Proceedings of the ASME 2025
International Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
IDETC-CIE2025
August 17-20, 2025, Anaheim, CA

DETC2025-168784

EXTRACTING DESIGN PROCESS HEURISTICS: A SYSTEMATIC COMPUTATIONAL APPROACH

Vikranth S. Gadi
School of Mechanical Engineering
Purdue University
West Lafayette, Indiana 47907

Zoe Szajnfarber
Engineering Management and
Systems Engineering
The George Washington University
Washington, DC 20052

Jitesh H. Panchal
School of Mechanical Engineering
Purdue University
West Lafayette, Indiana 47907

ABSTRACT

Engineering design relies heavily on heuristics, yet there is a lack of systematic methods for identifying and validating design heuristics. This paper introduces a computational approach to representing engineering design problems that involve decomposition and assignment decisions, facilitating systematic extraction of generalizable heuristics. We model design processes using a Markov Decision Process (MDP) framework, characterizing problems through attributes of the problem space, solver capabilities, and trade-offs embedded within preference functions. Reinforcement learning methods are employed to learn optimal policies, from which we extract inclusionary and exclusionary heuristics using Gaussian Mixture Models. The effectiveness of the approach is demonstrated through two case studies: solver-aware system architecting (SASA) for a robotic arm design and sequential information acquisition in parametric design optimization. The results highlight the context-dependent nature of learned heuristics, demonstrating how problem complexity, designer preferences, and solver characteristics influence their selection.

Keywords: Design heuristics, decision making, reinforcement learning, systems engineering

1. INTRODUCTION

Engineering design problems are inherently complex, ill-structured, and characterized by uncertainty and resource constraints [1]. Designers navigate these complexities by employing heuristics, context-dependent directives based on intuition, tacit knowledge, or experiential understanding, to efficiently reach satisfactory solutions [2, 3]. Despite their widespread use and recognized importance in engineering design [4, 5], heuristics are often extracted informally or implicitly from specific contexts without a systematic approach to generalization [6].

In contrast, fields such artificial intelligence (AI) have formalized heuristics as principles guiding decision-making toward effective solutions in computationally demanding scenarios [7, 8]. Pearl's [7] research on heuristics informs that a clear, formal problem representation is essential for extracting effective heuristics, linking the success of heuristic methods directly to how well the problem is defined.

Within the engineering design literature, the lack of a shared representation of design problems across different problem contexts limits our ability to systematically extract and generalize heuristics. Existing research often focuses on specific systems or applications without explicitly defining the boundaries or characteristics of the broader class of problems being addressed. For example, heuristics have been identified for space mission design in NASA's Jet Propulsion Laboratory (JPL) [4], yet these heuristics are presented without clear delineation of the underlying problem representation or scope. As a result, it is challenging to generalize these heuristics beyond their original context or to formally describe the conditions under which they are valid.

In contrast, well-established computational problems such as the Traveling Salesman Problem (TSP) provide clear examples of how shared representations of the problems enable effective identification of heuristics. The TSP represents a class of network optimization problems with universally agreed-upon abstractions. Consequently, researchers can systematically develop and evaluate heuristics tailored to specific network structures, e.g., random networks versus small-world networks, and determine when a heuristic performs best [9–11].

We highlight that effective heuristic extraction requires not only clear problem representation but also identification of distinct classes of problems within which heuristics can reliably generalize. Without such formalism, heuristic extraction remains ad-hoc and context-specific, limiting progress toward systematic improvements in engineering design practice.

To address this gap, we propose a formal approach to rep-

TABLE 1: COMPARATIVE ANALYSIS OF HEURISTIC EXTRACTION APPROACHES IN ENGINEERING DESIGN

Study	Problem Representation	Heuristic Type	Generalizability
Yilmaz et al. (2011) [5]	Textual design briefs with implicit constraints	Cognitive strategies	Domain-specific (early-stage product design)
Fu et al. (2019) [4]	Mission requirement documents with parametric constraints	Procedural rules	Mission-specific (limited to NASA systems)
Deshmukh et al. (2016) [6]	Optimization formulations (e.g., min $f(x)$ s.t. $g(x) \leq 0$)	Constraint management	Algorithm-specific (dependent on numerical methods)
Pearl and Meisel (2024) [12]	Open-ended concept generation without manufacturing constraints	Manufacturing-based (TM vs. AM) heuristics	Limited to contexts with similar designer experience levels
Pearl and Meisel (2025) [13]	Student-generated design concepts evaluated against predefined TM and AM heuristics	DfM guidelines	Educational contexts requiring expert-guided interventions
Proposed Framework	Markov Decision Process (S, A, T, R, γ)	Policy mappings between problem classes and design processes	Class-based generalization across decomposition/assignment problems

resenting engineering design problems involving decomposition and assignment decisions that is applicable across various domains including parametric design optimization, engineering design, and systems engineering. We argue that explicitly defining such representations enables systematic extraction of generalizable heuristics. Specifically, we introduce a framework that characterizes design problems through attributes of the problem space (e.g., complexity, coupling), solver capabilities (e.g., expertise), and trade-offs embedded within preference functions (e.g., resource constraints). By adopting this structured representation, we identify new avenues for heuristic extraction analogous to those successfully used in algorithmic contexts [7, 8, 14].

The contributions of this paper are threefold:

1. a formal representation of engineering design processes involving decomposition and solver assignment decisions,
2. demonstration of how the structured representation enables systematic extraction of heuristics using reinforcement learning methods, and
3. illustration of how extracted heuristics generalize across multiple instances within clearly defined classes of engineering design problems.

To validate our approach, we analyze the existing literature on heuristic extraction in engineering design and identify limitations arising from informal or implicit problem representations as shown in Table 1. We further illustrate our framework through case studies involving robotic system architecting and sequential information acquisition in engineering design. These examples highlight how formalizing problem representations facilitates meaningful comparisons across diverse contexts demonstrating that clear abstractions not only facilitate heuristic extraction but also significantly enhance our ability to generalize these insights across broader classes of complex design problems.

2. LITERATURE REVIEW

Heuristics have been studied in several different domains. Within engineering design, design heuristics have been investigated from the perspective of creating novel design solutions while managing the complexity of the design process [2–6, 12, 13]. Researchers in cognitive psychology have studied heuristics used by human decision makers as deviations from rationality, and the resulting cognitive biases [15–20]. Within computer science and artificial intelligence communities, heuristics have been studied to develop efficient algorithms to solve hard computational problems [7–11, 14]. In this section, we provide a brief review of the literature in these three domains.

2.1 Heuristics in Engineering Design

Based on a thorough evaluation of the definitions of heuristics, Fu et al. [2] define heuristics as "context-dependent directives, based on intuition, tacit knowledge, or experiential understanding, which provides design process direction to increase the chance of reaching a satisfactory but not necessarily optimal solution."

Heuristics are used throughout the design process, including the creation of new design concepts during the conceptual stage and decision making in the later stages of design. Through a protocol study, Yilmaz et al. [3] found that heuristics are frequently used by engineers in different domains during conceptual design. Yilmaz et al. [5] show how the use of heuristics by expert designers in the early stages of product design leads to novel and creative solutions.

Heuristics also play an important role in early-stage systems design. By interviewing ten experts at the Jet Propulsion Laboratory (JPL), Fu and Paredis [4] identified 101 heuristics used in the design of space missions. In the later stages of design, such as in design optimization, heuristics are used to reduce the complexity of finding optimal designs. Deshmukh et al. [6] show that while these heuristics help manage design optimization

tasks, they can also lead to unnecessary constraints and cognitive biases. Pearl and Meisel [12] examined the influence of prior manufacturing experiences and heuristic priming on early stage engineering designs. Their findings indicated that novices naturally produced designs that are more suitable for traditional manufacturing (TM) than additive manufacturing (AM). However, greater familiarity with AM processes led to designs that benefited AM-specific heuristics, particularly complexity incorporation. Based on these results, Pearl and Meisel [13] assessed the manifestation of design-for-manufacturing (DfM) heuristics and found discrepancies between students' self-assessments of heuristic usage and expert evaluations, suggesting that students often overestimate their application of relevant heuristics. They highlighted specific DfM heuristics, notably "incorporating complex shapes" and "avoiding large, flat regions," significantly improving additive manufacturability. Their study emphasized the importance of structured interventions and explicit guidance on heuristics early in the design process to better align student heuristics with intended manufacturing methods.

2.2 Heuristics in Cognitive Psychology

Heuristics have been studied extensively in psychology as an alternative to rational decision making. Tversky and Kahneman [15] classify decision-making heuristics into representativeness, availability, adjustment, and anchoring. Heuristics are considered efficient cognitive processes, conscious or unconscious, that ignore part of the information [16]. Heuristics help in making decisions more quickly, frugally, and/or accurately than more complex methods.

In contrast to the view that heuristics are procedural simplifications from rational models, recent work in cognitive science suggests that the use of heuristics is rational if we account for the cognitive constraints [17, 18]. Lewis et al. [19] have coined the phrase "computational rationality" to emphasize that theories of rational behavior should account for computational and cognitive constraints, and the cost of cognitive effort [20].

2.3 Heuristics in Computer Science

Within the computer science (CS) and artificial intelligence (AI) literature, heuristics are used to solve complex computational problems, such as the traveling salesman problem. Pearl defines heuristics as "criteria, methods, or principles for deciding which among several alternate courses of action promises to be the most effective in order to achieve some goal" [7]. Several efforts have been devoted to developing good heuristics for specific computational problems.

Within AI research, heuristics are used to design computational agents that mimic human behavior. The use of heuristics is particularly prominent in the reinforcement learning literature, where agents use heuristic policies for learning, such as pure exploration, pure exploitation, excitation policies, epsilon-greedy exploration, Boltzmann exploration, Upper confidence bounding, and Thompson sampling [14].

