = 0.10$ succeeds in 8/12 runs.

The probability gate functions as a transition mechanism between progress-guided and success-guided filtering. Early in an episode, success predictions are often weak or noisy because the planner has limited information about the graph, node roles, identities, and capabilities. In this phase, progress-based guidance can help the planner select actions that reveal useful structure or improve future success conditions. Later, when local options begin to have stronger predicted success, the planner should become more success-oriented.

The observed results are consistent with this interpretation. If the gate is too low, the planner can treat weak terminal-success estimates as meaningful too early. If the gate is too high, it can remain progress-oriented even after plausible terminal opportunities have appeared. The default value appears to provide a useful compromise for the tested setting. As with other parameters, this should be interpreted as empirical support for the chosen default in the present experimental regime, not as proof of a generally optimal threshold.

This also clarifies the relationship between progress score and terminal success. Progress is most useful when success estimates are not yet reliable. Once success estimates become informative, the planner should increasingly prioritise terminal completion. The probability gate is the current implementation mechanism that mediates this transition.

12.6. Reusable Local Strategy Learning

The same-type transfer results provide the strongest empirical support for the generalized policy library. The claim should still be stated carefully. The framework does not learn a complete general policy over

12.7. Same-Type versus Cross-Type Transfer

all possible networks. Instead, it learns that under a local structural signature of a certain kind, some abstract action templates tend to be useful.

This weaker claim is well supported by the same-type transfer results. On **small_sparse**, success remains 10/10 while success step decreases from 27.70 ± 7.66 to 21.00 ± 5.93 and remaining budget increases from 27.66 ± 12.79 to 35.20 ± 14.38 . On **medium_medium**, success remains 10/10 while success step decreases from 30.50 ± 8.72 to 23.70 ± 11.76 and remaining budget increases from 23.90 ± 14.45 to 34.44 ± 17.79 . On **large_dense**, same-type transfer increases success from 7/10 to 10/10 and reduces success step from 23.43 ± 10.49 to 19.20 ± 7.48 .

The distribution plots in Chapter 11 also show that the transfer effect is visible at the run level. The success-step distribution shifts downward under same-type transfer, and the remaining-budget distribution shifts upward. This supports the interpretation that the generalized policy library provides reusable local decision priors in structurally similar contexts, rather than merely improving a small number of average values.

The stage indicators reported in Table 11.2 further suggest that same-type transfer affects more than the final query phase. It improves first-*C* and first-DB timing in the reported held-out comparisons, indicating that the library provides useful directional guidance during intermediate movement, exploitation, and capability-building stages.

However, the same-type transfer results should not be interpreted as evidence of universal generalisation. The source and target instances share the same structural bucket. The evidence supports transfer across similar local structures, not across arbitrary graph families. The library should therefore be described as learning reusable local strategy priors for structurally related settings, rather than a complete domain-general planning policy.

12.7. Same-Type versus Cross-Type Transfer

The contrast between same-type and cross-type transfer is one of the most informative outcomes. Same-type transfer is strong and stable in the reported held-out comparisons. Cross-type transfer is useful but unstable.

In the cross-type transfer experiment, the library learned from **small_sparse** and **medium_medium** does not improve success rate on **large_dense**. Both **Subnet-Empty** and **Subnet-CrossTransfer** achieve 20/25 successes. However, cross-type transfer slightly reduces success step, from 26.65 ± 10.12 to 24.85 ± 10.15 , and reduces runtime, from 2272.05 ± 1416.45 to 1520.95 ± 1120.15 . The stage indicators reported in Table 11.3 also show earlier first-*C* and first-DB timing under cross-type transfer.

The correct conclusion is therefore mixed. Cross-type transfer provides useful directional bias and reduces some planning overhead, but it is not sufficiently reliable to improve terminal success probability. It helps the planner move toward relevant regions earlier in some runs, but this does not consistently translate into successful database query completion. It also does not improve remaining budget in the main summary, so the cross-type benefit should not be described as a broad efficiency improvement across all metrics.

12. Discussion

One interpretation, consistent with these results, is that the current structural signatures are sufficiently informative for same-type reuse but too coarse for reliable cross-type transfer. A signature that is useful in sparse or medium networks may correspond to a different local dynamic in dense networks. In addition, the library stores abstract first-action priors rather than complete multi-step local policies. If a transferred first action is only partially appropriate, recovery still depends on local solving and online feedback.

