University of Basel

Correlation Complexity Under Variant Descending and Dead-End Avoiding Heuristics

Master's Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence Research Group
https://ai.dmi.unibas.ch/

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Florian Pommerening

Nadine Sonderegger
nad.sonderegger@stud.unibas.ch
18-054-304

July 14, 2025

Abstract: Heuristics play a central role in classical planning by estimating the distance from a given state to the goal, thereby guiding search effectively. A key challenge is understanding how much information a heuristic must account for to avoid dead ends and ensure progress. This is captured by the concept of *correlation complexity*, which measures how many variables must be considered jointly. While various DDA (descending and dead-end avoiding) heuristics have been proposed, their correlation complexities and relative strengths across planning domains remain poorly understood. In this work, we establish a correlation complexity hierarchy among four variants (DDA, UDDA, ∞-DDA, and PDDA) and systematically analyse their behaviour across Blocksworld, Spanner, Logistics, and a modified Termes domain. We find that DDA, ∞-DDA, and PDDA share the same correlation complexity in Spanner and Logistics, with UDDA matching them in Logistics but being undefined in Spanner. In Blocksworld, ∞-DDA requires higher correlation complexity than DDA and PDDA. In modified Termes, we show evidence of correlation complexity 3. These findings are supported by constructing and validating potential heuristics using Fast Downward.

Acknowledgments

I am deeply grateful to my supervisor, Florian Pommerening, for his guidance throughout this thesis. His expertise, thoughtful feedback, and careful criticism of my writing were incredibly valuable. I'm especially grateful for his patience and the way he always helped me think through problems and find a path forward when I encountered difficulties. I'm also very grateful to Prof. Malte Helmert for giving me the opportunity to write my thesis within the Artificial Intelligence Research Group at the University of Basel. My gratitude extends to the members of the AI Research Group for generously taking the time to answer my questions and share their expertise whenever I needed support. Your help was very much appreciated. Lastly, I want to express my heartfelt appreciation to my family and friends for their constant support. Whether it was reading through my thesis drafts, offering feedback, or just listening when I needed to talk things through, your help means a lot to me.

Table of Contents

1 Introduction

In many areas of computer science and mathematics, understanding the complexity of a problem can be just as important as finding a solution. Instead of aiming for fast or efficient ways to solve a problem, we can focus on the quastion of how difficult it is to represent a solution in classical planning. A key question, then, is: *How complex must a heuristic be in order to guide a planner towards a solution?*

To explore this question, Seipp et al. [14] introduce the concept of *correlation complexity*, building on the framework of *potential heuristics* [12]. Potential heuristics estimate the cost to reach a goal by assigning weights to features (or facts) of a given state and summing these weights. Rather than focusing on the efficiency of finding a solution, correlation complexity quantifies the minimal amount of information a heuristic must encode to satisfy two key properties: it must strictly decrease along every solution path (*descending*) and avoid misleading the planner into unsolvable states (*dead-end avoiding*). Heuristics that satisfy both properties are known as *DDA heuristics*. Additionally, Seipp et al. [14] introduce simple operator-based criteria that can be used to establish lower bounds on the correlation complexity of planning tasks.

Extending this idea, Dold and Helmert [2] introduce a new, more general approach based on *states* rather than operators to detect lower bounds on correlation complexity. A central concept in their approach is the quartet criterion, which provides a way to check whether a potential heuristic can be represented in just two dimensions. This test looks for a special configuration of four states, called a witnessing quartet, that demonstrates the heuristic's dimension must be at least 2. The quartet criterion thus offers a useful tool for understanding the minimal complexity necessary for heuristics in planning tasks. Building on this, they further generalize their approach to establish lower bounds for arbitrary correlation complexity, making it possible to analyse heuristics with even higher-dimensional representations. By generalizing their approach, they extend its reach and offer enhanced insight into the correlation complexity of heuristics in planning.

While the original definition of DDA heuristics offers valuable insights, it has notable limitations: it only imposes constraints on solvable tasks. As a result, any arbitrary heuristic for an

unsolvable task can still satisfy the DDA criteria. To address these limitations, several refined variants of DDA heuristics are introduced, such as unrestricted DDA (UDDA), ∞ -DDA, and predicate-based pruning DDA (PDDA) [6]. Each variant captures distinct characteristics of planning domains and their solutions, making it possible to analyse how these differences impact correlation complexity.

In this thesis, we investigate the hierarchy of different DDA heuristic variants (DDA, ∞-DDA, UDDA, and PDDA) and examine their correlation complexities across Blocksworld, Spanner, Logistics, and a modified version of Termes. Our results show that in both Spanner and Logistics, the correlation complexities for DDA, ∞-DDA, and PDDA are the same. Moreover, the Logistics domain maintains this consistency for UDDA as well, while no UDDA heuristic exists for Spanner. In contrast, Blocksworld reveals an increase in the correlation complexity: from 2 for DDA and PDDA to 3 for ∞-DDA. For the modified Termes domain, we present evidence indicating a correlation complexity of 3. These findings are supported through the construction of concrete potential heuristics, whose properties we validate within the Fast Downward planning system.

The thesis is structured as follows: Chapters 2 and 3 provide the necessary background. Chapter 4 introduces and establishes the hierarchy among the DDA variants. Chapters 5 to 8 offer domain-specific analyses, and Chapter 9 summarizes our experimental results.

To enhance clarity, style, and coherence throughout the thesis, large language models were extensively used for linguistic refinement and grammatical consistency. These models supported language improvement through rephrasing, grammar correction, and stylistic adjustments of text that was written independently. At no point was any technical, conceptual, or scientific content generated by the models. All ideas, results, arguments, and interpretations presented are entirely original. The language models served strictly as writing aids, not as sources of content.

2
Background

This chapter provides the necessary foundation for understanding the concepts and methodologies that are central to this thesis. It introduces planning tasks, which serve as the basis for automated planning; potential heuristics, a key approach for evaluating planning tasks; and the various descending and dead-end avoiding (DDA) properties that form the focus of this work. Finally, the chapter discusses correlation complexity, which is used to assess the difficulty of representing solutions in classical planning and serves as the primary measure in this thesis.

2.1 Planning Tasks

Planning tasks in Artificial Intelligence involve finding a sequence of operators that transitions an agent from an initial state to a goal state. These tasks are formally represented as search problems in a state space, where each state corresponds to a configuration of the environment, and operators define transitions between states. A classical SAS$^+$ planning task [1] is typically represented as $\Pi = \langle V, O, I, g \rangle$ with the following components:

- V is a finite set of state variables. Each *state variable* $v \in V$ has a finite *domain* $dom(v)$. An *atom* is a pair $\langle v, d \rangle$ where $v \in V$ and $d \in dom(v)$. A set of atoms is called consistent if all atoms correspond to different variables. A partial variable assignment is a consistent set of atoms. We refer to such partial assignments as *features*. We write $vars(f)$ to denote the set of variables involved in a feature f. If $vars(f) = V$, then f is a state. A state s is consistent with a partial variable assignment f if $f \subseteq s$. In this case, we say that the feature f is *active* (or *present*) in s.

- O is a finite set of operators. An operator $o \in O$ is given as $\langle pre(o), eff(o) \rangle$, where $pre(o)$ and $eff(o)$ are partial variable assignments. An operator o is applicable in a state s if s is consistent with $pre(o)$. When an applicable operator o is *applied* in s, it produces a new state, called the successor state, denoted $s[\![o]\!]$. This successor state is derived by updating s according to the effects $eff(o)$, which specify changes to certain variables in s.

- I is the initial state.

- g is the goal condition. It is a partial variable assignment. A state s satisfies the goal if it is consistent with g. We denote the set with all states that are consistent with g by G.

It is useful to understand certain characteristics of states in the search space. For a state s, an *s-plan* is a sequence of operators from s to a goal state $s_\star \in G$, meaning the goal can be achieved from s. Formally, an s-plan $\langle o_1, \ldots, o_n \rangle$ is a finite sequence of operators such that $s[\![o_1]\!][\![o_2]\!] \ldots [\![o_n]\!]$ is a goal state. A state s is considered *solvable* if an s-plan exists and *unsolvable* otherwise. A task Π is solvable if the initial state I is solvable. A state s is defined as *reachable* if there exists a sequence of operators that leads from the initial state I to s, i.e., $\langle V, O, I, s \rangle$ is solvable. A state is referred to as an *alive state* if it is reachable, solvable, and not a goal state.

In classical planning, a *planning domain* describes a general environment in which various planning tasks can be formulated. A planning domain defines the predicates, actions, and constraints that govern possible interactions but does not specify a particular initial or goal state. Domains provide a framework for creating multiple planning tasks using domain-independent algorithms. The *Planning Domain Definition Language (PDDL)*, developed in the late 1990s [8], standardized this approach by providing a common language to define planning domains and tasks, which can then be translated into SAS$^+$ tasks to enable compatibility with various classical planning systems.