In summary, the usefulness of heuristics is well recognized in each of these fields. It is also recognized that different heuristics have different effectiveness under different conditions, such as different resource constraints. However, there is a lack of normative understanding of heuristics, which can guide designers to choose the right heuristics considering the nature of the problem. Therefore, there is a need for approaches for determining the appropriateness of heuristics for the problems commonly encountered in engineering design. To address this need, we present a reinforcement learning-based approach to determine the heuristics that a resourced constrained rational agent should use. The approach is discussed in the following section.

3. UNIFIED METHODOLOGY FOR HEURISTIC EXTRACTION

Given the inherently ill-structured nature of design problems [1], designers use their expertise and knowledge to define and shape a unique problem space specific to each design task. This constructed problem space comprises the set of possible ideas and solutions accessible to a designer, serving as a cognitive framework for performing design activities [21]. By providing a mental structure, the design problem space aids designers in exploring, generating, and evolving ideas, thus helping them manage and navigate the inherent complexities associated with the design process [22]. Within this cognitive space, concepts emerge, undergo iterative refinement, and ultimately are embodied in tangible artifacts whose performance can be evaluated. The design process involves the exploration and navigation of the de-

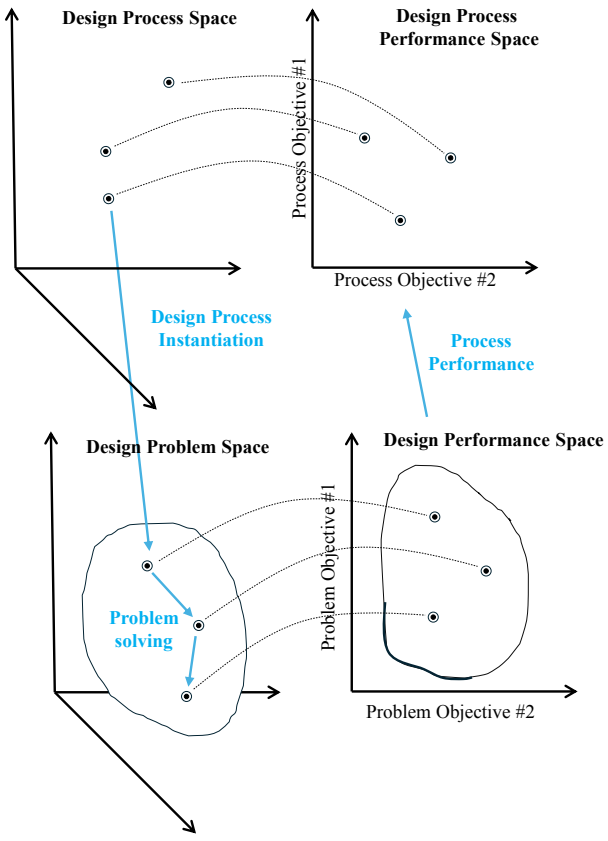

FIGURE 1: ILLUSTRATION OF DESIGN PROBLEM SPACE AND DESIGN PROCESS SPACE.

sign problem space. This navigation occurs through what Gero terms the Function-Behavior-Structure (FBS) framework [23], where designers move between different representations of the design problem while seeking solutions.

A design process can be conceptualized as a networked arrangement of activities. Each activity type serves specific functions within the broader design process, such as synthesis, analysis, evaluation, or reformulation. The design process space represents the collection of all possible design processes, which maps to a corresponding design process performance space. This space interacts with the design problem space, which in turn maps to the design performance space as shown in Figure 1. The relationship between these spaces helps to understand how different design processes lead to different outcomes.

To illustrate how the characteristics of a given problem influence the selection of solution approaches in practice, we examine two classes of problems that demonstrate this relationship– optimization problems and the Traveling Salesman Problems (TSP) which is a specific optimization problem.

Optimization problems can be solved by various approaches such as gradient-based optimization [24], gradient-free optimization [25], and surrogate-based optimization [26]. The characteristics of the problem at hand such as convexity, continuity, differentiability, presence of noise and cost of evaluation can help guide in the selection of an approach. Determining whether a design optimization problem is convex is often not straightforward. However, experience can help designers identify situations in which the problem can be effectively reformulated into a convex form. Typically, convexity arises in cases characterized by straightforward objective functions and constraints such as in control applications where optimization is performed repeatedly [27]. Although general gradient-based or gradient-free techniques can solve convex problems, neglecting to explicitly leverage the structure of a convex formulation is inefficient. Next, it is important to consider whether the underlying model is continuous and differentiable, or can be made smooth through model improvements. For large-scale design problems (typically involving tens or more design variables), gradient-free optimization methods generally become computationally prohibitive. In such cases, algorithms that rely on gradient computations are usually recommended [28]. Consequently, it may be necessary to enhance model smoothness to facilitate gradient-based approaches or otherwise reduce dimensionality if gradient-free methods are to be employed. When problems lack differentiability or involve noisy and computationally expensive evaluations, surrogate-based optimization methods offer an alternative way forward [26]. If we choose the surrogate-based optimization, we can still use a gradient-based approach to optimize the surrogate model because most such models are differentiable. Finally, for problems with a relatively small number of design variables, gradient-free methods can be a good fit [28].

Another example where the problem type can help guide design process is the Travelling Salesman Problem [9]. In the context of the Traveling Salesman Problem (TSP), design process selection refers to choosing the most appropriate algorithm or heuristic method to solve the problem based on specific characteristics and constraints. Solutions to TSP can be broadly classified as construction heuristics [29] such as nearest neighbors and insertion methods, improvement heuristics [30] such as 2-opt, 3-opt and Lin-Kernighan [31] and meta-heuristic approaches such as genetic algorithms [32], ant colony [11] and particle swarm optimization [33]. Considerations such as problem structure, size, and the desired balance between speed and solution quality can guide practitioners to choose the most appropriate method for their particular application. In Euclidean TSPs, where distances satisfy the triangle inequality, simpler construction heuristics like nearest neighbor or insertion methods can provide reasonable initial solutions quickly [29]. For small to moderate instances (typically fewer than 100 cities), simple improvement heuristics such as 2-opt or 3-opt can effectively refine initial solutions to near-optimality [10]. However, as the size of the problem grows, the computational cost of examining all possible edge exchanges becomes prohibitive, necessitating other approaches. When dealing with larger instances, metaheuristic approaches, such as genetic algorithms or ant colony optimization become more attractive despite their higher complexity [11]. These methods can effectively navigate vast solution spaces by maintaining population diversity and avoiding premature convergence to local optima. However, they require careful parameter tuning and typically demand more computational resources than simpler heuristics. The available computational budget and solution quality requirements also play crucial roles in algorithm selection. When quick solutions are needed, construction heuristics provide fast results, although potentially suboptimal. For applications requiring higher quality solutions and allowing longer computation times, hybrid approaches combining multiple heuristics often prove to be the most effective [9].

The effectiveness of a particular design process is inherently linked to the characteristics of the design problem it aims to solve, as illustrated in the cases of the optimization problem and the Traveling Salesman Problem. The alignment between process capabilities and problem requirements is therefore crucial for achieving optimal outcomes. This relationship suggests that certain design processes are more appropriately suited for specific classes of design problems, based on factors such as the complexity of the problem, the types of constraints, and the desired outcome characteristics. In this section, we describe a framework for understanding how design processes can be selected based on the specific requirements and characteristics of the design problem at hand.

3.1 Modeling Design Problems

A design problem can be mathematically represented as P, where the objective is to find a solution X within the design space $\mathcal{X}$. The design space encompasses all possible combinations of design variables, such as material properties, geometric parameters, or component configurations. Each solution $X \in \mathcal{X}$ maps to a point $Y \in \mathcal{Y}$ in the performance space. The performance space $\mathcal{Y}$ can include metrics such as mass, efficiency, cost, or stiffness that are used to evaluate the design.

3.2 Modeling the Design Process

Complex engineering design problems typically follow a structured process that involves decomposition, resource allocation, and sequential problem-solving. The design process can