This explanation is plausible, but it is not directly isolated by the current experiments. The cross-type results diagnose a limitation of the current abstraction and retrieval scheme, but they do not by themselves prove which component is responsible. Future work should isolate the effects of signature design, template representation, retrieval criteria, and negative feedback mechanisms on cross-type transfer stability.

12.8. Computation-Time Trade-Off

The experiments consistently show that the proposed framework has substantial wall-clock overhead. This overhead arises from strict subnet construction, repeated budgeted MCTS calls, approximate non-dominated option generation, global candidate aggregation, and feedback-based rebuilding. In the main comparison, pure MCTS is much faster than the subnet-based framework across all buckets, even when runtime is plotted on a logarithmic scale.

This computational cost does not invalidate the interaction-level results, but it limits the current practical scope of the method. The framework is most defensible in settings where action cost, budget consumption, or the risk of failed interaction is more important than wall-clock planning time. If rapid online response is the primary requirement, the current implementation is not yet competitive.

The MCTS-budget sensitivity experiment illustrates this trade-off. The low-budget setting is cheaper but less robust, while the high-budget setting improves some successful-run metrics at higher computational cost. The default budget is therefore a practical compromise within the tested range. This result suggests that future work should focus not only on better planning logic but also on reducing local-solver cost through caching, incremental subnet updates, parallel rollout evaluation, or learned value approximations.

12.9. Interpretation of Sensitivity Experiments

The ablation and sensitivity experiments should be interpreted as diagnostic studies rather than as full parameter optimisation. They show that several major design choices have observable behavioural effects under representative local changes. In particular, frontier-directed path expansion, progress-aware aggregation, probability gating, and the default local-search budget are supported by the tested comparisons.

However, these experiments do not prove that the selected parameter values are globally optimal. The framework contains many coupled parameters, including subnet size, MCTS rollout budget, progress

weight, probability gate, aggregation weights, frontier scoring parameters, and candidate-adjustment terms. Changing one parameter can alter the effect of another. For example, increasing subnet size may require more MCTS budget; changing the progress weight may interact with the probability gate; and changing aggregation weights may alter how often library candidates are selected.

The current experiments address the concern that the framework contains several manually selected parameters. They show that important choices have meaningful behavioural effects and that the selected defaults are reasonable within the tested ranges. Nevertheless, the results should be presented as support for reasonable default settings in the present experimental regime, not as evidence of optimal hyperparameters.

A more complete treatment would require either a larger grid of parameter combinations or an adaptive parameter-selection mechanism. Such an extension would be a separate research problem because it would require defining a tuning objective over multiple metrics, including success rate, interaction step count, remaining budget, runtime, and transfer stability.

12.10. Experimental Limitations

Several limitations should be made explicit.

First, the number of graph instances is finite. The main comparison uses five instances per bucket and five seeds per instance, which is sufficient to show the main empirical patterns in this thesis but not enough to make strong population-level claims about all possible network structures.

Second, the same-type transfer evaluation uses held-out instances within each bucket, but the number of held-out instances is still limited. More held-out instances would be needed to make stronger statistical claims about transfer stability.

Third, cross-type transfer is evaluated only in the direction from `small_sparse` and `medium_medium` to `large_dense`. A complete transfer matrix across all source–target combinations remains future work.

Fourth, the ablation and sensitivity experiments are diagnostic rather than exhaustive. They test local changes to important parameters and mechanisms, but they do not perform full hyperparameter optimisation.

Fifth, several components remain coupled in the current evaluation. For example, strict subnet solving and approximate non-dominated local option generation are evaluated together in the main comparison. A more isolated ablation would be required to separate their individual effects completely.

Sixth, the framework currently has significant wall-clock overhead. Strict subnet construction, budget sweeps, local MCTS, option generation, and rebuilding are expensive. Reducing this cost is necessary before the framework can be positioned as computationally efficient.

Seventh, although the results report standard deviations and include distributional plots for the main comparisons, the thesis does not provide distributional plots for every recorded metric. The reported plots support the main empirical claims, while the full numerical summaries are provided in Chapter 11, especially Tables 11.5 and 11.6.