Example 1 (Blocksworld)**.** In the *Blocksworld domain* [15], a set of blocks denoted as $\mathcal{B}$ must be rearranged from an initial configuration into a specified goal arrangement. Each block can either be placed on a table or stacked on another block. Operators allow moving a block from one block to another, from a block to the table, or from the table onto a block.
The planning task in this Blocksworld domain can be represented formally as $\Pi = \langle V, O, I, g \rangle$:

- **State Variables (V)**:

  For all $A \in \mathcal{B}$ there are the following two variables:

  - pos_A: The location of block A, where $\text{dom}(pos_A) = (\{T\} \cup \mathcal{B}) \setminus \{A\}$, meaning that A is either on the table or on another block B.
  - $clear_A$: Whether block A has no block on top of it, with domain $\text{dom}(clear_A) = \{yes, no\}$.

  An example state could be $\{\langle pos_A, B\rangle, \langle clear_A, yes\rangle, \langle pos_B, T\rangle, \langle clear_B, no\rangle\}$, meaning block A is on B, B is on the table, A is clear and B is not.

- **Operators (O)**:

  - *move-to-T*$(A, B) = \langle \{pos_A = B, clear_A = yes, \}, \{pos_A := T, clear_B := yes\}\rangle$ moves block A from on top of block B and places it on the table.
  - *move-to-B*$(A, B) = \langle \{clear_A = yes, clear_B = yes\}, \{pos_A := B, clear_B := no\}\rangle$ moves block A onto block B.

- **Initial State (I)**:
  An example initial state could be $\{\langle pos_A, B\rangle, \langle pos_B, T\rangle, \langle pos_C, T\rangle, \langle clear_A, yes\rangle,$

$\langle clear_B, no\rangle, \langle clear_C, yes\rangle\}$, where block A is on B, B is on the table, and C is on the table, with A and C being clear.

- **Goal Condition (g)**:
  The goal might be $\{\langle pos_A, T\rangle, \langle pos_B, C\rangle, \langle clear_A, yes\rangle, \langle clear_B, yes\rangle\}$, indicating that block A should be on the table and block B should be on C, with both A and B being clear.

Starting from I, a valid plan could involve first moving block A from B to the table using the *move-to-T(A, B)* operator. Then, block B is moved onto block C using the *move-to-B(B, C)* operator. This sequence defines an *s-plan* that transitions the initial state I to a state satisfying the goal condition g.

2.2 Potential Heuristics

A heuristic is a function $h : S \to \mathbb{R}^+$ that estimates the cost from a given state to the nearest goal state. Pommerening et al. [12] introduced a new class of heuristics, namely *potential heuristics*. A potential heuristic assigns a numerical weight to each feature of a planning task. The heuristic value of a state can then be calculated from the sum of the weights of the feature present in that state. This simple structure allows potential heuristics to be evaluated very efficiently, making them a practical choice in many domains.

In this context, we define *features* as partial assignments of variables. We say that a state s *has the features* F if s is consistent with F. Additionally, we use *Iverson brackets* $[P]$ to represent indicator functions. Specifically, this function takes the value 1 for variable assignments where the statement P is true, and 0 otherwise.

Definition 1 (Potential Heuristic [12])**.** Let Π be a planning task, $\mathcal{F}$ be a set of state features of Π, and $w : \mathcal{F} \to \mathbb{R} \cup \{\infty\}$. The *potential heuristic* with features $\mathcal{F}$ and weight function w maps each state s to:

$$h_{pot}(s) = \sum_{F \in \mathcal{F}} w(F)[F \subseteq s]$$

The complexity of a potential heuristic is measured by the size of the largest conjunction used as a feature, called its *dimension*. In formal terms, a potential function with features $\mathcal{F}$ has dimension $\max_{F \in \mathcal{F}} |F|$.

2.3 Descending and Dead-End Avoiding Heuristics (DDA)

In complex planning tasks, search processes often encounter *dead ends*, states from which reaching the goal is impossible, or states that do not significantly contribute to goal progression. descending and dead-End avoiding (DDA) heuristics aim to address these challenges by guiding the search towards states with lower heuristic values and avoiding paths that may lead to unsolvable areas.

Definition 2 (Descending Heuristic)**.** A heuristic h is *descending* if, for all alive states s that are not goal states, there exists a successor state s' such that $h(s') < h(s)$.

In other words, a descending heuristic ensures that, at every step, there is at least one neighbouring state with a lower heuristic value, guiding the search closer to the goal.

Definition 3 (Dead-End Avoiding Heuristic)**.** A heuristic h *avoids dead ends* if every improving successor of an alive state is solvable. Formally, for all states $s \in S$ and their successors $s' \in \text{succ}(s)$:

$$s \text{ is alive and } h(s') < h(s) \implies s' \text{ is solvable.}$$

This property ensures that any state transition to a lower heuristic value (i.e. an improving successor) from an alive state will always lead to a state from which the goal remains reachable, thereby avoiding paths that lead into dead ends.

Together, these properties define the DDA heuristic framework, introduced by Seipp et al. [14], which guides the process towards finding a solution by focusing on states with progressively lower heuristic values, while avoiding dead ends in complex state spaces.

2.4 Landmarks

Landmarks, first introduced by Porteous et al. [13], are facts that must be true at some point in every valid plan. They represent necessary conditions that any solution must satisfy and can be extended into two key types: *state landmarks* and *action landmarks*.

A state landmark is a complete state that must be reached in every plan before achieving the goal. These serve as intermediate stages that cannot be bypassed. While the initial and goal states are trivial examples, more complex state landmarks can be derived through domain structure and causal analysis.

An action landmark, introduced by Zhu and Givan [16], is an action that must appear in all valid plans. Unlike fact landmarks, which concern conditions, action landmarks identify operations that are unavoidable for goal achievement.

3
Previous Work

3.1 DDA Variants

An interesting detail of the DDA definition is pointed out by Helmert et al. [6], namely that it only applies constraints to solvable tasks. For unsolvable tasks, since there are no reachable states, any heuristic can technically satisfy the DDA conditions. For example, even a constant heuristic that immediately stalls without finding an improved successor counts as descending and dead-end-avoiding in unsolvable tasks. This aspect reveals a potential limitation of DDA, as it combines two different properties: the ability to solve solvable tasks by local search and the handling of unsolvable tasks.

The following sections introduce three different variants of the DDA heuristic proposed by Helmert et al. [6]. Each variant represents a different property of a heuristic, based on how it deals with dead-ends in the search space.

Unrestricted DDA Heuristics (UDDA)

Unrestricted descending and dead-end avoiding (UDDA) heuristics are the simplest form of DDA heuristics. These heuristics guarantee that the search remains goal-directed by ensuring that every non-goal state has a successor with a lower heuristic value.

Definition 4 (UDDA heuristic)**.** A heuristic h is UDDA if it satisfies the following condition for all states $s \in S$ (where S is the set of all states):

$$\forall s \in (S \setminus G)\ \exists s' \in \mathrm{succ}(s) : h(s') < h(s)$$

This condition ensures that every non-goal state has a successor with a lower heuristic value, effectively guiding the search towards the goal by continually improving the heuristic value. Since UDDA treats all states equally and does not distinguish between solvable and unsolvable states, it applies the descending condition universally. This approach makes UDDA a simpler property than DDA, as it does not require the heuristic to check whether states are alive (i.e. both reachable from the initial state and solvable).

However, this simplicity comes with a key limitation: UDDA is less powerful because it cannot handle tasks that contain unsolvable states. If the search encounters an unsolvable area, it may continue indefinitely, unable to determine that no solution exists. Thus, UDDA heuristics are only effective in fully solvable state spaces.
To address this limitation, more advanced variants like ∞-DDA and PDDA introduce mechanisms for identifying and avoiding unsolvable states, extending applicability to more complex tasks.

∞-DDA Heuristics

The ∞-DDA heuristic extends UDDA by incorporating the notion of dead-end avoidance while still allowing for descending behaviour. The primary idea behind ∞-DDA is to assign infinite heuristic values to states that are considered dead ends. This adjustment ensures that the search avoids these dead-end states by treating them as unreachable, while maintaining the descent condition in solvable areas.

In the original definition of potential heuristics [12], the heuristic could not assign an infinite value to any state. This limitation meant that dead-end states were not explicitly handled and remained part of the search space, even though they could not contribute to goal progression. However, this limitation was later addressed by modifying the potential heuristic definition to allow the assignment of infinite values to dead-end states.

Definition 5 (∞-DDA heuristic)**.** A heuristic h is ∞-DDA if it satisfies the following conditions:

$$\forall s \in (S_{\text{fin}} \setminus G)\ \exists s' \in \text{succ}(s) : h(s') < h(s),$$

$$I \in S_{\text{fin}},$$

where S_{fin} denotes the set of states with finite heuristic values. This ensures that the search avoids dead ends while maintaining the descent condition in solvable regions.

The introduction of infinite heuristic values resolves one of the primary shortcomings of UDDA by providing a clear mechanism to avoid dead ends, but it still relies on the heuristic's ability to define the alive state space accurately. However, ∞-DDA can still be limited in situations where identifying dead ends requires more nuanced decision-making than simply assigning infinite values.

Predicate-Based Pruning DDA Heuristics (PDDA)

Predicate-based pruning descending and dead-end avoiding (PDDA) heuristics generalize the ∞-DDA approach by introducing a predicate-based pruning mechanism for identifying dead-end states. Unlike ∞-DDA, which relies on a single potential function to assign infinite heuristic values to dead ends, PDDA uses two distinct potential functions to separately determine finite heuristic values and dead-end conditions.