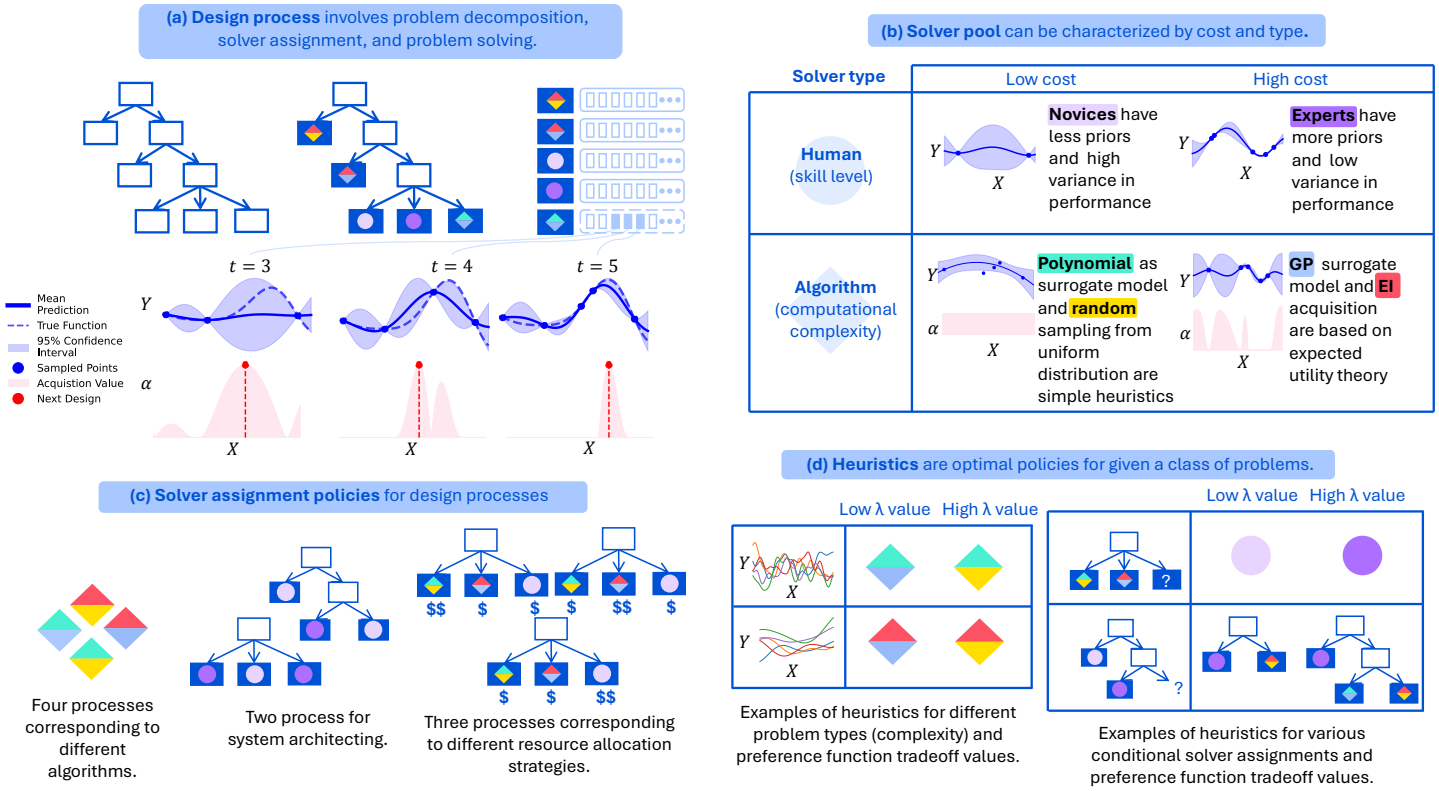

FIGURE 2: EXAMPLES ILLUSTRATING (A) DESIGN PROCESS, (B) SOLVER POOL, (C) POLICIES, AND (D) HEURISTICS.

be conceptualized as follows:

1. **Problem Decomposition:** The overall problem P is recursively divided into subproblems or modules M_k, where each module represents a logically related set of tasks with internal coupling within the module, loose coupling between modules, and handoff rules defining module interactions. For example, in aircraft design, the problem may be decomposed into fuselage design, wing optimization, and propulsion system selection.
2. **Resource Allocation:** Each module M_k is allocated a portion of the total resource budget B. The allocation $\{B_1, B_2, ..., B_N\}$ must satisfy $\sum_{i=1}^{N} B_i \leq B$, reflecting constraints on overall available resources such as time, money, computational power, or human effort.
3. **Solver Assignment:** Each module M_k is assigned a solver s_k from a pool of solvers $\mathcal{S}$, which may include domain experts, novices who are generalists with limited expertise, crowdsourced tournaments in which a group of participants competing to provide solutions or algorithms such as Bayesian optimization. Each solver-module pair incurs a cost, which is deducted from the module's budget. Costs may represent computational expenses for algorithms or contractual compensation for human solvers.
4. **Problem Solving:** Within each module, the problem-solving process follows an iterative sequence (either implicitly in humans or explicitly by algorithms), as shown in Figure 2. At each iteration t, the following steps are performed:

(a) **Design Space Representation:** The relationship between design variables and performance metrics is approximated as: $\hat{f}(X) : \mathcal{X} \rightarrow \mathcal{Y}$, where $X_t \in \mathcal{X}$ represents a design point in the design space, and $Y_t = f(X_t)$ is its corresponding performance in the performance space. Algorithmic solvers employ surrogate models (e.g., Gaussian Processes (GP) [34] or Neural Networks [35]) to represent the design space whereas human solvers do this implicitly. Experts have more prior observations in this space and less uncertainty in performance compared to novices.

(b) **Design Selection:** The next design point is identified by maximizing its acquisition function α over the domain based on the current state of knowledge: $X_{t+1} = \arg\max_{X \in \mathcal{X}} \alpha(X|H_t)$, where $H_t = \{(X_1, Y_1), ..., (X_t, Y_t)\}$ is the state of knowledge after t iterations. Algorithms for Bayesian optimization use acquisition functions such as Expected Improvement [36] and Upper Confidence Bound (UCB) [37]. Humans are known to use expected utility-based models and fast and frugal heuristics in some cases [38].

(c) **Performance Evaluation:** The selected design point X_t is evaluated through experiments, which may involve building physical prototypes or conducting computational simulations: $Y_t = f(X_t)$.

(d) **Knowledge Update:** The state of knowledge is updated with the new observation: $H_{t+1} = H_t \cup (X_t, Y_t)$.

(e) **Termination**: Deciding whether to stop or continue

exploration using criterion δ. The iterative process continues until terminated. The stopping criterion can be based on convergence or when a budget limit is met.

5. **Integration:** Results from individual modules are integrated according to the predefined interfaces and handoff rules to form a complete system solution.

This general framework can be instantiated for specific design contexts. For example, in sequential experimentation, modules might correspond to function learning, information acquisition, and stopping criteria determination. In system architecting, modules might align with functional subsystems like grasping, reaching, or controlling mechanisms.

The output of the design process can be a single optimal solution $X^* = \arg\max_{X \in \mathcal{X}} f(X)$, representing the best-performing design within the constraints or, a Pareto-optimal set of solutions $\mathcal{P} = \{X | X' : f(X') > f(X) \text{ for all objectives}\}$.

3.3 Design Process Heuristics

A class of problems $\mathcal{C}$ represents a set of related design problems that share similar characteristics, objectives or constraints. $\mathcal{C} = \{P_1, P_2, ..., P_n\}$ where each problem P_i has, a design space $\mathcal{X}_i$, a performance space $\mathcal{Y}_i$, a preference function $V_i(Y)$ and a budget constraint B_i. For complex problems, decomposition splits the problem P into modules $\{M_1, M_2, ..., M_N\}$.

Policies h on the design process space can specify:

- **Decomposition:** How to partition the problem $h : P \rightarrow \{M_1, M_2, ..., M_N\}$,
- **Resource allocation:** How to distribute budget across modules $h : B \rightarrow \{B_1, B_2, ..., B_N\}$ where $\sum_{i=1}^{N} B_i \leq B$,
- **Solver assignment:** Mapping from modules to solvers $h : M_i \rightarrow s_j$ where $s_j \in \mathcal{S}$,
- **Problem solving:** Constraints on the sequential information acquisition within modules $h : (H_t, B_t) \rightarrow \{\hat{f}, \alpha, \delta\}$.

A heuristic h^* is an optimal policy on the design process space for a class of problems,

$$h^* = \arg\max_{h \in \mathcal{H}} \mathbb{E}_{P \in \mathcal{C}}[V(h(s)) | s \in \mathcal{G}_P]$$

where $\mathcal{H}$ is the space of possible design process constraints and $\mathcal{G}_P$ represents the situations encountered in problem P.

3.4 Markov Decision Process Formulation

In prior work, we have demonstrated the effectiveness of reinforcement learning approaches for extracting heuristics in specific contexts: solver assignment in systems architecting [39] and resource allocation in hierarchical systems design [40]. However, these approaches were applied to specific problem instances without a unifying theoretical framework to enable generalization across different classes of engineering design problems. We adopt a Markov Decision Process (MDP) framework and formally define our MDP as the tuple (S, A, T, R, γ), where each component is described as follows:

State Space (S): Each state s represents the current status of solver-module assignments within the design process, including historical decisions and their outcomes. Formally, a state is defined as:

$$s_t = \{(M_1, s_1), (M_2, s_2), ..., (M_t, s_t)\}$$

and it captures historical solver assignments made up to iteration t. Each module M_k corresponds to a distinct subproblem within the hierarchical decomposition of the design problem. The remaining available budget or resource constraints at iteration t is implicitly encoded within this representation. Thus, states encapsulate both past solver-module combinations and their impact on resource availability and design performance. Formally, a terminal state s_T is a state where no further actions (solver-module assignments) can be taken or are necessary. This occurs at the final iteration T where either all modules have been assigned solvers, the available budget has been fully consumed, or the design objectives have been satisfied to a predetermined degree.

Action Space (A): The action space consists of all possible solver-module combinations. Actions correspond to selecting solver-module combinations from a predefined set. At each state, an action is formally represented as:

$$a_t = (M_{t+1}, s_{t+1})$$

where: M_{t+1} is the next module or subproblem to be addressed and $s_{t+1} \in \mathcal{S}$ is the solver assigned from a predefined pool of solvers. The solver pool may include domain experts (Exp), novices (Nov), specialists (e.g., Grasping Specialist, Software Specialist) or computational models (e.g., Gaussian Process, Expected Improvement).

Each action thus identifies a specific solver-module pairing decision that advances the design process toward completion.

Transition Probability Function (T): The transition function describes how states evolve probabilistically given current states and actions. It is defined as:

$$T(s'|s, a) = P(s_{t+1} = s' | s_t = s, a_t = a)$$

In our formulation, transitions capture uncertainties inherent in evaluating new designs (e.g., noisy simulations or experimental variability), solver performance variability (e.g., expert versus novice outcomes), and budget consumption dynamics. We assume that transitions are stationary and satisfy the Markov property by explicitly encoding historical information within each state.

Reward Function (R): The reward function quantifies immediate utility or cost associated with each action taken from a given state:

$$R(s, a) : S \times A \rightarrow \mathbb{R}$$

In our framework, rewards explicitly incorporate preference functions (λ) that balance multiple competing objectives such as cost minimization, accuracy maximization, computational efficiency, or solution robustness. This formulation assumes that utility is a linear combination of the objectives. For example, given two conflicting objectives, cost ($f_1(X)$) and quality ($f_2(X)$), the reward associated with evaluating a new design point can be expressed as:

$$R(s_t, a_t) = \begin{cases} 0 & \text{if } s_{t+1} \text{ is not terminal} \\ \lambda f_2 - (1 - \lambda) f_1 & \text{if } s_{t+1} \text{ is terminal} \end{cases}$$

Similarly, actions involving solver assignments incur rewards based on solver-specific costs and expected solution quality improvements.