12. Discussion

Finally, the experiments are conducted in a simulated network-penetration domain. This is appropriate for controlled evaluation of partial observability, object discovery, identity association, and budgeted on-line planning, but it does not capture all operational details of real cyber environments. The conclusions should therefore be interpreted as evidence about the proposed planning abstraction, not as a direct claim of deployment readiness.

12.11. Chapter Summary

The discussion leads to a more precise interpretation of the thesis contribution. The proposed framework is not a uniformly superior replacement for pure MCTS. Its current strength lies in improving interaction-level efficiency in successful runs and enabling reusable local strategic guidance. Strict subnet solving can improve local decision quality when the selected local view contains relevant structure; frontier-directed path expansion reduces local-search myopia under strict subnet limits; progress-aware aggregation stabilises early decision making; the probability gate mediates the transition from progress-oriented to success-oriented filtering; and the generalized policy library provides effective same-type transfer.

At the same time, several limitations remain. The framework has higher wall-clock overhead than pure MCTS, cross-type transfer is only partially effective, and the sensitivity experiments support reasonable default settings rather than globally optimal hyperparameters. Overall, the thesis demonstrates a useful structured planning framework for partially observable online planning with object discovery, while also identifying the main issues that must be addressed before the approach can become more broadly robust and computationally efficient.

Conclusion

This thesis studied a class of budget-constrained online planning problems characterised by partial observability, dynamic object discovery, and initially unstable identities. The problem was instantiated in a simulated network-penetration domain for formal modelling and empirical evaluation. In contrast to static attack-path search, the setting considered in this thesis assumes that the planner does not initially know the full environment structure, the true roles of objects, or their stable identities. Instead, it must interact with the environment online, progressively construct an internal knowledge graph, revise local beliefs, and achieve true target verification under a limited budget.

To address this problem, the thesis proposed an integrated online planning framework consisting of an object-level knowledge graph, strict-subnet local solving, approximate non-dominated local option generation, abstract action templates, generalized policy-library reuse, and online model revision. The framework separates the hidden real environment state from the planner’s internal knowledge state, allowing the internal representation to grow, be revised, and be reorganised through observation and identity confirmation. The planner then uses this internal graph to construct local solving domains, generate local trade-off options, retrieve reusable action templates from previous experience, and select executable actions through a global aggregation layer.

The experimental results show that the proposed framework should not be interpreted as a uniformly superior replacement for pure MCTS. Pure MCTS remains stronger in raw success robustness and wall-clock runtime, achieving 25/25 successes in all three main buckets and requiring substantially less computation time. The main advantage of the proposed framework lies instead in interaction-level efficiency. When the subnet-based planner succeeds, it typically reaches terminal success in fewer action steps and preserves more remaining budget than pure MCTS. This indicates that strict local solving and structured local option generation can help the planner use execution resources more economically, even though they introduce additional computational overhead and do not always improve terminal success rate.

The same-type transfer results provide the strongest evidence for the usefulness of the generalized policy library. On held-out instances from the same structural bucket, policy-library reuse consistently reduces success step, improves remaining budget, advances first entry into key structural layers, and reduces runtime. In the `large_dense` held-out setting, same-type transfer increases success from 7/10 to 10/10. These results support the claim that the framework can extract reusable local action priors from previous experience. The learned object is not a complete general policy over all possible networks, but a structure-indexed library of abstract first-action templates with associated success, progress, and cost statistics.

13. Conclusion

The cross-type transfer results are more limited. When a library learned from `small_sparse` and `medium_medium` is applied to `large_dense`, the success rate remains the same as the empty-library subnet baseline, with both settings achieving 20/25 successes. Nevertheless, cross-type transfer improves several measured indicators, including success step, first-*C*, first-DB, and runtime. This suggests that the current abstraction provides some directional guidance across structural types, but it is not yet reliable enough to support a strong claim of general policy learning. The most appropriate conclusion is therefore that the thesis demonstrates local strategy abstraction and reusable decision priors, with stable benefits in same-type transfer and weaker, less stable benefits in cross-type transfer.