Definition 6 (PDDA heuristic)**.** a PDDA heuristic uses two potential functions h_{pot1} and h_{pot2}, and is defined as:

$$h(s) = \begin{cases} \infty & \text{if } h_{\text{pot2}}(s) > 0, \\ h_{\text{pot1}}(s) & \text{otherwise.} \end{cases}$$

Additionally, the heuristic satisfies the following properties:

$$\forall s \in (S_{\text{fin}} \setminus G)\ \exists s' \in \text{succ}(s) : h(s') < h(s),$$

$$I \in S_{\text{fin}},$$

where S_{fin} denotes the set of states with finite heuristic values.
In this formulation, $h_{\text{pot1}}(s)$ is used to assign the heuristic value for non-dead-end states, while $h_{\text{pot2}}(s)$ is used to determine whether a state should be considered a dead end (i.e., assigned an infinite heuristic value). If a state s satisfies the predicate $h_{\text{pot2}}(s) > 0$, it is treated as a dead end and assigned an infinite heuristic value. Otherwise, the heuristic is derived from $h_{\text{pot1}}(s)$.

This predicate-based pruning approach allows for greater flexibility and accuracy in dead-end detection, especially in tasks where dead ends are defined by complex conditions that a single heuristic function may not capture. For instance, h_{pot2} can incorporate conditions specific to the task, such as resource constraints or state properties indicating unsolvability.

By decoupling the heuristic values from dead-end detection, PDDA heuristics can adapt to a wider range of problem domains than ∞-DDA heuristics. However, this flexibility requires careful design of h_{pot2} to ensure it accurately captures meaningful dead-end conditions, making PDDA a powerful yet sophisticated tool for planning in complex state spaces.

3.2 Correlation Complexity

To further understand the demands of a planning task, Seipp et al. [14] introduce *correlation complexity*, which evaluates the minimum dimensional complexity required for a DDA heuristic to solve a task. Based on potential heuristics, correlation complexity provides a measure of how many features or conditions must be considered simultaneously to reliably guide the search.

Definition 7 (correlation complexity of a planning task)**.** The correlation complexity of a planning task Π is the minimum dimension d for which there exists a potential heuristic that is both descending and dead-end avoiding for Π.

Intuitively, correlation complexity reflects how many state features an agent must evaluate simultaneously to determine the best successor state. It is always bounded above by the number of state variables, n, since, in the worst case, a potential heuristic can assign a feature with weight $h^*(s)$ to every state s. Because each state has $|s| = n$ variables, this results in a heuristic of dimension of at most n. Here, $h^*(s)$ represents the perfect heuristic, which gives the exact cost from state s to the goal. This ensures that the correlation complexity of any planning task is well-defined. As long as not all reachable states are goal states, the correlation complexity is at least 1. Seipp et al. [14] identify criteria that indicate when a planning task has a correlation

complexity of at least 2. To generalize this concept, correlation complexity has been extended from individual tasks to entire planning domains.

Definition 8 (correlation complexity of a planning domain)**.** The correlation complexity of a planning domain is the highest correlation complexity among all planning tasks within the domain. If the correlation complexities of the tasks are unbounded, the domain's correlation complexity is ∞.

In their analysis of several common planning domains, they found that many domains exhibit a relatively low correlation complexity of 2. These domains, such as *Spanner*, *Gripper*, *VisitAll*, and *Blocksworld*, can be solved with heuristics that consider only pairs of state features. This indicates that these domains do not require highly complex heuristics to avoid local minima and navigate towards the goal.

Witnessing Quartet

A formal tool for proving a lower bound on correlation complexity is introduced by Dold and Helmert [2] through the concept of a *witnessing quartet*. This construction captures specific interactions among variables that demonstrate the necessity of at least two-dimensional potential heuristics.

Definition 9 (Witnessing Quartet [2])**.** Let $\Pi = \langle V, O, I, g \rangle$ be a planning task and h a heuristic for Π. A pair $\langle [a, b, c, d], [W, M] \rangle$ is a *witnessing quartet* for h if a, b, c, d are states in Π, and $\{W, M\}$ is a partition of the variables V, such that:

$$h(a) > h(b), \quad a^W = b^W, \quad a^M = d^M,$$
$$h(c) \geq h(d), \quad c^W = d^W, \quad b^M = c^M.$$

The witnessing quartet provides a structural pattern that potential heuristics must respect to remain consistent and dead-end avoiding. If a heuristic assigns values to a set of states in a way that forms a witnessing quartet, this immediately implies a lower bound on the heuristic's dimensionality.

Theorem 1 (Quartet Criterion [2])**.** *Let* Π *be a planning task, and let* h *be a potential heuristic defined over* Π*. If there exists a witnessing quartet for* h*, then* $\dim(h) \geq 2$*.*

This result links directly to correlation complexity by certifying that certain planning tasks cannot be solved using heuristics of dimension 1. The witnessing quartet thus acts as a diagnostic tool for identifying minimum heuristic complexity, and supports the broader theoretical foundations of correlation complexity.

Building on previous work, we examine how the correlation complexity varies under different DDA heuristics, specifically UDDA, ∞-DDA, and PDDA. To formalize this, we introduce a new notation: when referring to the DDA variant correlation complexity, we mean the correlation complexity evaluated under that specific DDA heuristic.

4

Comparison of DDA Variants

In this chapter, we focus on comparing the original DDA definition, the *unrestricted DDA (UDDA)*, the *∞-DDA*, and the *predicate-based pruning DDA (PDDA)*. While these variants were introduced in Section 3.1, our focus here is on understanding how their properties influence one another, particularly in terms of different correlation complexity. By analysing these differences, we gain insights into how each heuristic interacts with the structure of the planning problem.

4.1 UDDA vs. DDA, ∞-DDA and PDDA

A first result in this comparison is that UDDA heuristics always satisfy the conditions of other DDA variants, meaning that UDDA heuristics can be transformed into DDA, ∞-DDA, and PDDA heuristics.

Lemmas 1. *Every UDDA heuristic can be transformed into a DDA, ∞-DDA and PDDA heuristic.*

Proof. Let Π be a planning task and let h be a UDDA heuristic. By definition of UDDA (Definition 4), the heuristic h ensures that for every non-goal state s, there exists a successor state s' such that $h(s') < h(s)$.

- UDDA implies DDA:
  Since every non-goal state has a successor with a strictly lower heuristic value, it follows that every alive state (i.e., a state that belongs to a path leading to a goal) also has such a successor. This property directly satisfies the definition of a DDA heuristic, implying that h is a DDA heuristic.

- UDDA implies ∞-DDA:
  Because h provides a strictly decreasing sequence of heuristic values along at least one path to a goal from every state, it guarantees that all states are guided towards a solution. Consequently, no state requires an infinite heuristic value to prevent dead ends, meaning that h is also an ∞-DDA heuristic.

- UDDA implies PDDA:
  To transform h into a PDDA heuristic, we define $h_{pot1} = h$ and set the potential function h_{pot2} to a constant negative value (e.g., $h_{pot2}(s) = -1$ for all states s). This construction ensures that h satisfies the requirements of a PDDA heuristic.

Thus, every UDDA heuristic can be transformed into a DDA, ∞-DDA, and PDDA heuristic. □

Therefore, in cases where UDDA heuristics exist, the DDA, ∞-DDA, and PDDA correlation complexity is always upper bounded by the UDDA correlation complexity. This is because every UDDA heuristic satisfies the conditions of DDA, ∞-DDA, and PDDA (as shown in Lemma 1), meaning that any heuristic from these variants can be derived from a UDDA heuristic.

To further illustrate the relationship between the correlation complexities of the different DDA variants, we present an example using a modified Gray code [4]. The Gray code is a well-known encoding scheme where consecutive numbers differ by only one bit. This example will show how the correlation complexities of DDA, ∞-DDA, and PDDA heuristics can be lower than that of a UDDA heuristic.

Lemmas 2. *There are tasks where the DDA, ∞-DDA, and PDDA correlation complexities are lower than the UDDA correlation complexity.*

Proof. We modify the Gray code [4] example used by Seipp et al. [14]. The Gray code graph represents a systematic way of encoding binary sequences such that consecutive numbers differ by only one bit, making it a useful structure for analysing correlation complexities.

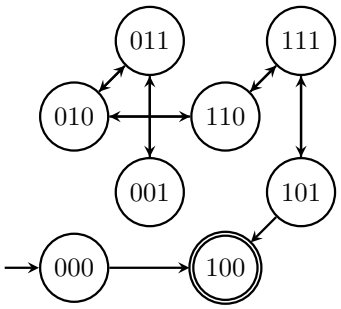
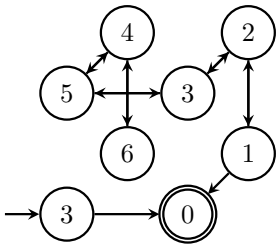


Figure 4.1: State space representation of a planning task using a modified Gray code. The task consists of three binary variables, v_1, v_2, and v_3. Each node, labelled xyz, represents the state $\langle v_1, x\rangle, \langle v_2, y\rangle, \langle v_3, z\rangle$. Directed edges indicate operators that have three preconditions and one effect. The initial state is 000, and the goal state is 100.