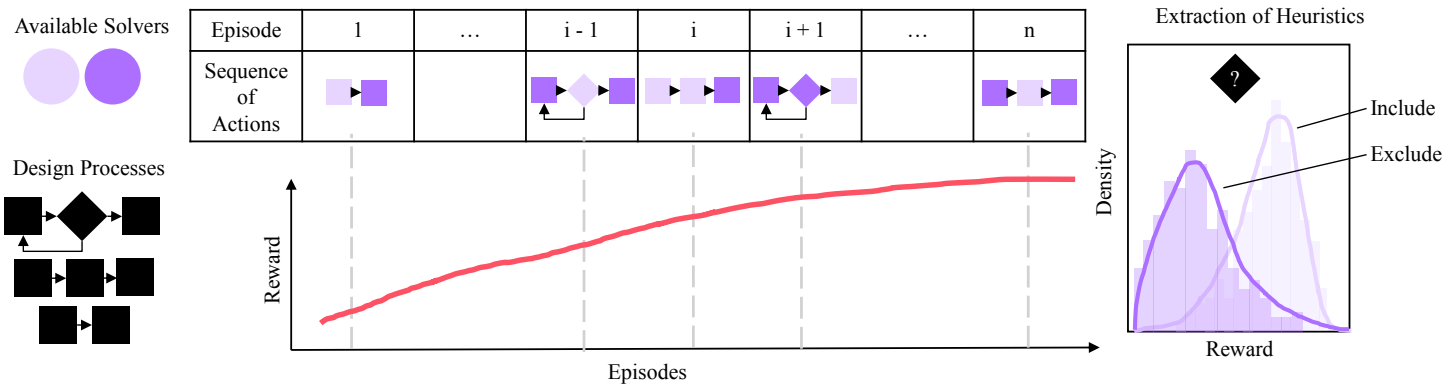

FIGURE 3: ILLUSTRATION OF REINFORCEMENT LEARNING AND EXTRACTION OF HEURISTICS. THE DESIGN PROBLEM THAT CAN BE SOLVED WITH THREE DESIGN PROCESS CONTAINING TWO OR THREE MODULES. THE SOLVER POOL CONTAINS TWO SOLVERS CAN BE ASSIGNED TO DIFFERENT MODULES IN THE DESIGN PROCESS FOR PROBLEM SOLVING.

Discount Factor (γ): The discount factor $\gamma \in [0, 1)$ balances immediate versus future rewards. A value close to zero emphasizes immediate outcomes; values approaching one prioritize long-term cumulative performance.

At time t, the agent can observe the environment state S_t. By taking an action, a, at time, t, the agent interacts with the environment producing a new state, S_{t+1}, receives a reward, R_{t+1}, as a result. The sum of all the future rewards G_t, that agent can receive is,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}.$$

The behavior of an RL agent is determined by a policy π which maps states to actions (deterministic policy), or states to a probability distribution over actions (stochastic policy). The objective of an RL agent is to maximize the expected discounted return $\mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \right]$. We measure this objective by a value function, which measures the expected discounted return after taking the action a in state s:

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi [G_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} | s_t = s, a_t = a \right]. \end{aligned} \tag{1}$$

This MDP formulation provides a structured representation of sequential decision-making in engineering design processes involving decomposition into subproblems, solver assignments, and sequential information acquisition decisions. By explicitly modeling states as combinations of current designs, accumulated knowledge histories, and budget constraints, and defining actions as solver selections or information acquisition decisions, we capture essential dynamics underlying complex design scenarios. Furthermore, integrating explicit preference functions into reward definitions enables systematic consideration of trade-offs among competing objectives inherent in real-world engineering problems.

3.4.1 Assumptions for Tabular MDP Representation.

To ensure tractability and theoretical soundness of our MDP formulation, particularly when employing tabular reinforcement learning methods, we make several assumptions:

1. **Discrete and Finite State-Action Space:** We discretize continuous design variables and knowledge states into finite sets to enable tabular representation.
2. **Markov Property:** The next state depends solely on the current state and action taken, independent of prior history. We achieve this by explicitly encoding historical information within our state representation.
3. **Stationary Transition Dynamics:** Transition probabilities remain constant over time. This assumption holds under controlled experimental conditions and stable environmental contexts.
4. **Bounded Rewards:** Immediate rewards are bounded to ensure convergence of reinforcement learning algorithms.

These assumptions enable us to leverage standard tabular reinforcement learning algorithms (e.g., Q-learning) to systematically explore and evaluate heuristic policies [8].

3.4.2 Reduction to Multi-Armed Bandit Problems in Special Cases.

In certain simplified scenarios within our framework, the general MDP formulation reduces naturally to a Multi-Armed Bandit (MAB) problem. This simplification occurs when:

1. The decision horizon is effectively one step (single-action selection without sequential dependencies).
2. State transitions do not depend on actions (i.e., stationary states), or all actions lead to equivalent future states.

Under these conditions, the problem simplifies from sequential decision-making to selecting among competing alternatives based on uncertain rewards. Recognizing these special cases allows us to apply simpler algorithms (e.g., Upper Confidence Bound or Thompson Sampling), significantly reducing computational complexity [8].

3.5 Extraction of Heuristics

To systematically derive actionable heuristics from learned RL policies, we propose a structured heuristic extraction approach that analyzes reward histories to determine optimal solver assignments for each subproblem or module in the design process. An example of this is shown in Figure 3. Our methodology focuses on identifying both inclusionary and exclusionary recommendations for solver assignments, with the constraint that at least one solver must be included for each module. We first identify high-performing state-action pairs based on learned value functions from the RL policy. Specifically, we define a confidence threshold τ, representing a minimum acceptable performance level relative to other choices.

3.5.1 Gaussian Mixture Model Clustering. After collecting sufficient reward history data from the reinforcement learning process, we extract heuristics using a comparison. For each module M_i, we consider all possible solver assignment combinations represented as a set of constraints $G_i = \{g_1, g_2, \ldots, g_k\}$. Each constraint $g_j \in G_i$ consists of a pair (S_j^+, S_j^-) where S_j^+ denotes the set of solvers to include (at least one solver must be included) and S_j^- denotes the set of solvers to exclude (can be empty). We formalize this constraint as $|S_j^+| \geq 1$ and $S_j^+ \cap S_j^- = \emptyset$. For each constraint g_j, we compute the probability that the reward obtained using this constraint exceeds the performance of a solution obtained by the opposite constraint. This probability is formally defined as $P(g_j) = \Pr(R(g_j) > R(\bar{g}_j))$ where $\bar{g}_j$ represents the opposite constraint. A constraint g_j is considered a valid heuristic if its probability exceeds a predefined confidence threshold τ, $P(g_j) \geq \tau$.

By varying the threshold parameter τ, we control the specificity versus generality of the extracted heuristics: higher thresholds yield fewer but more reliable heuristics, whereas lower thresholds result in broader applicability but potentially reduced reliability. To identify distinct decision-making contexts that might warrant different solver assignment strategies, we apply Gaussian Mixture Models (GMMs) to cluster high-performing state-action pairs. Formally, a GMM with K components models the probability density function of data points x as $p(x|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$, where π_k are mixing coefficients satisfying $\sum_{k=1}^{K} \pi_k = 1$, $\mathcal{N}(x|\mu_k, \Sigma_k)$ denotes the Gaussian distribution with mean vector μ_k and covariance matrix Σ_k for component k. To select the optimal number of clusters (K), we employ the Bayesian Information Criterion (BIC), defined as $\text{BIC} = -2\ln(\hat{L}) + d\ln(n)$, where $\hat{L}$ is the maximized likelihood function for the fitted GMM, d is the number of parameters estimated by the model, and n is the number of data points.

3.5.2 Identification of Inclusionary and Exclusionary Heuristics. From the analyzed reward histories, we extract inclusionary and exclusionary heuristic pairs for each module. These pairs specify which solvers to include and which to exclude for optimal performance:

Formally, for each module M_i, we derive a heuristic pair (h_i^+, h_i^-) where:

- **Inclusionary heuristic set** h_i^+: Specifies the set of solvers S^+ that should be included for module M_i. This set must be non-empty: $|S^+| \geq 1$.
- **Exclusionary heuristic set** h_i^-: Specifies the set of solvers S^- that should be excluded from consideration for module M_i. This set may be empty and must be disjoint from the inclusionary set: $S^+ \cap S^- = \emptyset$.

The derived heuristics provide clear guidance for solver assignment in future design processes, with rigorous probabilistic guarantees of their effectiveness based on the accumulated reward history data from the reinforcement learning process. Through this approach, leveraging MDP representations coupled with heuristic extraction via probabilistic clustering we aim to systematically identify generalizable heuristics.

4. EXPERIMENTAL SETUP

In this section, we describe the two case studies and instantiate our formal framework to extract heuristics for each domain. We begin with solver-aware system architecting for a robotic arm design, followed by sequential information acquisition in parametric design optimization task and highlight the differences in Table 2.

4.1 Solver-Aware System Architecting (SASA)

SASA problem comprises four main elements: a reference problem, alternative task structures (or modularizations), solver assignment, and the simulated solving process.