The ablation and sensitivity experiments further clarify the role of several design choices. Removing frontier-directed path expansion reduces success rate, delays database-layer entry, increases success step, and leaves less remaining budget. This supports the interpretation that frontier-directed path expansion is a useful local-domain construction mechanism rather than an executable shortcut. Subnet-size sensitivity shows that the default subnet size performs favourably among the tested settings for the current problem and network scales. MCTS-budget sensitivity shows that more rollout computation can improve some successful-run metrics, but at higher runtime cost. Progress-weight and aggregation-profile sensitivity show that progress-aware decision making stabilises the planner before reliable terminal-success estimates are available. Probability-gate sensitivity shows that the default gate provides a useful transition between progress-guided and success-guided local filtering. These experiments should be interpreted as diagnostic support for the chosen design, not as exhaustive hyperparameter optimisation or evidence of globally optimal parameter values.

Overall, the contributions of the thesis can be summarised in four points. First, it formulates and studies an online planning problem with partial observability, dynamic object discovery, and initially unstable identities. Second, it proposes an object-centric knowledge-graph representation that separates true environment state from internal planning state and supports identity association, role beliefs, and edge-reliability estimates. Third, it develops a strict-subnet local planning mechanism that generates approximate non-dominated local options over predicted success, progress, and cost. Fourth, it introduces an action-template abstraction and generalized policy library that reuse local first-action priors across episodes and improve same-type transfer performance.

Several limitations remain. The number of graph instances and random seeds is sufficient to show the main empirical patterns in this thesis, but not enough to make strong population-level claims about all possible network structures. The ablation and sensitivity experiments examine important local changes, but they do not constitute a full hyperparameter-optimisation study. Several components, such as strict subnet solving and approximate non-dominated option generation, are still evaluated together rather than fully isolated. The framework also has significant wall-clock overhead, mainly due to subnet construction, budget sweeps, local MCTS, and rebuilding. Finally, cross-type transfer remains unstable, indicating that the current structural signatures and first-action templates are not yet expressive enough for reliable generalisation across substantially different network families.

Future work should address these limitations in several directions. First, the structural signature and template representation should be refined to improve cross-type transfer reliability. Second, additional ablation experiments should isolate components such as identity confirmation, traffic sensing, slow forgetting, trigger-based rebuilding, and policy-library update rules. Third, the computational overhead of

strict-subnet solving and budget sweeps should be reduced, possibly through better caching, selective recomputation, parallel rollout evaluation, or learned local value approximators. Fourth, the transfer evaluation should be expanded into a fuller source–target matrix across all network buckets. Finally, future work could investigate adaptive or learned parameter selection, so that progress weights, probability gates, aggregation profiles, and frontier-scoring parameters are adjusted automatically rather than manually selected.

In summary, this thesis demonstrates that combining knowledge-graph-based internal state representation, strict local subnet solving, approximate non-dominated option organisation, and reusable action-template priors can help an online planner construct, abstract, reuse, and revise local strategic experience in partially observable environments where objects, identities, and useful structure are discovered progressively during interaction. The framework is not yet computationally efficient enough to replace pure MCTS as a fast baseline, and its cross-type transfer remains only partially reliable. Nevertheless, it provides a concrete step toward structured online planning methods for environments with dynamic object discovery and evolving object identity.

Progress Score Implementation Details

A.1. Implementation Form of the Progress Score

This appendix gives the implementation-level form of the progress score introduced in Section 7.8. The main text interprets the progress score as a belief-state potential used to provide dense guidance when terminal success is sparse. In the implementation, this score is computed from the internal state I_t , not from the hidden true environment state.

The implemented progress score has the form

$$g(I_t) = w_{\text{pot}}g_{\text{pot}}(I_t) + w_{\text{exp}}g_{\text{exp}}(I_t) + w_{\text{unc}}g_{\text{unc}}(I_t) + w_{\text{cap}}g_{\text{cap}}(I_t), \quad (\text{A.1})$$

where g_{pot} is the distance-discounted database-potential term, g_{exp} is the discovered-structure term, g_{unc} is the local uncertainty term, and g_{cap} is the capability term. The weights w_{pot} , w_{exp} , w_{unc} , and w_{cap} are planner parameters. The final value is clipped to the interval $[0, 1]$.