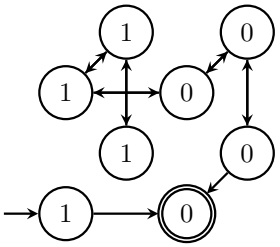
Consider the state space in Figure 4.1, where each node represents a binary string of length three, and directed edges indicate valid state transitions. On the left, we display heuristic values within the nodes, while on the right, we define weight functions for different heuristics:

UDDA: Applying Theorem 5 from Seipp et al. [14], we observe that whenever a planning task contains critical operators that are inverses of each other, its correlation complexity must be at least 2. In the modified Gray code graph, the operators responsible for flipping the bit values of v_2 create such inverse relationships, thereby establishing the lower bound for UDDA. The weight function for a UDDA heuristic with dimension 2 is defined as follows:



$$w(\{\langle v_1, 0\rangle\}) = 3$$
$$w(\{\langle v_1, 1\rangle \langle v_2, 1\rangle\}) = 2$$
$$w(\{\langle v_1, 0\rangle \langle v_3, 1\rangle\}) = 2$$
$$w(\{\langle v_2, 0\rangle \langle v_3, 1\rangle\}) = 1$$
$$w(\{\langle v_2, 1\rangle \langle v_3, 0\rangle\}) = 1$$

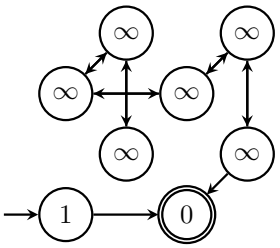
DDA: Since DDA only requires an improved successor for all alive states, it can assign weights based on a simple condition. Specifically, it assigns a weight of 1 when $v_1 = 0$ and a weight of 0 otherwise. This results in a straightforward evaluation with a DDA correlation complexity of 1. The weight function is:



$$w(\{\langle v_1, 0\rangle\}) = 1$$

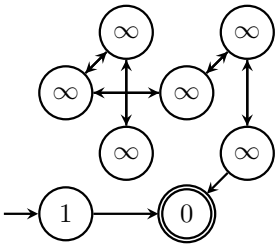
We cannot directly apply this heuristic to ∞-DDA and PDDA. Both variants require a strictly improving successor for every state with a finite heuristic value. Under this heuristic, state *001* has a heuristic value of 1, implying that its successor must have a lower value. However, this would result in a DDA correlation complexity of 2, as seen in UDDA. To address this, we define new heuristics that assign infinity to specific states.

∞-DDA: The ∞-DDA heuristic assigns infinite weights to the second and third variables when they are set to 1, while maintaining the same weights as the DDA heuristic for the first variable. This also leads to a ∞-DDA correlation complexity of 1. The weight function is:



$$w(\{\langle v_2, 1\rangle\}) = \infty$$
$$w(\{\langle v_3, 1\rangle\}) = \infty$$
$$w(\{\langle v_1, 0\rangle\}) = 1$$

PDDA: The PDDA heuristic assigns positive weights to variables v_2 and v_3 when they are 1, using the potential function h_{pot2}, while maintaining the same weights for h_{pot1} as the DDA heuristic. This also results in a PDDA correlation complexity of 1. The weight functions are:



$$\begin{aligned} \text{for } h_{pot2}: \quad & w_2(\{\langle v_2, 1\rangle\}) = 1 \\ & w_2(\{\langle v_3, 1\rangle\}) = 1 \\ \text{for } h_{pot1}: \quad & w_1(\{\langle v_1, 0\rangle\}) = 1 \end{aligned}$$

In summary, the DDA, ∞-DDA, and PDDA correlation complexity maintains 1, whereas the UDDA correlation complexity is 2 due to dependencies between variable pairs.

□

4.2 ∞-DDA vs. PDDA

Another key result in this comparison is that ∞-DDA heuristics can always be transformed into a PDDA heuristic.

Lemmas 3. *Every ∞-DDA heuristic can be transformed into a PDDA heuristic.*

Proof. As noted by Helmert et al. [6], the *predicate-based pruning DDA (PDDA)* heuristic is a generalization of the *∞-DDA* heuristic. In this proof, we show that any ∞-DDA heuristic can indeed be transformed into a PDDA heuristic.
An ∞-DDA heuristic is defined by a weight function $w_\infty : \mathcal{F} \to \mathbb{R} \cup \{\infty\}$, where:

- Features assigned a finite weight, i.e., $w_\infty(f) \in \mathbb{R}$, contribute to the heuristic value as usual.
- Features assigned an infinite weight, i.e., $w_\infty(f) = \infty$, define dead-end states where the heuristic value must be infinite.

To transform this into a PDDA heuristic, we construct two potential heuristics, h_{pot1} and h_{pot2}, with corresponding weight functions $w_1, w_2 : \mathcal{F} \to \mathbb{R}$, as follows:

Handling Dead-End Features: Let $\mathcal{F}_\infty$ be the set of features assigned an infinite weight in the ∞-DDA heuristic, i.e., $\mathcal{F}_\infty = \{f \in \mathcal{F} \mid w_\infty(f) = \infty\}$.
All features $f \in \mathcal{F}_\infty$ that were assigned an infinite weight in the ∞-DDA heuristic are transferred to the second potential heuristic, h_{pot2}. Instead of assigning infinity, we assign a constant $c > 0$:

$$w_2(f) = c \quad \text{for all } f : w_\infty(f) = \infty.$$

This ensures that any state containing such a feature still evaluates to infinity in the PDDA framework.

Handling Non-Dead-End Features: The remaining features, which had finite weights in the ∞-DDA heuristic, are retained in the first potential heuristic, h_{pot1}, with their original weights:

$$w_1(f) = w_\infty(f) \quad \text{for all } f : w_\infty(f) \in \mathbb{R}.$$

To complete the proof, we show that the resulting PDDA heuristic correctly combines h_{pot1} and h_{pot2} to preserve the ∞-DDA heuristic's behaviour. A PDDA heuristic uses two potential functions h_{pot1} and h_{pot2}, and is defined as:

$$h(s) = \begin{cases} \infty & \text{if } h_{\text{pot2}}(s) > 0, \\ h_{\text{pot1}}(s) & \text{otherwise.} \end{cases}$$

This ensures that the states that were assigned an infinite heuristic value in the original ∞-DDA heuristic (due to the features in $\mathcal{F}_\infty$) will still evaluate to infinity, while all other states will retain the heuristic values as determined by h_{pot1}.
Therefore, the transformation preserves the heuristic values and satisfies the PDDA properties, proving that every ∞-DDA heuristic can be expressed as a PDDA heuristic.

□

As a result, the PDDA correlation complexity is always upper bounded by the ∞-DDA correlation complexity. This is because every ∞-DDA heuristic can be transformed into a PDDA heuristic (as shown in Lemma 3), meaning that a PDDA heuristic can be derived from a ∞-DDA heuristic. Thus, the PDDA correlation complexity heuristics cannot exceed the ∞-DDA correlation complexity.
To further illustrate the relationship between the ∞-DDA and PDDA correlation complexity, we present an example in Figure 4.2. It will show how the PDDA correlation complexity can be lower than the ∞-DDA correlation complexity.

Lemmas 4. *There are tasks where the PDDA correlation complexity is lower than the ∞-DDA correlation complexity.*

Proof. Consider the Blocksworld domain, which will be formally introduced in Chapter 5. As we prove in Theorem 3, the ∞-DDA correlation complexity in this domain is 3. In contrast, Theorem 4 demonstrates that the PDDA correlation complexity is only 2. This establishes that, for certain tasks, the PDDA correlation complexity can be strictly lower than that of ∞-DDA.

□

4.3 DDA vs. ∞-DDA

When comparing DDA and ∞-DDA, we observe that neither can be universally transformed into the other. In other words, for certain problems, the DDA correlation complexity is higher than the ∞-DDA correlation complexity, while for others, the opposite holds. The following lemmas illustrate this contrast.

Lemmas 5. *There are tasks where the DDA correlation complexity is lower than the ∞-DDA correlation complexity.*

Proof. Consider the Blocksworld domain, which will be formally introduced in Chapter 5. Seipp et al. [14] establish that the DDA correlation complexity for Blocksworld is 2. However, as we will prove in Theorem 3, the ∞-DDA correlation complexity in this domain is 3. This shows that

the DDA correlation complexity can be strictly lower than the ∞-DDA correlation complexity for certain tasks.

□

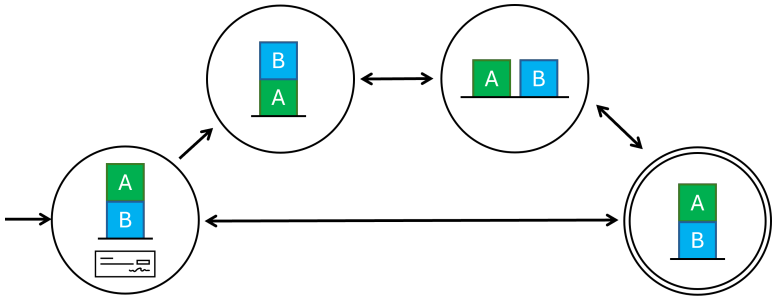


Figure 4.2: State space representation of a planning task in which the agent receives a coupon at the start and must decide whether to use it. Using the coupon leads to a Blocksworld subproblem that must be solved, while discarding it allows the agent to reach the goal directly. This example illustrates the concept for two blocks, but the approach generalizes to an arbitrary number of blocks.