4.1.1 Design Problem. Consider the design of a robotic arm intended for deployment on the International Space Station (ISS). Its primary functions include reliably attaching to the ISS external handrails, securely holding on, and subsequently detaching. This design problem, adapted from NASA's Astrobee gripper challenge [41], emphasizes minimizing mass under stringent constraints on physical dimensions, strength (forces and moments), and energy efficiency. The design task decomposes into five primary functions illustrative of typical robotic design tasks: (i) *grasping*, securely attaching to handrails; (ii) *reaching*, moving the gripper to the required location; (iii) *orienting*, positioning the grasping mechanism correctly at the attachment point; (iv) *control (software)*, managing intelligent grasp strategies and movement paths; and (v) *electronics*, facilitating consistent power and signal distribution. Each simulation begins with decomposition, solver assignment, and culminates in evaluating results according to the total system mass and overall cost of the solving process.

4.1.2 Design Process. The baseline robotic arm reference problem is modularized into task structures based on functional and disciplinary decompositions [42]. This yields four plausible architectures [41]: an undecomposed reference architecture ($D1$), a two-module configuration separating fine positioning from attachment ($D2$), an alternative two-module configuration delineating coarse positioning from positioning attachment ($D3$), and a three-module disciplinary decomposition separating reaching, grasping, and positioning ($D4$). Each modularized architecture includes explicit interface rules governing integration at the functional level. We describe the module below:

- *SRA (Smart Robotic Arm)*: The entire robotic arm design considered as a single, undecomposed problem. This serves as the baseline architecture against which other decomposed architectures are compared.

TABLE 2: COMPARISON OF THE HEURISTICS EXTRACTION FORMULATION ACROSS TWO DISTINCT DESIGN PROBLEMS

	Solver-Aware System Architecting (SASA)	Sequential Information Acquisition (SIADM)
Task Modules	Functional decomposition (e.g., SRA, SFA, SAM, SCA, SPAM, PRM)	Process decomposition (Function Learning, Information Acquisition, Stopping Criteria)
Solvers	Professionals, Novices, Grasping Specialists, Software Specialists	Function Learning: Linear, Quadratic, Cubic, GP Information Acquisition: RS, UCB, EI Stopping: FRB, REI
Objectives	Mass minimization and cost minimization	Solution quality maximization and cost minimization
Extracted Heuristics	Solver recommendations for each module (e.g., "Use Novices for PRM")	Solver recommendations for each subproblem based on complexity level (e.g., "Use GP for function learning when problem complexity is high")

- *SFA (Smart Fine-positioning Arm)*: One of the two subproblems in the $D2$ architecture, focusing on the fine-positioning aspects of the robotic arm.
- *SAM (Smart Attachment Mechanism)*: The second subproblem in the $D2$ architecture, dealing with the mechanism for attaching the robotic arm to the International Space Station's (ISS) handrail1. It also appears in architecture D4.
- *SCA (Smart Coarse-positioning Arm)*: One of the two subproblems in the $D3$ architecture, responsible for the coarse-positioning movements of the robotic arm. It also appears in architecture D4.
- *SPAM (Smart Positioning Attachment Mechanism)*: The second subproblem in the $D3$ architecture, focusing on the positioning aspects of the attachment mechanism. PRM (Pass-through Reaching Mechanism): A subproblem in the $D4$ architecture, related to the reaching functionality of the robotic arm.

4.1.3 Design Process Heuristics.

Four solver types are represented, analogous to typical engineering teams spanning multiple expertise levels:

- *Professionals* represent highly capable multidisciplinary robotic design teams (e.g., internal NASA Astrobee team). They exhibit high competence across mechanical, electronics, and software/control domains relevant to the robotic arm. Professionals are generally skilled across all tasks.
- *Novices* are non-traditional solvers (e.g., open innovation participants) with significantly lower average performance and greater variability (uncertainty). Novices have limited expertise across all domains.
- *Grasping Specialists* have deep expertise specifically in mechanical grasping mechanisms. Outside their area of specialty, performance resembles that of Novices.
- *Software Specialists* possess specialized software control expertise, particularly in robotic control strategies. They perform similarly to Novices outside software/control domains.

Solver capabilities are represented as probabilistic distributions calibrated from real robotic arm solution data documented in the Astrobee design challenge [41, 43].

To simulate solving, alternative task structures $\{D1, D2, D3, D4\}$, solver assignments, and contract mechanisms are combined with solver type capabilities to produce robotic arm designs. Outcomes (mass and cost) are computed as follows.

Performance (M) is measured based on total robotic arm system mass. Let f denote each function (e.g., grasping), Q_f represent normalized performance quality (mass), and w_f represent the mass contribution weight of function f. Performance score for architecture a is computed as: $M_a = \sum_{f \in a} w_f \cdot Q_f$, with function-specific distributions calibrated from robotic arm design datasets [41].

Cost (c) for solver assignment reflects realistic contractual mechanisms. Professionals follow conventional employment contracts, incurring fixed wage costs proportional to effort. Specialists (both Grasping and Software) are engaged through competitive contracts involving three competing specialists, where the winner receives full compensation, and losing participants receive partial bidding costs. Novices are contracted through a prize-based open innovation contest with a fixed prize purse, incurring only the prize payout to the best solution.

4.2 Markov Decision Process Formulation for SASA Problem

In this subsection, we describe the Markov Decision Process (state, action, and reward) used for decision-making for solver assignment in the Robotic Arm model. For each alternative task structure in the Robotics example (D1, D2, etc), we can assign combinations of available solvers (Professional, Novice, Grapsing Specialist, Software Specialist). The state is initialized as an empty list, which keeps track of all the solver assignments made. The actions contain the possible solver assignments that can be made (e.g.: SAM-Professional, SPAM-Grasping Specialist, etc). We formulate the SASA (Sequential Architecture-Solver Assignment) problem for designing the robotic arm as a Markov Decision Process (MDP) and define the following:

State Space. The state space $s_t \in \mathcal{S}$ at decision step t encapsulates the solver-module assignments made thus far: $s_t = \{(m, \sigma_m)\}_{m \in M_a}$ where M_a represents the set of modules for which solvers have already been assigned and $\sigma_m \in$ {Professional, Novice, Grasping Specialist, Software Specialist} denotes the solver type assigned to module m. Initially, $s_0 = \emptyset$.

Action Space. Actions $a_t \in \mathcal{A}(s_t)$ correspond to selecting the next solver-module combination: $a_t = (m', \sigma_{m'})$, $m' \in M_u$, $\sigma_{m'} \in \Sigma$ where M_u represents unassigned modules and Σ is the set of available solver types.

Reward Function. Upon completion of assignments in a specific architecture, the design is evaluated. The final reward is:

$$R(s_t, a_t) = \begin{cases} 0 & \text{if assignments incomplete} \\ -[\lambda M + (1-\lambda)C] & \text{at terminal state} \end{cases}$$

where M is the scaled total system mass, C is the scaled total cost and λ is the weight for mass minimization and cost minimization tradeoff.

Transitions. The state transitions deterministically as: $s_{t+1} = s_t \cup \{a_t\}$ The MDP terminates once all modules are assigned solvers.

4.3 Sequential Information Acquisition and Decision Making (SIADM)

The experiment considers the process of engineering design framed as a sequential information acquisition and decision-making problem under resource constraints. The designer sequentially selects and evaluates design alternatives with the goal of maximizing performance within a finite evaluation budget [44, 45]. The simulated experimental setup consists of three primary components: the reference design problem, alternative heuristic decision strategies for solving three key sub-problems, and a simulation procedure.

TABLE 3: OBJECTIVE FUNCTION COMPLEXITY PARAMETERS.

Complexity Level	Length-scale (ℓ)	Variance (σ^2)
Low	5	4
Medium	3	3
High	1.5	5

4.3.1 Design Problem.

We examine a parametric engineering design task described by a one-dimensional continuous design variable $x \in \mathcal{X} = [-10, 10]$, evaluated by an unknown but deterministic performance function $f(x)$. The designer aims to identify the design parameter x that maximizes the performance $y = f(x)$. The exact functional relation between design parameter and performance is not explicitly known; instead, the designer must learn this relationship iteratively through costly evaluations (simulations or experiments) at sequentially chosen design points. The objective functions are simulated as Gaussian process (GP) realizations with radial basis function (RBF) kernels. Variations in function complexity are simulated by adjusting GP parameters (length-scale ℓ and variance σ^2), allowing evaluation of heuristic effectiveness under low, medium, and high complexities (see Table 3).

4.3.2 Design Process.

Designers face three sequential decision sub-problems: (1) selecting a function-learning model to approximate the unknown mapping between design parameter and performance, (2) deciding which next design point to evaluate (information acquisition), and (3) determining when to terminate experimentation (stopping criteria). The simulation experiment assumes the following simplifying conditions: a fixed resource budget per run, consumed incrementally with each design evaluation; deterministic performance evaluations without measurement noise; a single information source with constant cost per evaluation; and each simulated run involves sequential heuristic selection for each of the three sub-problems (function learning, next design point acquisition, stopping decision), applied iteratively until termination.

4.3.3 Design Process Heuristics.

Each sub-problem is solved using alternative candidate heuristic strategies:

- *Function-Learning Models:* Four competing heuristics are considered for learning the performance function mapping:
  1. *Linear Bayesian regression* (Linear basis) [35, Chapter 3].
  2. *Quadratic Bayesian regression* (Quadratic polynomial basis).
  3. *Cubic Bayesian regression* (Cubic polynomial basis).
  4. *Gaussian Process regression (GP)* (nonparametric approach, RBF kernel) [34, Chapter 2].
- *Next-Design-Point Acquisition Strategies:* Three heuristics represent ways to select the next point for experimentation:
  1. *Random Selection (RS)*: Selecting random untested design points.
  2. *Upper Confidence Bound (UCB)*: Balancing exploration (uncertainty) and exploitation (performance) [37].
  3. *Expected Improvement (EI)*: Choosing points based on predicted incremental improvement relative to current best solution [36].
- *Stopping Criteria Heuristics:* Two heuristics determine whether to continue or terminate the sequential information-acquisition process:
  1. *Fixed Remaining Budget (FRB)*: Stops evaluation after a specified fixed budget remains.
  2. *Relative Expected Improvement (REI)*: Stops if expected improvement drops below a pre-defined fraction of the initial expected improvement.