A.1.1. Database-Potential Term

Let D_t be the set of internal nodes reached by a capped Dijkstra search from the current internal location $\hat{\ell}_t$ in the internal graph $\hat{G}_t$. For each reachable internal node $v \in D_t$, define its distance-discounted database potential as

$$q_D(v \mid I_t) = b_v(DB) \exp\left(-\frac{d_t(\hat{\ell}_t, v)}{\tau_D}\right), \quad (\text{A.2})$$

where $b_v(DB)$ is the current database-role belief of node v , $d_t(\hat{\ell}_t, v)$ is the estimated internal movement distance from the current location to v , and τ_D controls distance discounting.

The database-potential term is then

$$g_{\text{pot}}(I_t) = \max\left(\max_{v \in D_t} q_D(v \mid I_t), \lambda_{\text{mean}} \text{mean}_{v \in D_t} q_D(v \mid I_t)\right). \quad (\text{A.3})$$

The first term favours the single most promising reachable database-like node, while the second term gives a weaker regional signal when several nodes have moderate database belief. The coefficient $\lambda_{\text{mean}} \in [0, 1]$ controls how much this regional average contributes. In the current implementation, $\lambda_{\text{mean}} = 0.35$.

A. Progress Score Implementation Details

A.1.2. Discovered-Structure Term

The discovered-structure term is a saturating function of the current internal graph size:

$$g_{\text{exp}}(I_t) = \alpha_E \left(1 - \exp \left(-\frac{|\hat{V}_t| - 1}{\tau_V} \right) \right) + (1 - \alpha_E) \left(1 - \exp \left(-\frac{|\hat{E}_t^{\text{und}}|}{\tau_E} \right) \right). \quad (\text{A.4})$$

Here $|\hat{V}_t|$ is the number of currently represented internal nodes, and $|\hat{E}_t^{\text{und}}|$ is the number of undirected internal edge pairs derived from the directed internal edge set. The constants τ_V and τ_E control the saturation rates for node and edge discovery respectively. The coefficient $\alpha_E \in [0, 1]$ controls the relative contribution of discovered nodes and discovered edges. In the implementation, $\alpha_E = 0.60$, so node discovery receives slightly higher weight than edge discovery.

This term is intended to reward useful structural discovery early in an episode, while preventing repeated discovery from dominating the progress score indefinitely.

A.1.3. Local Uncertainty Term

The uncertainty term is based on the role-belief entropy at the current internal location:

$$g_{\text{unc}}(I_t) = \frac{H(b_{\hat{\ell}_t})}{H(b_{\hat{\ell}_t}) + 1} \cdot \frac{1}{1 + \text{scanCount}(\hat{\ell}_t)}. \quad (\text{A.5})$$

Here $H(b_{\hat{\ell}_t})$ is the entropy of the role-belief distribution at the current internal node. The first factor increases when the current node is semantically uncertain. The second factor reduces the value of repeated scanning at the same node, preventing the planner from repeatedly collecting low-value information from one location.

This term should not be interpreted as an exact information-gain calculation. It is a lightweight local uncertainty signal used by the planner to distinguish states that may still benefit from information gathering.

A.1.4. Capability Term

The capability term represents whether the planner has obtained conditions that make terminal database verification more likely:

$$g_{\text{cap}}(I_t) = \alpha_\kappa \hat{\kappa}_t + \alpha_\pi \min \left(1, \frac{\hat{\pi}_t(\hat{\ell}_t)}{\pi_{\text{max}}} \right), \quad \alpha_\kappa + \alpha_\pi = 1. \quad (\text{A.6})$$

Here $\hat{\kappa}_t$ is the internal credential flag, $\hat{\pi}_t(\hat{\ell}_t)$ is the privilege estimate at the current internal location, and π_{max} is the privilege normalisation constant. In the implementation, $\alpha_\kappa = 0.65$, $\alpha_\pi = 0.35$, and $\pi_{\text{max}} = 2$. This gives slightly higher weight to credentials than to local privilege, reflecting the domain assumption that reaching a database-like node is not sufficient if the agent lacks the capability required for successful `query_db`.