Lemmas 6. *There are tasks where the ∞-DDA correlation complexity is lower than the DDA correlation complexity.*

Proof. Consider the planning task represented in Figure 4.2. The state space contains a decision point where the agent chooses either to use a coupon, forcing it to solve a Blocksworld subproblem (as described in Example 1), or to discard the coupon and transition directly to the goal. This task extends Blocksworld by introducing a new variable, *coupon*, which is set to *no* in all states except the initial one, where it is set to *yes*. The initial state is the goal configuration of the Blocksworld task, but with the coupon present. Effectively, this modification doubles the Blocksworld problem, as the *coupon* variable can take two values, but only the initial state has the *coupon* set to *yes*. In all other states, the coupon is *no*, meaning the variable can be treated as a constant during heuristic evaluation. Consequently, the addition of this variable does not affect the DDA correlation complexity. Thus, it remains unchanged from the original Blocksworld problem. Seipp et al. [14] show that the DDA correlation complexity for Blocksworld is 2, and this result directly carries over to the extended task.

In contrast, an ∞-DDA heuristic ensures that the only possible action is to discard the coupon and reach the goal. To achieve this, we assign an infinite weight to all states in which a block has any variable assigned differently than in the initial state, except for the coupon. This enforces that the transition into the Blocksworld subproblem is impossible, leaving discarding the coupon as the only viable option. Formally, we define the weight function as follows:

$$
\begin{aligned}
&w(\{\langle pos_X, Y\rangle\}) = \infty && \text{for all } \langle pos_X, Y\rangle \notin I \\
&w(\{\langle clear_X, Z\rangle\}) = \infty && \text{for all } \langle clear_X, Z\rangle \notin I \\
&w(\{\langle coupon, yes\rangle\}) = 1
\end{aligned}
$$

With this weighting scheme, only the initial and goal states are assigned finite weights. Consequently, the only operator that remains applicable without incurring an infinite cost is discarding the coupon, which directly leads to the goal. As a result, the heuristic avoids evaluating the internal structure of the Blocksworld subproblem, establishing an ∞-DDA correlation complexity of 1. □

4.4 DDA vs. PDDA

When comparing DDA and PDDA, we observe that neither can be universally transformed into the other. In other words, for certain problems, the DDA correlation complexity is higher than the PDDA correlation complexity, while for others, the opposite holds. The following two lemmas illustrate this contrast.

Lemmas 7. *There are tasks where the PDDA correlation complexity is lower than the DDA correlation complexity.*

Proof. Consider the planning task represented in Figure 4.2. As shown in Lemma 6, the DDA correlation complexity of the planning task is 2 while the ∞-DDA correlation complexity for the same task is 1. By Lemma 3, we can transform the ∞-DDA heuristic into a PDDA heuristic, thereby establishing a PDDA correlation complexity of 1.

□

The previous lemma demonstrates that, in some cases, PDDA heuristics can achieve lower correlation complexity than DDA heuristics. However, the relationship between DDA and PDDA correlation complexities is not one-sided. In the following lemma, we present a complementary result: there are planning tasks where DDA heuristics require strictly lower correlation complexity than PDDA heuristics.

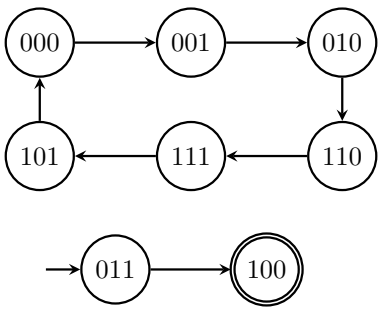


Figure 4.3: State space representation of a planning task consisting of three binary variables, v_1, v_2, and v_3. Each node, labelled xyz, represents the state $\langle v_1, x\rangle, \langle v_2, y\rangle, \langle v_3, z\rangle$. Directed edges indicate operators. The initial state is 011, and the goal state is 100.

Lemmas 8. *There are tasks where the DDA correlation complexity is lower than the PDDA correlation complexity.*

Proof. Consider the planning task illustrated in Figure 4.3. Define a heuristic h based on the following weight function:

$$w(\{\langle v_1, 0\rangle\} = 1.$$

This heuristic satisfies the DDA property. Specifically, since the only two alive states in this task are the initial state and the goal state, and the heuristic is descending along the path

between them, it is trivially DDA. Hence, the DDA correlation complexity of this task is 1.

We now show that the PDDA correlation complexity of the same task is at least 2, using the witnessing quartet method (see Definition 9). Assume for contradiction that there exists a 1-dimensional potential function h_{pot} (defined as in Definition 6) satisfying the PDDA property. Consider the following four states (as shown in Figure 4.3):

$$s_1 : \{\langle v_1, 0\rangle, \langle v_2, 1\rangle, \langle v_3, 1\rangle\} \qquad s_2 : \{\langle v_1, 1\rangle, \langle v_2, 0\rangle, \langle v_3, 0\rangle\}$$
$$s_3 : \{\langle v_1, 0\rangle, \langle v_2, 1\rangle, \langle v_3, 0\rangle\} \qquad s_4 : \{\langle v_1, 1\rangle, \langle v_2, 0\rangle, \langle v_3, 1\rangle\}$$

The initial state s_1 and the goal state s_2 are both solvable, so their heuristic values for h_{pot2} must be non-positive: $h_{\text{pot2}}(s_1) \leq 0$ and $h_{\text{pot2}}(s_2) \leq 0$. In contrast, states s_3 and s_4 are clearly unsolvable, which implies that their heuristic values for h_{pot2} must be strictly positive: $h_{\text{pot2}}(s_3) > 0$ and $h_{\text{pot2}}(s_4) > 0$.

We now define the variable partition $\{W, M\}$ as follows:

$$W = \{v_1, v_2\}, \quad M = \{v_3\}.$$

We claim that $\langle [s_4, s_2, s_3, s_1], [W, M]\rangle$ forms a witnessing quartet for h_{pot2} as per Definition 9. We verify the conditions:

- $h_{pot2}(s_4) > h_{pot2}(s_2)$: State s_4 is unsolvable, while s_2 is a goal state.
- $s_4^W = s_2^W$: Both share the same assignments on $v_1 = 1$ and $v_2 = 0$.
- $s_4^M = s_1^M$: Both have $v_3 = 1$.
- $h_{pot2}(s_3) \geq h_{pot2}(s_1)$: State s_3 is unsolvable, while s_1 is the initial state.
- $s_3^W = s_1^W$: Both have $v_1 = 0$ and $v_2 = 1$.
- $s_3^M = s_2^M$: Both have $v_3 = 0$.

All conditions are satisfied, so this is indeed a valid witnessing quartet. By Theorem 1, the existence of such a quartet implies that the potential function h_{pot2} must be of dimension at least 2. This contradicts the assumption that a 1-dimensional PDDA heuristic exists.

Therefore, the PDDA correlation complexity of this task is at least 2, while the DDA correlation complexity is 1. □

4.5 Correlation Complexity Hierarchy

In this section, we analyse the hierarchical and practical relationships among the four principal variants of the DDA heuristic: UDDA, DDA, ∞-DDA, and PDDA.

The hierarchical relationships among the four heuristics are shown in Figure 4.4, where each edge labelled $X \geq Y$ indicates that the correlation complexity of X is always at least that of Y.

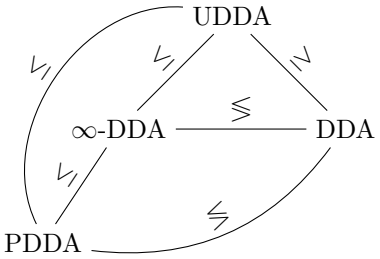


Figure 4.4: Hierarchy of DDA heuristic variants. An edge labelled $X \geq Y$ indicates that the correlation complexity of X is always greater than or equal to that of Y. An edge labelled $X \lesseqgtr Y$ indicates that the correlation complexities of X and Y are incomparable, neither consistently dominates the other.

Edges labelled $X \lesseqgtr Y$ indicate that the correlation complexities of X and Y are incomparable: their relative performance varies depending on the problem instance.

Among all variants, UDDA is the most general. As established in Lemma 1, every UDDA heuristic can be transformed into a DDA, ∞-DDA, or PDDA heuristic. This implies that the correlation complexity of any of these specialized forms is always upper-bounded by that of UDDA.

Through concrete examples, such as the modified Gray code, we demonstrated that DDA, ∞-DDA, and PDDA can, in certain planning tasks, yield strictly lower correlation complexity than UDDA (Lemma 2).

Furthermore, every ∞-DDA heuristic can be reduced to an equivalent PDDA heuristic (Lemma 3), making PDDA a more general representation. Consequently, the correlation complexity of PDDA is always upper-bounded by that of ∞-DDA. Nonetheless, there are problem instances where PDDA achieves strictly lower correlation complexity (Lemma 4).

In contrast, DDA and ∞-DDA are not mutually reducible. Neither formulation can be universally transformed into the other, highlighting a structural divergence between them. Their relative correlation complexities depend on the specific instance: Lemma 5 and 6 show that either heuristic can yield lower complexity than the other, depending on the specific planning scenario. A similar incomparability holds between DDA and PDDA (Lemma 7 and 8).