This experimental framework provides a structured method to compare the suitability of heuristics across different conditions, informing heuristic selection and engineering design decision-making under diverse resource constraints. To systematically evaluate heuristic effectiveness, the simulation fixes the number of initial observations (4) and varies the relative weighting between solution qulity q and computational cost C, and the complexity of the underlying problem (as defined in Table 3).

4.4 Markov Decision Process Formulation for SIADM Problem

The SIADM (Sequential Information Acquisition and Decision-Making) problem is formulated as an MDP and define the following:

State Space. The state $s_t \in \mathcal{S}$ tracks prior heuristic-subproblem assignments: $s_t = \{(h, p)\}_{h \in H_a, p \in P}$ where H_a represents subproblems assigned heuristics and $p \in P$ indicates the selected heuristic for each subproblem. Initially, $s_0 = \emptyset$.

Action Space. Actions represent selecting a heuristic-subproblem combination: $a_t = (h', p'), \quad h' \in H_u, \quad p' \in P_{h'}$ where H_u denotes unassigned subproblems and $P_{h'}$ is the set of heuristics for subproblem h'. Examples of actions include Function Learning-GP , Stopping-FRB, etc.

TABLE 4: HEURISTICS IDENTIFIED FOR VARIOUS MODULES AND λ VALUES GIVEN $\tau = 0.8$. THE INCLUSIONARY HEURISTICS ARE SHOWN IN NORMAL TEXT AND EXCLUSIONARY HEURISTICS IN STRIKETHROUGH TEXT. WE INCLUDE AT LEAST ONE SOLVER AND OPTIONALLY EXCLUDE SOME SOLVERS.

Module	Confidence threshold = 80%				
	$\lambda = 0.75$	$\lambda = 0.81$	$\lambda = 0.87$	$\lambda = 0.93$	$\lambda = 1$
PRM	Nov, ~~Exp~~, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~
SAM	Exp, Nov, Grasp,~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~
SCA	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Exp, ~~Nov~~, ~~Grasp~~, ~~Soft~~
SFA	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Nov, Exp, ~~Grasp~~, ~~Soft~~	Exp, ~~Nov~~, ~~Grasp~~, ~~Soft~~
SPAM	Exp, Nov, Grasp,~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~
SRA	Exp, Nov, ~~Grasp~~,~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, Nov, Grasp, ~~Soft~~	Exp, ~~Nov~~, ~~Grasp~~, ~~Soft~~

Reward Function. Upon solving all subproblems, the design is evaluated:

$$R(s_t, a_t) = \begin{cases} 0 & \text{if subproblems unsolved} \\ \lambda q + (1-\lambda)C & \text{at terminal state} \end{cases}$$

where q measures the scaled design quality, C is the scaled computational cost, and λ is the weight balancing performance maximization and cost minimization.

Transitions. State transitions are deterministic: $s_{t+1} = s_t \cup \{a_t\}$ The process terminates when all subproblems are assigned heuristics.

5. RESULTS AND ANALYSIS

In this section, we present the results of applying the reinforcement learning-based heuristic extraction methodology to the two case studies. We first examine the heuristics learned for solver assignment in the robotic arm design problem (SASA), followed by the heuristics identified for sequential information acquisition in parametric design optimization (SIADM). For both cases, we analyze how problem characteristics, designer preferences, and solver capabilities influence the optimal heuristic selection.

5.1 Learned Heuristics for SASA

We calculate the Bayesian Information Criterion (BIC) scores to determine the optimal number of Gaussian Mixture Model (GMM) components for each solver reward distribution and pick a GMM with 3 components. This approach allows us to accurately characterize solver performance distributions and subsequently generate solver-assignment heuristics. Using a confidence threshold ($\tau = 0.8$), we identify inclusionary and exclusionary heuristics across different modules and varying weights of performance in the reward function (λ). For all simulations, we used Q-learning with UCB policy with $t = 5000, \gamma = 0.99$ and ran the model for 1000 runs (50 samples of 20 runs). Table 4 summarizes these heuristics, indicating inclusionary heuristics in normal text and exclusionary heuristics in strikethrough.

We observe several trends from the identified heuristics. For the PRM module, Novices consistently emerge as an inclusionary heuristic across all values of λ, while Grasping and Software Specialists are consistently exclusionary. Professionals transition from exclusionary at lower values of λ (0.75) to inclusionary at higher values ($\lambda \geq 0.81$). This indicates that as performance becomes more heavily weighted in the reward function, employing Professionals alongside Novices becomes beneficial. The SAM and SPAM modules exhibit similar heuristic patterns, consistently favoring Novices, Professionals, and Grasping Specialists while excluding Software Specialists across all considered λ values. This suggests that software specialization does not significantly enhance performance or cost-effectiveness for these modules. Modules SCA and SFA show a clear shift with increasing emphasis on performance ($\lambda = 1$): while Novices and Professionals are both inclusionary heuristics for lower λ values, at maximum $\lambda = 1$, only Professionals remain inclusionary, indicating that as performance becomes paramount, Professional solvers become preferable over Novices. For module SRA, we notice a transition: at lower λ values (0.75), only Professionals and Novices are inclusionary; as λ increases (0.81–0.93), Grasping Specialists also become inclusionary; however, at the highest performance emphasis ($\lambda = 1$), only Professionals remain inclusionary, excluding all other solver types. These results indicate that solver-assignment heuristics are sensitive to both module characteristics and the relative importance placed on system mass versus cost (λ). Specifically, as the weight of performance (λ) increases—prioritizing mass minimization over cost reduction—the optimal solver assignments shift towards higher-performing solvers such as Professionals and Specialists. Conversely, when cost considerations dominate (lower λ), lower-cost solvers like Novices frequently become inclusionary.

The threshold value ($\tau = 0.8$) chosen for heuristic identification ensures a high confidence level in the solver assignments presented. While a higher threshold reduces the total number of identified heuristics compared to this values, it provides greater confidence in their applicability across scenarios. Overall, our results demonstrate that solver assignment heuristics depend significantly on module-specific characteristics and the designer's preference for balancing mass performance with associated costs. These insights underscore the importance of carefully considering solver capabilities and contractual mechanisms when architecting robotic systems intended for space applications such as NASA's Astrobee robotic arm design challenge.

5.2 Learned Heuristics for SIADM

We conducted reinforcement learning simulations to identify optimal heuristics across three engineering design subproblems—function learning, information acquisition, and stopping criteria—under varying complexity levels and performance-cost preferences. We pick a GMM with 5 components after evaluating BIC scores. For all simulations, we used Q-learning

TABLE 5: HEURISTICS IDENTIFIED FOR VARIOUS PROBLEM TYPES AND λ VALUES GIVEN $\tau = 0.8$. THE INCLUSIONARY HEURISTICS ARE SHOWN IN NORMAL TEXT AND EXCLUSIONARY HEURISTICS IN STRIKETHROUGH TEXT. WE INCLUDE AT LEAST ONE SOLVER AND OPTIONALLY EXCLUDE SOME SOLVERS.

Problem Type	Complexity	Confidence threshold = 80%				
		$\lambda = 0.4$	$\lambda = 0.55$	$\lambda = 0.7$	$\lambda = 0.85$	$\lambda = 1$
Function Learning	Low	Linear, ~~Quadratic~~ ~~Cubic~~, ~~GP~~	Linear, Quadratic Cubic, ~~GP~~	~~Linear~~, ~~Quadratic~~ ~~Cubic~~, GP	~~Linear~~, ~~Quadratic~~ ~~Cubic~~, GP	~~Linear~~, ~~Quadratic~~ ~~Cubic~~, GP
	Medium	Linear, ~~Quadratic~~ ~~Cubic~~, ~~GP~~	Linear, ~~Quadratic~~ ~~Cubic~~, ~~GP~~	Linear, ~~Quadratic~~ ~~Cubic~~, GP	~~Linear~~, ~~Quadratic~~ ~~Cubic~~, GP	~~Linear~~, ~~Quadratic~~ ~~Cubic~~, GP
	High	Linear, ~~Quadratic~~ ~~Cubic~~, ~~GP~~	Linear, ~~Quadratic~~ ~~Cubic~~, ~~GP~~	Linear, ~~Quadratic~~ ~~Cubic~~, GP	~~Linear~~, ~~Quadratic~~ ~~Cubic~~, GP	~~Linear~~, ~~Quadratic~~ ~~Cubic~~, GP
Information Acquisition	Low	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, ~~EI~~	~~RS~~, UCB, EI	~~RS~~, UCB, EI
	Medium	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, ~~EI~~	~~RS~~, UCB, EI	~~RS~~, UCB, EI
	High	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, ~~EI~~	RS, ~~UCB~~, EI
Stopping Criteria	Low	FRB, ~~REI~~	FRB, ~~REI~~	FRB, ~~REI~~	FRB, ~~REI~~	~~FRB~~, REI
	Medium	FRB, ~~REI~~	FRB, ~~REI~~	~~FRB~~, REI	~~FRB~~, REI	~~FRB~~, REI
	High	FRB, ~~REI~~	FRB, ~~REI~~	~~FRB~~, REI	~~FRB~~, REI	~~FRB~~, REI

with UCB policy for $t = 25000, \gamma = 0.99$ and ran the model for 1000 runs (50 samples of 20 runs). Table 5 summarizes the identified heuristics for each sub-problem, complexity level, and performance-cost ratios (λ), using a confidence threshold ($\tau = 0.8$). Inclusionary heuristics are indicated in normal text, while exclusionary heuristics are shown in strikethrough.