A.1.5. Role in the Planner

The progress score is used as an internal guidance signal rather than as a replacement for the terminal task objective. It appears in local evaluation, option prediction, global aggregation, action-prior feedback, and execution diagnostics. For example, execution records store the observed post-action progress value, and action-prior feedback can reward actions that increase progress when they do not immediately achieve terminal success.

The progress score therefore acts as a practical belief-state value approximation. It helps distinguish locally promising states when terminal success is sparse, but it is not claimed to define a policy-invariant reward transformation. The sensitivity experiments in Chapter 11 evaluate the effect of changing the progress weight in the global aggregation stage.

Bibliography

- AKL, S. G. AND NEWBORN, M. M., 1977. The principal continuation and the killer heuristic. *Proceedings of the 1977 Annual Conference of the ACM*, (1977), 466–473. [Cited on page 83.]
- BERCHER, P.; ALFORD, R.; AND HÖLLER, D., 2019. A survey on hierarchical planning: One abstract idea, many concrete realizations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 6267–6275. International Joint Conferences on Artificial Intelligence. doi:10.24963/ijcai.2019/872. [Cited on pages 16, 53, and 77.]
- BONET, B.; FRANCÈS, G.; AND GEFFNER, H., 2019. Learning features and abstract actions for computing generalized plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2703–2710. doi:10.1609/aaai.v33i01.33012703. [Cited on pages 16 and 72.]
- BONET, B. AND GEFFNER, H., 2018. Features, projections, and representation change for generalized planning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 4667–4673. International Joint Conferences on Artificial Intelligence. doi:10.24963/ijcai.2018/648. [Cited on pages 16 and 72.]
- BOUTILIER, C.; REITER, R.; AND PRICE, B., 2001. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 690–697. Morgan Kaufmann. [Cited on pages 9, 16, and 37.]
- BROWNE, C. B.; POWLEY, E.; WHITEHOUSE, D.; LUCAS, S. M.; COWLING, P. I.; ROHLFSHAGEN, P.; TAVENER, S.; PEREZ, D.; SAMOTHRAKIS, S.; AND COLTON, S., 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4, 1 (2012), 1–43. doi:10.1109/TCIAIG.2012.2186810. [Cited on pages 4, 11, 53, and 61.]
- CIMATTI, A.; PISTORE, M.; ROVERI, M.; AND TRAVERSO, P., 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147, 1–2 (2003), 35–84. doi:10.1016/S0004-3702(02)00374-0. [Cited on page 18.]
- DIUK, C.; COHEN, A.; AND LITTMAN, M. L., 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, 240–247. ACM. doi:10.1145/1390156.1390187. [Cited on pages 9, 16, 21, and 37.]
- EHRGOTT, M., 2005. *Multicriteria Optimization*. Springer, Berlin, Heidelberg, 2 edn. ISBN 9783540213987. doi:10.1007/3-540-27659-9. [Cited on pages 2, 11, 12, 18, 67, and 82.]