5
Blocksworld

As outlined in Example 1 of Chapter 2, the objective in the Blocksworld domain is to rearrange a set of blocks $\mathcal{B}$ from an initial configuration into a goal configuration that specifies every variable assignment. Therefore, this goal configuration corresponds to a single, unique goal state G. The domain is modelled using state variables that represent the position of each block (pos_X) and whether the top of each block is clear ($clear_X$), for all $X \in \mathcal{B}$. Both the initial and goal states are *structurally valid*, satisfying the following constraints: no block is positioned on itself, no two distinct blocks are positioned on the same block simultaneously, and a block is clear if and only if no other block is positioned on top of it, with the opposite holding for a block that is not clear. Operators in the domain define actions that move a block either from one block onto another or from a block onto a table.

Seipp et al. [14] introduce a DDA heuristic for the Blocksworld domain and show that its correlation complexity is 2. In this chapter, we extend their work by analysing three additional DDA variants: the UDDA, the ∞-DDA and the PDDA heuristic. In the following subsections, we prove that UDDA heuristics for Blocksworld do not exist, that the PDDA correlation complexity is 2, while the ∞-DDA correlation complexity is 3.

5.1 Non-Existence of UDDA

We now show that no heuristic can satisfy the UDDA property in the Blocksworld domain.

Theorem 2. *There exists no UDDA heuristic for Blocksworld.*

Proof. Consider a Blocksworld planning task Π with two blocks, A and B. Let s be a state in which block A is on block B, and simultaneously, block B is on block A. Furthermore, assume that neither block is clear. This configuration creates a cycle in which no block can be moved. As a result, no action is applicable in s. Therefore, s is a dead-end (it has no successors). Although s is unreachable from the initial state, it is still part of the state space. Since a UDDA heuristic requires that any non-goal state must allow progress towards the goal, the existence of this unsolvable state s contradicts this requirement. Hence, no heuristic function

can satisfy the UDDA property in Blocksworld, proving that a UDDA heuristic does not exist for this domain.

□

5.2 Lower Bound Results

In this section, we establish lower bounds on the correlation complexity of Blocksworld for the PDDA and ∞-DDA heuristic variants.

Lemmas 9. *The PDDA correlation complexity of Blocksworld is at least 2.*

Proof. Assume for contradiction that the PDDA correlation complexity is less than 2. Then there exists a one-dimensional potential heuristic h_{pot} satisfying the PDDA properties.

Consider a Blocksworld task with four blocks. Initially, A is on B, B on C, and C on D. The goal is for A to be on B, B on D, and D on C. To achieve this goal, any valid plan must involve first removing A from B (placing it on the table), and later placing A back onto B. These two actions, placing A from B onto the table and placing A from the table back onto B, are action landmarks. They are unavoidable and occur in every valid plan that solves the task. Since PDDA requires that every finite state has a successor state with a strictly lower heuristic value, all actions that must appear in every valid plan, i.e. action landmarks, must cause a decrease in the heuristic when applied.

Consider the first action: picking up A from B. This changes the state as follows: pos_A transitions from B to T (table), and $clear_B$ from *no* to *yes*. Let s be the state before, and s' the state after the action. Then the PDDA property implies:

$$h(s') - h(s) = w(\{\langle pos_A, T\rangle\}) + w(\{\langle clear_B, yes\rangle\}) - w(\{\langle pos_A, B\rangle\}) - w(\{\langle clear_B, no\rangle\}) < 0$$
$$\Rightarrow w(\{\langle pos_A, T\rangle\}) + w(\{\langle clear_B, yes\rangle\}) < w(\{\langle pos_A, B\rangle\}) + w(\{\langle clear_B, no\rangle\}).$$

Now consider the later landmark action: placing A back on B. This reverses the variable changes: pos_A goes from T to B and $clear_B$ from *yes* to *no*. Applying the same reasoning:

$$h(s') - h(s) = w(\{\langle pos_A, B\rangle\}) + w(\{\langle clear_B, no\rangle\}) - w(\{\langle pos_A, T\rangle\}) - w(\{\langle clear_B, yes\rangle\}) < 0$$
$$\Rightarrow w(\{\langle pos_A, B\rangle\}) + w(\{\langle clear_B, no\rangle\}) < w(\{\langle pos_A, T\rangle\}) + w(\{\langle clear_B, yes\rangle\}).$$

Combining both inequalities leads to a contradiction:

$$\begin{aligned} w(\{\langle pos_A, B\rangle\}) + w(\{\langle clear_B, no\rangle\}) &< w(\{\langle pos_A, T\rangle\}) + w(\{\langle clear_B, yes\rangle\}) \\ &< w(\{\langle pos_A, B\rangle\}) + w(\{\langle clear_B, no\rangle\}). \end{aligned}$$

Hence, no one-dimensional PDDA potential heuristic can exist for this task. The PDDA correlation complexity of Blocksworld is therefore at least 2. □

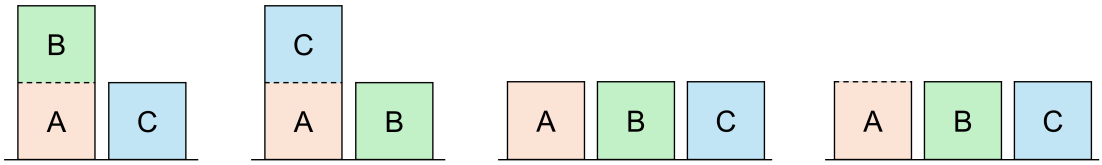
By the structural relationship summarized in Figure 4.4, the ∞-DDA correlation complexity is bounded below by the PDDA correlation complexity. Since the latter is at least 2, we obtain the following immediate consequence:

Corollary 9.1. *The ∞-DDA correlation complexity of Blocksworld is at least 2.*

We now strengthen this result by showing that even 2-dimensional heuristics are insufficient under ∞-DDA constraints. This yields the following lemma:

Lemmas 10. *The ∞-DDA correlation complexity of Blocksworld is at least 3.*

Proof. By Corollary 9.1, the ∞-DDA correlation complexity of Blocksworld is at least 2. Assume for contradiction that it is exactly 2. Then there exists a heuristic h of dimension 2 that satisfies the ∞-DDA properties.



(a) Initial state I (b) Goal state G (c) State landmark s (d) Unsolvable state s'

Figure 5.1: Four example states from a Blocksworld task Π with three blocks. If the top of a block is shown with a dashed outline, it indicates that the block is *not clear*.

Consider a Blocksworld planning task Π with three blocks A, B, and C as depicted in Figure 5.1. The initial state is

$$I = \{\langle pos_A, T\rangle, \langle pos_B, A\rangle, \langle pos_C, T\rangle, \langle clear_A, no\rangle, \langle clear_B, yes\rangle, \langle clear_C, yes\rangle\},$$

and the goal state is

$$G = \{\langle pos_A, T\rangle, \langle pos_B, T\rangle, \langle pos_C, A\rangle, \langle clear_A, no\rangle, \langle clear_B, yes\rangle, \langle clear_C, yes\rangle\}.$$

To achieve the goal, block B must first be moved to the table to clear block A, allowing block C to be placed on top of A. Therefore, the following state s is a state landmark in this task:

$$s = \{\langle pos_A, T\rangle, \langle pos_B, T\rangle, \langle pos_C, T\rangle, \langle clear_A, yes\rangle, \langle clear_B, yes\rangle, \langle clear_C, yes\rangle\}.$$

Now consider the following state:

$$s' = \{\langle pos_A, T\rangle, \langle pos_B, T\rangle, \langle pos_C, T\rangle, \langle clear_A, no\rangle, \langle clear_B, yes\rangle, \langle clear_C, yes\rangle\}.$$

In this state s', all blocks are on the table, but A is not clear. Since placing C on A requires A to be clear, and there is no block on A that could be removed, the goal is unreachable from this state. Therefore, s' must be assigned an infinite heuristic value.

To satisfy the ∞-DDA condition at dimension 2, some singleton atom or pair of atoms present in s' must explain this infinite cost. There are 6 variables in s', giving 6 singleton atoms and $\binom{6}{2} = 15$ pairs, a total of 21 features. As shown in Table 5.1, every singleton or pairwise features present in s' also appears in at least one of the following: the initial state I, the goal state G, or the state s. Since each of these features must therefore have finite heuristic, no 1- or 2-dimensional feature can explain the infinite cost of s'. This contradicts the assumption that a 2-dimensional ∞-DDA heuristic exists. Hence, the ∞-DDA correlation complexity of Blocksworld must be at least 3. □