For the function learning sub-problem, we observe clear shifts in heuristic preference based on the emphasis placed on accuracy versus computational cost. At lower λ values (0.4 and 0.55), the Linear model consistently emerges as inclusionary across all complexity levels (low, medium, high), indicating that simpler regression models are preferable when computational cost dominates the reward function. Conversely, Quadratic, Cubic, and Gaussian Process (GP) models are exclusionary at these lower λ values. As the emphasis on accuracy increases ($\lambda \geq 0.7$), the Gaussian Process (GP) model becomes inclusionary across all complexity levels, while simpler polynomial models (Linear, Quadratic, Cubic) become exclusionary. This shift clearly demonstrates that GP-based similarity models outperform simpler rule-based models when accuracy is prioritized. Interestingly, at medium and high complexities, the transition from Linear to GP occurs at higher λ values compared to low complexity problems. This suggests that for more complex problems, simpler models remain competitive until accuracy is heavily weighted.

For the second sub-problem—information acquisition, heuristic effectiveness varies significantly with both complexity and performance-cost balance. For low and medium complexities, Random Selection (RS) is inclusionary when cost considerations dominate ($\lambda = 0.4, 0.55, 0.7$), while Upper Confidence Bound (UCB) and Expected Improvement (EI) heuristics become inclusionary at higher performance emphasis ($\lambda = 0.85, 1$). Thus, when computational cost outweighs performance gains, simple random selection is sufficient; however, as performance becomes more critical relative to cost, strategies informed by uncertainty quantification (UCB and EI) become advantageous. For high complexity problems in information acquisition, RS remains consistently inclusionary across nearly all λ values except at maximum performance emphasis ($\lambda = 1$), where EI also becomes inclusionary alongside RS. This result indicates that for highly complex problems with limited information availability, random selection remains robust even when performance is highly valued.

For the third sub-problem, determining when to stop acquiring information, the identified heuristics exhibit clear trends based on problem complexity and designer preferences: At low complexity levels and lower λ values (0.4–0.85), Fixed Remaining Budget (FRB) consistently emerges as an inclusionary heuristic while Relative Expected Improvement (REI) is exclusionary. However, at maximum emphasis on performance ($\lambda = 1$), REI becomes preferable due to its ability to adaptively stop based on diminishing returns in expected improvement. At medium and high complexities, FRB remains inclusionary only at lower λ values (0.4–0.55). As performance becomes increasingly important ($\lambda \geq 0.7$), REI emerges as inclusionary while FRB becomes exclusionary. This suggests that adaptive stopping criteria based on expected improvement are more effective when prioritizing design quality over cost savings in complex scenarios.

In summary, our results highlight the critical role of problem complexity and designer preferences in determining optimal heuristics for engineering design sub-problems. Simpler heuristic strategies such as linear regression for function learning or random selection for information acquisition are effective under conditions prioritizing computational efficiency or limited available information. Conversely, expected utility based heuristics such as Gaussian Processes for function learning or Expected Improvement-based strategies for information acquisition and stopping decisions become preferable when accuracy or design performance is emphasized. These findings provide valuable normative guidance for designers seeking to select appropriate heuristics tailored to their specific problem contexts and resource constraints.

6. DISCUSSION

In this work we present a reinforcement learning-based approach to extract heuristics for engineering design problems involving decomposition and assignment decisions. By formal-

izing the design process as a Markov Decision Process, we are able to systematically derive heuristics that can guide designers in making informed decisions. The framework's effectiveness was demonstrated through two case studies. Despite their distinct application contexts – robotic system architecting and sequential information acquisition – the SASA and SIADM frameworks share commonalities.

In this section, we discuss their shared traits and how the generality of our proposed heuristic extraction method enables its applicability to a broader range of engineering design problems. Both SASA and SIADM approaches are fundamentally framed as sequential decision-making problems under conditions of uncertainty, resource constraints, and clearly defined preferences. Both frameworks feature clear modularization of complex problems. In SASA, the robotic system design task is decomposed into functional modules (e.g., grasping, reaching), while in SIADM, design tasks are decomposed into sequential subtasks (function learning, information acquisition, and stopping decision). This modular decomposition facilitates systematic assignment of solver types or heuristic strategies tailored to specific subtasks or modules, optimizing overall design outcomes. Engineering design tasks typically have inherent dependencies and information flows making them effectively represented as networks of interconnected activities or as a Design Structure Matrix (DSM) and the unified methodology presented in this paper enables heuristic extraction in a wide range of design processes with such representations.

Next, we compare the two case studies and the heuristics learned with experiments performed with similar problem representations in previous studies.

6.1 Generalizable Heuristics for SASA

Across diverse complex system design contexts, including the robotics problem described and abstract system engineering models like Golf [40], several solver assignment heuristics provide actionable guidance. First, decompose the system such that subproblems align closely with specialized external expertise, enhancing performance by leveraging deep domain knowledge; for example, assign grasping tasks to Grasping Specialists in robotics or use specialist solvers for tasks matching their expertise similar to Specialist assignment for the Tee module in Golf. Second, leverage crowdsourcing tournaments for subproblems with low entry barriers and high solution variability, enabling the selection of superior outcomes from numerous attempts at minimal cost, which works especially well when cost reduction is a major factor. Third, prioritize high-performing solvers (e.g., Professionals, Specialists) as the weight on performance maximization increases (high λ) to achieve optimal outcomes, even if at a higher cost. Conversely, prioritize low-cost solvers (e.g., Novices, Amateurs) when the weight on cost minimization is high (low λ) to minimize expenses, even if at the expense of performance. Finally, avoid assigning specialized solvers to subproblems that do not align with their core competencies, as their performance in these areas may not justify the added cost or complexity.

6.2 Generalizable Heuristics for SIADM

In comparing the SIADM experiments in racecar design using a vehicle dynamics simulator [39] and our information acquisition experiment, several key similarities and differences emerge. Both experiments frame engineering design as a sequential information acquisition and decision-making problem under finite resource constraints, employing reinforcement learning to identify optimal heuristics. In The Open Racing Car Simulator (TORCS) experiment, the hierarchical approach decomposes the problem into system-level budget allocation and subsystem-level heuristic selection, consistently prioritizing higher-dimensional subsystems (gearbox) and favoring expected utility based heuristics (UCB), while explicitly avoiding Probability of Improvement (PI) for certain subsystems [39]. Conversely, our SIADM experiment evaluates heuristic effectiveness across three distinct subproblems: function learning, information acquisition, and stopping criteria, highlighting that simpler heuristics (e.g., Random Selection) are effective when computational efficiency dominates, whereas expected utility based heuristics (e.g., Expected Improvement) are preferred when performance is emphasized. While both experiments demonstrate the value of informed heuristic selection, the earlier study emphasizes hierarchical decomposition to manage complexity efficiently and learns heuristics for information acquisition, whereas our SIADM experiment systematically varies problem complexity and performance-cost trade-offs to derive generalizable heuristic recommendations across multiple subproblems beyond information acquisition.

6.3 Limitations and Future Work

While our reinforcement learning approach offers a promising path toward systematically extracting generalizable design heuristics, it is essential to acknowledge its current limitations and outline avenues for future research. These limitations primarily stem from the inherent complexities of representing engineering design problems, the computational demands of reinforcement learning, and challenges in validating and generalizing extracted heuristics. The Markov property assumption might not fully capture the path-dependent nature of some design decisions. Future work should explore more expressive representations, such as Partially Observable Markov Decision Processes (POMDPs) or graph-based models, to better capture uncertainty, long-term dependencies, and the complex relationships between design variables and performance metrics. The computational cost of reinforcement learning, particularly with tabular methods, poses a significant limitation for large-scale design problems. The "curse of dimensionality" restricts the applicability of our approach to problems with a relatively small number of design variables, modules, and solver types. Future research should investigate more scalable reinforcement learning algorithms, such as function approximation methods (e.g., deep reinforcement learning), to handle high-dimensional state and action spaces. External validation with human designers in real-world engineering contexts is currently limited. The extracted heuristics, while computationally effective, might not always align with human intuition or be easily integrated into existing design workflows. Future work should focus on conducting user studies to assess the understandability, applicability, and perceived value of the extracted

heuristics.

Several promising avenues for future research exist. Integrating Large Language Models (LLMs) as potential solvers within the framework could unlock new possibilities for concept generation, requirements analysis, and design evaluation. Research could explore optimal sequencing of LLM-based solvers with other computational and human experts. Future work can quantify the entropy of design process trajectories and identify high-likelihood patterns analogous to language modeling similar to Shannon's demonstration of n-gram models which generate sequences preserving statistical structure while enabling entropy-based analysis of information content [46]. Recent advances in concept generation using LLMs with professional personas [47] further demonstrate how semantic diversity can be systematically induced, and when combined with the proposed RL methodology for heuristic extraction, it enables cross-domain knowledge transfer by aligning n-gram structures between various design problems. Finally, exploring hybrid approaches that combine computational heuristic extraction with human expertise could lead to more effective and user-friendly design tools. This could involve developing interactive design systems that provide designers with algorithmically derived heuristics while allowing them to leverage their own intuition and experience.

7. CONCLUSION

In this paper, we introduced a framework for extracting generalizable heuristics in engineering design through formal problem representation and reinforcement learning. The approach involves defining a Markov Decision Process to model the design process, where states represent solver-module assignments, actions correspond to selecting solver-module combinations, and rewards reflect design performance and cost. Q-learning is then used to learn an optimal policy for solver assignment, and Gaussian Mixture Models are used to identify heuristics. Our approach characterizes design problems through attributes of the problem space, solver capabilities, and preference functions, enabling the identification of context-dependent heuristics.