Bibliography

- FELLEGI, I. P. AND SUNTER, A. B., 1969. A theory for record linkage. *Journal of the American Statistical Association*, 64, 328 (1969), 1183–1210. doi:10.1080/01621459.1969.10501049. [Cited on pages 10, 26, and 41.]
- GHALLAB, M.; NAU, D.; AND TRAVERSO, P., 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann, San Francisco, CA. ISBN 9781558608566. [Cited on pages 1, 2, 8, and 9.]
- HOGAN, A.; BLOMQUIST, E.; COCHEZ, M.; D’AMATO, C.; DE MELO, G.; GUTIERREZ, C.; KIRrane, S.; LABRA GAYO, J. E.; NAVIGLI, R.; NEUMAIER, S.; NGOMO, A.-C. N.; POLLERES, A.; RASHID, S. M.; RULA, A.; SCHMELZEISEN, L.; SEQUEDA, J.; STAAB, S.; AND ZIMMERMANN, A., 2021. Knowledge graphs. *ACM Computing Surveys*, 54, 4 (2021), 1–37. doi:10.1145/3447772. [Cited on pages 3, 9, 23, and 37.]
- HOWARD, R. A., 1966. Information value theory. *IEEE Transactions on Systems Science and Cybernetics*, 2, 1 (1966), 22–26. doi:10.1109/TSSC.1966.300074. [Cited on page 55.]
- JACCARD, P., 1901. Étude comparative de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37 (1901), 547–579. [Cited on page 42.]
- KAEHLING, L. P.; LITTMAN, M. L.; AND CASSANDRA, A. R., 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1–2 (1998), 99–134. doi:10.1016/S0004-3702(98)00023-X. [Cited on pages 1, 8, 15, 21, 37, 47, and 77.]
- KOCSIS, L. AND SZEPESVÁRI, C., 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, vol. 4212 of *Lecture Notes in Computer Science*, 282–293. Springer. doi:10.1007/11871842_29. [Cited on pages 4, 11, and 61.]
- MAVROTAS, G., 2009. Effective implementation of the ϵ -constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213, 2 (2009), 455–465. doi:10.1016/j.amc.2009.03.037. [Cited on pages 4, 12, 18, and 63.]
- NG, A. Y.; HARADA, D.; AND RUSSELL, S. J., 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 278–287. Morgan Kaufmann. [Cited on page 62.]
- OU, X.; BOYER, W. F.; AND MCQUEEN, M. A., 2006. Scalable attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 336–345. ACM. doi:10.1145/1180405.1180446. [Cited on pages 1 and 17.]
- OU, X.; GOVINDAVAJHALA, S.; AND APPEL, A. W., 2005. MulVAL: A logic-based network security analyzer. In *Proceedings of the 14th USENIX Security Symposium*, 113–128. USENIX Association. [Cited on pages 1, 2, and 17.]
- PHILLIPS, C. AND SWILER, L. P., 1998. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*, 71–79. ACM. [Cited on pages 1 and 17.]

- POOLSAPPASIT, N.; DEWRI, R.; AND RAY, I., 2012. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9, 1 (2012), 61–74. doi:10.1109/TDSC.2011.34. [Cited on pages 1 and 17.]
- PUTERMAN, M. L., 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York. ISBN 9780471619772. doi:10.1002/9780470316887. [Cited on page 7.]
- ROIJERS, D. M.; VAMPLEW, P.; WHITESON, S.; AND DAZELEY, R., 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48 (2013), 67–113. doi:10.1613/jair.3987. [Cited on pages 18, 34, and 82.]
- SARRAUTE, C.; BUFFET, O.; AND HOFFMANN, J., 2012. POMDPs make better hackers: Accounting for uncertainty in penetration testing. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 1816–1824. AAAI Press. [Cited on pages 17, 21, 47, and 89.]
- SARRAUTE, C.; BUFFET, O.; AND HOFFMANN, J., 2013. Penetration testing == POMDP solving? [Cited on page 17.]
- SHEYNER, O.; HAINES, J.; JHA, S.; LIPPMANN, R.; AND WING, J. M., 2002. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 273–284. IEEE. doi:10.1109/SECPRI.2002.1004377. [Cited on pages 1, 2, and 17.]
- SILVER, D. AND VENESE, J., 2010. Monte-carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23*, 2164–2172. Curran Associates, Inc. [Cited on pages 15, 47, 53, and 77.]
- SUTTON, R. S. AND BARTO, A. G., 2018. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edn. ISBN 9780262039246. [Cited on page 7.]
- SUTTON, R. S.; PRECUP, D.; AND SINGH, S., 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112, 1–2 (1999), 181–211. doi:10.1016/S0004-3702(99)00052-1. [Cited on pages 3, 12, 16, 19, and 67.]
- WANG, L.; SINGHAL, A.; AND JAJODIA, S., 2007. Measuring the overall security of network configurations using attack graphs. In *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 98–112. Springer. doi:10.1007/978-3-540-73538-0_8. [Cited on pages 1, 2, and 17.]
- YAMAUCHI, B., 1997. A frontier-based approach for autonomous exploration. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 146–151. IEEE. doi:10.1109/CIRA.1997.613851. [Cited on pages 1 and 55.]
- YE, N.; SOMANI, A.; HSU, D.; AND LEE, W. S., 2017. DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research*, 58 (2017), 231–266. doi:10.1613/jair.5328. [Cited on pages 15, 53, and 77.]
- YOON, S. W.; FERN, A.; AND GIVAN, R., 2002. Inductive policy selection for first-order mdps. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, 568–576. [Cited on page 19.]