Feature	**Initial** (I)	**Goal** (G)	**State** s
$\langle pos_A, T\rangle$	✓	✓	✓
$\langle pos_C, T\rangle$	✓	×	✓
$\langle clear_A, no\rangle$	✓	✓	×
$\langle clear_B, yes\rangle$	✓	✓	✓
$\langle clear_C, yes\rangle$	✓	✓	✓
$\langle pos_B, T\rangle$	×	✓	✓
$\{\langle clear_B, yes\rangle,\ \langle clear_C, yes\rangle\}$	✓	✓	✓
$\{\langle clear_B, yes\rangle,\ \langle clear_A, no\rangle\}$	✓	✓	×
$\{\langle clear_C, yes\rangle,\ \langle clear_A, no\rangle\}$	✓	✓	×
$\{\langle clear_B, yes\rangle,\ \langle pos_A, T\rangle\}$	✓	✓	✓
$\{\langle clear_C, yes\rangle,\ \langle pos_A, T\rangle\}$	✓	✓	✓
$\{\langle clear_A, no\rangle,\ \langle pos_A, T\rangle\}$	✓	✓	×
$\{\langle clear_A, no\rangle,\ \langle pos_C, T\rangle\}$	✓	×	×
$\{\langle pos_A, T\rangle,\ \langle pos_C, T\rangle\}$	✓	×	✓
$\{\langle clear_C, yes\rangle,\ \langle pos_C, T\rangle\}$	✓	×	✓
$\{\langle clear_B, yes\rangle,\ \langle pos_C, T\rangle\}$	✓	×	✓
$\{\langle clear_A, no\rangle,\ \langle pos_B, T\rangle\}$	×	✓	×
$\{\langle pos_A, T\rangle,\ \langle pos_B, T\rangle\}$	×	✓	✓
$\{\langle clear_B, yes\rangle,\ \langle pos_B, T\rangle\}$	×	✓	✓
$\{\langle clear_C, yes\rangle,\ \langle pos_B, T\rangle\}$	×	✓	✓
$\{\langle pos_B, T\rangle,\ \langle pos_C, T\rangle\}$	×	×	✓

Table 5.1: The 21 singleton and pairwise features present in state s', along with their presence in the initial state (I), goal state (G), and state s. A checkmark (✓) indicates the feature appears in the corresponding state, a cross (×) indicates it does not.

5.3 Upper Bound Results

For both the ∞-DDA and PDDA heuristics, assigning infinite values to unsolvable states is essential. Let $\mathcal{B}$ denote the set of blocks in a task. We identify two types of problematic features that can make a state s unsolvable:

(i) **Incorrectly marked block as not clear:**
A block X is marked as not clear (i.e., $\langle clear_X, no\rangle \in s$), yet no other block is positioned on top of it. In this case, block X will remain not clear indefinitely, since no operator can change its state. This becomes a problem if the goal requires placing another block on X, which would require it to be clear. Formally:

$$\langle clear_X, no\rangle \in s \text{ and for all } Y \in \mathcal{B}:\ s(pos_Y) \neq X$$

(ii) **Cyclic dependency of blocks:**
A cycle exists in which each block is stacked on another block in the cycle, and none of the blocks are on the table. Additionally, all blocks in the cycle are not clear. This configuration is unsolvable because no block in the cycle can be moved. Formally:

$$\{\langle pos_{Y_1}, Y_2\rangle, \langle pos_{Y_2}, Y_3\rangle, \ldots, \langle pos_{Y_n}, Y_1\rangle\} \cup \{\langle clear_{Y_i}, no\rangle \mid 1 \leq i \leq n\} \subseteq s$$

where $Y_1, Y_2, \ldots, Y_n \in \mathcal{B}$ are distinct blocks and none of them is positioned on the table.

Identifying unsolvable states and assigning them a heuristic value of infinity is crucial, as the ∞-DDA and PDDA properties require that every state with a finite heuristic value has at least one successor with a strictly lower value. If an unsolvable state were given a finite value, it could begin a descending chain. However, since no state in this chain can reach a goal, the sequence must eventually reach a state with no successor or only successors with higher heuristic values, violating the ∞-DDA and PDDA properties.

Before proceeding, we introduce some terminology and notation [14] that will be used in both the PDDA and ∞-DDA constructions. Let $\mathcal{B}$ be the set of blocks in a given Blocksworld task. For each block $A \in \mathcal{B}$, let G_A denote its *goal position*, i.e. the position of A in the goal state (either another block or the table). A block A is said to be *correctly placed* in a state s if $s(pos_A) = G_A$, and *misplaced* otherwise. Furthermore, we say that block A *controls* block B if B appears below A in the goal tower structure. To formalize this notion of vertical ordering, we assign a *level* to each block in the goal configuration: in every goal tower, the block at the top is assigned level 1, the block immediately beneath it level 2, and so on. A block is said to be *done* in a state if it is correctly placed and all blocks below it in the goal stack are also correctly placed.

We now construct PDDA and ∞-DDA heuristics for Blocksworld, establishing that their correlation complexities are bounded above by 2 and 3, respectively. Both heuristics are designed to encourage a structured, two-phase solution strategy: first, move all blocks that are not yet done onto the table. Then, build the goal towers from the bottom up.

Lemmas 11. *The PDDA correlation complexity of Blocksworld is at most 2.*

Proof. We prove the lemma by constructing a heuristic h_{pot} of dimension 2 that satisfies the PDDA property. Let Π be a Blocksworld task, and let $\mathcal{B}$ denote its set of all blocks in Π. To ensure that planning avoids dead-end states, we define the potential heuristic h_{pot2} over the following weight function w_2:

(a) We immediately assign a weight large enough that the heuristic function h_{pot} evaluates to infinity whenever the following invalid or undesirable configurations are detected:

   (i) A block cannot be declared clear when another block is on top of it:
   $w_2(\{\langle clear_A, yes\rangle, \langle pos_B, A\rangle\}) = |\mathcal{B}| + 1$
   for all $A, B \in \mathcal{B}$.

   (ii) Prevent structurally invalid or unintended stacking configurations by penalizing certain pairs of positional relationships between blocks:
   Specifically, if two block-to-block relations (e.g., block A on block B, and block C on block D) do not jointly occur in either the initial state or the goal state, they are considered invalid and heavily penalized. This general formulation captures specific undesirable cases, such as a block being on itself or multiple blocks stacked on the same block.
   $w_2(\{\langle pos_A, B\rangle, \langle pos_C, D\rangle\}) = |\mathcal{B}| + 1$
   for all $A, B, C, D \in \mathcal{B}$ with $A \neq C$, $\{\langle pos_A, B\rangle, \langle pos_C, D\rangle\} \not\subset I$ and
   $\{\langle pos_A, B\rangle, \langle pos_C, D\rangle\} \not\subset g$.

(b) We ensure consistency between clear annotations and actual block placements. Specifically, we count the number of blocks marked as not clear and compare it to the number of blocks that are actually positioned on top of other blocks. If more blocks are marked as not clear than are truly supporting another block, this indicates an inconsistency, i.e. at least one block is incorrectly marked as not clear despite having nothing on top of it.

(i) Block marked as not clear:
$w_2(\{\langle clear_A, no\rangle\}) = 1$
for all $A \in \mathcal{B}$.

(ii) Block positioned on another block:
$w_2(\{\langle pos_B, A\rangle\}) = -1$
for all $A, B \in \mathcal{B}$ with $A \neq B$.

We show that $|\mathcal{B}|+1$ is large enough to guarantee $h_{\text{pot2}}(s) > 0$ whenever any feature from item (a)(i) or (a)(ii) is active. Such a feature contributes a penalty of $|\mathcal{B}| + 1$ to $h_{\text{pot2}}(s)$. To offset this, the state would need a total negative contribution of at least $-(|\mathcal{B}| + 1)$ from item (b)(ii) features. Since there are only $|\mathcal{B}|$ blocks, at most $|\mathcal{B}|$ such features can be active, limiting the maximum negative offset to $-|\mathcal{B}|$, which is insufficient to cancel the penalty. Hence, $h_{\text{pot2}}(s)$ remains strictly positive whenever a feature from item (a) is active. Consequently, any state s with $h_{\text{pot2}}(s) \leq 0$ must satisfy:

- Every clear block is truly clear: If a block A is marked as clear in s, i.e. $\langle clear_A, yes\rangle \in s$, then no block B may be positioned on top of A, that is, $\langle posB, A\rangle \notin s$ for any B. Otherwise, a conflicting feature would be activated, contributing a penalty of $|\mathcal{B}| + 1$ to $h$pot2(s), causing it to exceed zero.

- Block configurations are structurally valid: Any feature $F = \{\langle pos_A, B\rangle, \langle pos_C, D\rangle\}$ with $A \neq C$ cannot be present in s unless F is a subset of either the initial state I or the goal state G. Since both I and G are assumed to be structurally valid, such features in s imply that no block is on itself and no two distinct blocks are placed on the same block at the same time. Thus, s must also respect these constraints.

- Every not clear block is truly not clear: For the heuristic value to be negative, the number of facts $\langle pos_B, A\rangle \in s$ (blocks positioned on other blocks) must be at least as large as the number of facts $\langle clear_A, no\rangle \in s$ (blocks marked as not clear). The previous points guarantee that no block is placed on itself and that no two distinct blocks share the same support, so each block can have at most one block on top. Combined with the fact that every block marked as clear is truly clear, this means these two counts must in fact be equal.

Therefore, any state s with $h_{\text{pot2}}(s) \leq 0$ is structurally valid.
While these penalty terms exclude undesirable states, they provide no guidance within the space of valid, finite configurations. To handle these remaining states, we adapt the weight function that Seipp et al. [14] define for the Blocksworld domain. Accordingly, h_{pot1} is defined using the following weight function w_1:

(i) $w_1(\{\langle pos_A, X\rangle\}) = 2$
for all $A \in \mathcal{B},\ X \in \mathcal{B} \setminus \{A\},\ X \neq G_A$.

(ii) $w_1(\{\langle pos_A, T\rangle\}) = -1$
for all $A \in \mathcal{B}$ with $G_A = T$.