Through two case studies, solver-aware system architecting (SASA) for robotic arm design and sequential information acquisition in parametric design optimization (SIADM), we demonstrated how our framework effectively extracts inclusionary and exclusionary heuristics. For SASA, we found that solver assignment heuristics depend significantly on module characteristics and designer preferences for balancing performance with cost. As performance emphasis increases, optimal solver assignments shift toward higher-performing solvers like Professionals and Specialists. For SIADM, we observed that simpler heuristics (e.g., Linear models, Random Selection) are effective when computational efficiency dominates, while expected utility-based heuristics (e.g., Gaussian Processes, Expected Improvement) are preferred when performance is emphasized. By formalizing problem representations and systematically extracting heuristics, our approach represents a significant step toward understanding the context-dependent nature of effective design strategies and providing designers with actionable guidance tailored to specific problem characteristics.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support from the National Science Foundation through NSF CMMI Grants 2129539 (Purdue University) and 2129574 (the George Washington University). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Simon, Herbert A. "The structure of ill structured problems." *Artificial intelligence* Vol. 4 No. 3-4 (1973): pp. 181–201.

[2] Fu, Katherine K., Yang, Maria C. and Wood, Kristin L. "Design Principles: Literature Review, Analysis, and Future Directions." *Journal of Mechanical Design* Vol. 138 No. 10 (2016).

[3] Yilmaz, Seda, Daly, Shanna R., Seifert, Colleen M. and Gonzalez, Richard. "How do designers generate new ideas? Design heuristics across two disciplines." *Design Science* Vol. 1 (2015).

[4] Fillingim, Kenton B., Nwaeri, Richard O., Borja, Felipe, Fu, Katherine and Paredis, Christiaan J. J. "Design Heuristics: Extraction and Classification Methods with Jet Propulsion Laboratory's Architecture Team." *Journal of Mechanical Design* (2019): p. 1.

[5] Yilmaz, Seda and Seifert, Colleen M. "Creativity through design heuristics: A case study of expert product design." *Design Studies* Vol. 32 No. 4 (2011): pp. 384–415.

[6] Deshmukh, Anand P, Thurston, Deborah L and Allison, James T. "Heuristics for Formulating Design Optimization Models: Their Uses and Pitfalls." *5th Internations Engineering Systems Symposium, CESUN 2016*. 2016. Washington, D.C., USA.

[7] Pearl, J. *Heuristics: Intelligent search strategies for computer problem solving* (1984).

[8] Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT press (2018).

[9] Applegate, David L. *The traveling salesman problem: a computational study*. Vol. 17. Princeton university press (2006).

[10] Johnson, David S and McGeoch, Lyle A. "The traveling salesman problem: a case study." *Local search in combinatorial optimization* (1997): pp. 215–310.

[11] Dorigo, Marco and Di Caro, Gianni. "Ant colony optimization: a new meta-heuristic." *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, Vol. 2: pp. 1470–1477. 1999. IEEE.

[12] Pearl, Seth and Meisel, Nicholas A. "Assessing the manufacturability of students' early-stage designs based on previous experience with traditional manufacturing and additive manufacturing." *Journal of Mechanical Design* Vol. 146 No. 1 (2024): p. 012301.

[13] Pearl, Seth and Meisel, Nicholas A. "Exploring the Manifestation of Design for Manufacturing Heuristics in Students' Early-Stage Engineering Design Concepts." *Journal of Mechanical Design* Vol. 147 No. 3 (2025).

[14] Powell, Warren B. and Ryzhov, Ilya. *Optimal learning, 2nd Edition*. Wiley-Interscience, Hoboken, N.J. (2018).

[15] Tversky, A. and Kahneman, D. "Judgment under Uncertainty: Heuristics and Biases." *Science* Vol. 185 No. 4157 (1974): pp. 1124–1131.

[16] Gigerenzer, Gerd and Gaissmaier, Wolfgang. "Heuristic Decision Making." *Annual Review of Psychology* Vol. 62 No. 1 (2011): pp. 451–482.

[17] Griffiths, Thomas L., Lieder, Falk and Goodman, Noah D. "Rational Use of Cognitive Resources: Levels of Analysis Between the Computational and the Algorithmic." *Topics in Cognitive Science* Vol. 7 No. 2 (2015): pp. 217–229.

[18] Lieder, Falk and Griffiths, Thomas L. "Resource-rational analysis: understanding human cognition as the optimal use of limited computational resources." *Behavioral and Brain Sciences* (2019): p. 1–85.

[19] Lewis, Richard L., Howes, Andrew and Singh, Satinder. "Computational Rationality: Linking Mechanism and Behavior Through Bounded Utility Maximization." *Topics in Cognitive Science* Vol. 6 No. 2 (2014): pp. 279–311.

[20] Westbrook, Andrew and Braver, Todd S. "Cognitive effort: A neuroeconomic approach." *Cognitive, Affective, & Behavioral Neuroscience* Vol. 15 No. 2 (2015): pp. 395–415.

[21] Dorst, Kees and Cross, Nigel. "Creativity in the design process: co-evolution of problem–solution." *Design studies* Vol. 22 No. 5 (2001): pp. 425–437.

[22] Dorst, Kees. "The core of 'design thinking'and its application." *Design studies* Vol. 32 No. 6 (2011): pp. 521–532.

[23] Gero, John S and Kannengiesser, Udo. "The situated function–behaviour–structure framework." *Design studies* Vol. 25 No. 4 (2004): pp. 373–391.

[24] Polak, Elijah. *Optimization: algorithms and consistent approximations*. Vol. 124. Springer Science & Business Media (2012).

[25] Conn, Andrew R, Scheinberg, Katya and Vicente, Luis N. *Introduction to derivative-free optimization*. SIAM (2009).

[26] Queipo, Nestor V, Haftka, Raphael T, Shyy, Wei, Goel, Tushar, Vaidyanathan, Rajkumar and Tucker, P Kevin. "Surrogate-based analysis and optimization." *Progress in aerospace sciences* Vol. 41 No. 1 (2005): pp. 1–28.

[27] Bertsekas, Dimitri, Nedic, Angelia and Ozdaglar, Asuman. *Convex analysis and optimization*. Vol. 1. Athena Scientific (2003).

[28] Martins, Joaquim RRA and Ning, Andrew. *Engineering design optimization*. Cambridge University Press (2021).

[29] Rosenkrantz, Daniel J, Stearns, Richard E and Lewis, Philip M, II. "An analysis of several heuristics for the traveling salesman problem." *SIAM journal on computing* Vol. 6 No. 3 (1977): pp. 563–581.

[30] Croes, Georges A. "A method for solving traveling-salesman problems." *Operations research* Vol. 6 No. 6 (1958): pp. 791–812.

[31] Lin, Shen and Kernighan, Brian W. "An effective heuristic algorithm for the traveling-salesman problem." *Operations research* Vol. 21 No. 2 (1973): pp. 498–516.

[32] Holland, John H. "Genetic algorithms." *Scientific american* Vol. 267 No. 1 (1992): pp. 66–73.

[33] Kennedy, James and Eberhart, Russell. "Particle swarm optimization." *Proceedings of ICNN'95-international conference on neural networks*, Vol. 4: pp. 1942–1948. 1995. ieee.

[34] Rasmussen, Carl Edward and Williams, Christopher KI. *Gaussian processes for machine learning*. The MIT Press (2006).

[35] Bishop, Christopher M. *Pattern recognition and machine learning*. springer (2006).

[36] Jones, Donald R, Schonlau, Matthias and Welch, William J. "Efficient global optimization of expensive black-box functions." *Journal of Global optimization* Vol. 13 No. 4 (1998): pp. 455–492.

[37] Auer, Peter. "Using confidence bounds for exploitation-exploration trade-offs." *Journal of Machine Learning Research* Vol. 3 No. Nov (2002): pp. 397–422.

[38] Kahneman, Daniel and Tversky, Amos. "On the interpretation of intuitive probability: A reply to Jonathan Cohen." (1979).

[39] Gadi, Vikranth S, Szajnfarber, Zoe and Panchal, Jitesh H. "Developing Heuristics for Resource Allocation and Utilization in Systems Design: A Hierarchical Reinforcement Learning Approach." *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 88377: p. V03BT03A003. 2024. American Society of Mechanical Engineers.

[40] Gadi, Vikranth S, Topcu, Taylan G, Szajnfarber, Zoe and Panchal, Jitesh H. "Heuristics for Solver-Aware Systems Architecting: A Reinforcement Learning Approach." *Journal of Mechanical Design* Vol. 147 No. 2 (2025).

[41] Szajnfarber, Zoe, Hennig, Anthony, Mukherjee, Suparna, Rader, Steven and Crusan, Jason. "Linking solver characteristics, solving processes and solution attributes: A data explainer for an open innovation generated robotic design dataset." *Data in Brief* Vol. 50 (2023): p. 109547.

[42] Ulrich, Karl. "The role of product architecture in the manufacturing firm." *Research policy* Vol. 24 No. 3 (1995): pp. 419–440.

[43] Dharmarajan, Athul Chakkithara, Topcu, Taylan G, Panchal, Jitesh H and Szajnfarber, Zoe. "Valuing Outliers: a Modeling Framework to Consider Non-Traditional Solutions From Non-Traditional Solvers." *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 88407: p. V006T06A011. 2024. American Society of Mechanical Engineers.

[44] Shergadwala, Murtuza, Bilionis, Ilias, Kannan, Karthik N and Panchal, Jitesh H. "Quantifying the impact of domain knowledge and problem framing on sequential decisions in engineering design." *Journal of Mechanical Design* Vol. 140 No. 10 (2018).

[45] Chaudhari, Ashish M, Bilionis, Ilias and Panchal, Jitesh H. "Descriptive models of sequential decisions in engineering design: An experimental study." *Journal of Mechanical Design* Vol. 142 No. 8 (2020).

[46] Shannon, Claude E. "A mathematical theory of communication." *The Bell system technical journal* Vol. 27 No. 3 (1948): pp. 379–423.

[47] Feng, Wangchuan Bradley, Hélie, Sébastien and Panchal, Jitesh H. "Using Personas to Increase the Diversity of Design Concepts Generated by Large Language Models." *International Conference on-Design Computing and Cognition*: pp. 71–88. 2024. Springer.