(iii) $w_1(\{\langle pos_A, T\rangle\}) = 1$
for all $A \in \mathcal{B}$ with $G_A \neq T$.

(iv) $w_1(\{\langle pos_A, G_A\rangle, \langle pos_B, X\rangle\}) = 2^{level(A)}$
for all $A, B \in \mathcal{B}$ where B is controlled by A and all $X \in dom(pos_B)$ with $X \neq G_B$.

Now, we will verify that this heuristic function satisfies the PDDA properties, ensuring the PDDA correlation complexity is at most 2.

The initial state has a finite value, i.e. $I \in S_{\mathbf{fin}}$: First, consider the high-penalty terms defined in part (a) of w_2. The penalty in (a)(i) is triggered when a block is simultaneously marked as clear and has another block positioned on top of it. However, by definition of the initial state in Blocksworld, the clear predicates are consistent with the block positions: a block is marked clear if and only if no other block is on top of it. Therefore, such conflicting feature pairs $\{\langle clear_A, yes\rangle, \langle pos_B, A\rangle\}$ do not appear in I, and the corresponding penalty is avoided.

For (a)(ii), the heuristic assigns a penalty to any pair of stacking relationships that does not jointly occur in either the initial state I or the goal state g. Since we are evaluating h_{pot2} in the initial state itself, any such pair that appears in I is, by definition, part of the initial state and thus not penalized.

Now consider part (b) of the weight function, which uses positive weights for features $\langle clear_A, no\rangle$ and negative weights for features $\langle pos_B, A\rangle$. In the initial state, every block that is marked as not clear indeed has another block on top of it, and vice versa. This means the total count of "not clear" annotations exactly matches the number of actual stacking relationships. Therefore, the positive and negative contributions cancel out, yielding a net weight of zero.

Combining these observations, we conclude that no penalties are applied in $h_{\text{pot2}}(I)$, and the overall sum is zero.

For all non-goal states with a finite heuristic value there exists a successor with a lower heuristic value, i.e. $\forall s \in (S_{\mathbf{fin}} \setminus G)\exists s' \in \mathbf{succ}(s) : h(s') < h(s)$: To prove that, we make a case distinction over an arbitrary state $s \in S_{\text{fin}} \setminus G$:

As shown above, the finiteness of s implies that all block arrangements in s are structurally valid. Additionally, all pairs of stacking relations that appear in s must be consistent with either the initial state or the goal state, as enforced by the penalty described in part (a)(ii) of the weight function w_2. We distinguish two cases:

Case 1: There exists at least one block that is positioned on a block as in the initial state I, but not as in the goal state g. Therefore, s has a tower of at least two blocks such that the top block A is not done. Seipp et al. [14] show that placing block A onto the table decreases the heuristic value. Hence, we only have to prove that the heuristic value stays finite:

After moving A onto the table, the resulting state s' avoids any infinite penalty features. In particular, if in s no block incorrectly marked as clear has another block on top of it, this property is preserved in s' since placing a block onto the table does not introduce such a violation. Similarly, all pairs of stacked blocks remain consistent with either the initial or the goal state in s', as moving A removes a block-on-block relation without introducing any illegal ones. Moreover, since the number of blocks marked as not clear matches exactly the number of blocks stacked on top of others, moving A onto the table clears the block below A and decreases the number of block-on-block relations by one. Thus, the equality between these quantities is preserved, ensuring that no infinite penalties are introduced. Hence, the heuristic value in s' remains finite.

Case 2: No block remains in its initial position (if that position differs from the goal), all remaining stacks are partially or fully aligned with the goal state, meaning that all not-done blocks are placed on the table. Seipp et al. [14] show that moving block A onto G_A, where G_A is done, decreases the heuristic value. , we only have to proof that the heuristic value stays finite:

The reasoning is analogous to Case 1. After moving block A onto its goal position G_A, which is done, the resulting state s' maintains consistency with the goal stack relations: since s contained no block-on-block relations from the initial state I and the move only adds a block-on-block relation present in the goal g, s' contains only goal state relations. Additionally, the balance between blocks marked as not clear and the actual block-on-block relations is preserved. No block incorrectly marked as clear gains a block on top, ensuring that no infinite penalties are introduced. Therefore, the heuristic value in s' remains finite.

In both cases, we can apply a valid action that strictly reduces the value of the heuristic. Therefore, for all non-goal states $s \in S_{\text{fin}} \setminus G$, there exists a successor $s' \in \text{succ}(s)$ such that $h(s') < h(s)$, as required.

□

We now turn to the ∞-DDA variant, using a similar construction to show that the ∞-DDA correlation complexity for Blocksworld is upper-bounded by 3.

Lemmas 12. *The ∞-DDA correlation complexity of Blocksworld is at most 3.*

Proof. We prove the lemma by constructing a heuristic h of dimension 3 that satisfies the ∞-DDA property. Let Π be a Blocksworld task, and let $\mathcal{B}$ denote its set of all blocks in Π.
Our heuristic h is defined by adapting weights from the previously defined heuristic h_{pot2}. The key difference is that features which previously received a large but finite penalty in h_{pot2} are now assigned infinite weights to strictly enforce the ∞-DDA property. Specifically, these infinite penalties directly exclude invalid configurations such as blocks incorrectly marked clear with

another block on top, and cyclic or structurally invalid stacking, as defined before.

A notable additional difference addresses blocks marked as not clear when no block is on top of them. We solve this problem by assigning infinite weights to all blocks marked not clear but none of the expected blocks to be on them (according to the initial state I and goal G) actually occupy that position. To capture this, we introduce the following infinite penalties:

(i) $w(\{\langle clear_A, no\rangle, \langle pos_B, C\rangle, \langle pos_D, E\rangle\}) = \infty$
for all $A, B, C, D, E \in \mathcal{B}$, with $C \neq A$, and $E \neq A$, where $\langle pos_B, A\rangle \in I$, $\langle pos_D, A\rangle \in G$.

(ii) $w(\{\langle clear_A, no\rangle, \langle pos_B, C\rangle\}) = \infty$
for all $A, B, C \in \mathcal{B}$, with $C \neq A$, where $\langle pos_B, A\rangle \in I$ and $\langle clear_A, yes\rangle \in G$.

(iii) $w(\{\langle clear_A, no\rangle, \langle pos_B, C\rangle\}) = \infty$
for all $A, B, C \in \mathcal{B}$, with $C \neq A$, where $\langle pos_B, A\rangle \in G$ and $\langle clear_A, yes\rangle \in I$.

(iv) $w(\{\langle clear_A, no\}) = \infty$
for all $A \in \mathcal{B}$, with $\langle clear_A, yes\rangle \in I$, $\langle clear_A, yes\rangle \in G$.

For all remaining finite states, the heuristic uses the weight function proposed by Seipp et al. [14]. The argument that h satisfies the ∞-DDA property follows exactly as in the PDDA proof, with infinite penalties ensuring stricter exclusion of invalid or undesired states while preserving the existence of improving successors for all finite, non-goal states.
Thus, the ∞-DDA correlation complexity of Blocksworld is at most 3. □

5.4 Exact Correlation Complexities

With the lower and upper bounds established in the preceding lemmas and corollaries, we now arrive at the main conclusion of this chapter: the ∞-DDA correlation complexity of Blocksworld is 3, while the PDDA correlation complexity is 2.

Theorem 3. *The ∞-DDA correlation complexity of Blocksworld is 3.*

Proof. Lemma 10 and 12 establish that the ∞-DDA correlation complexity is bounded below and above by 3, respectively. Hence, we conclude that the correlation complexity is 3. □

Theorem 4. *The PDDA correlation complexity of Blocksworld is 2.*

Proof. By combining Lemma 9 and 11, which provide a matching lower and upper bound of 2, we conclude that the exact correlation complexity is 2. □

6
Spanner

In the Spanner domain (IPC 2014), an agent must navigate a sequence of $m + 1$ locations, loc_0 through loc_m, to reach a gate at loc_m. Along this path, the agent encounters N single-use spanners placed at various locations before reaching the gate. At the gate itself, there are n loose nuts, each of which requires a spanner to be tightened. We assume $N \geq n$. Otherwise, the task would be unsolvable, making it impossible to define an UDDA, ∞-DDA or PDDA heuristic.

The agent must collect enough spanners before reaching the gate. Importantly, movement is constrained: the agent can only move forward, not backward. A visual representation of a Spanner task is shown in Figure 6.1.

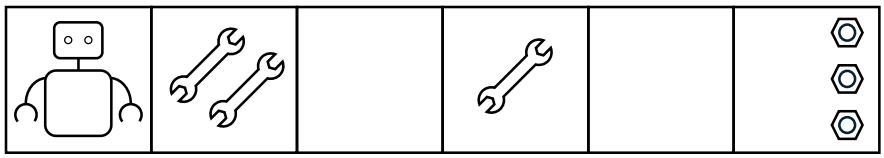


Figure 6.1: Example of the initial state of a Spanner task with six locations, three loose nuts, and three spanners scattered among the locations. The goal is to collect all spanners and tighten the three nuts. The agent can only move from left to right.

The domain is formally defined using variables that describe:

- *agent*: The agent's current location, with domain $\text{dom}(agent) = \{loc_i\}_{i=0}^{m}$.
- $spanner_j$: The location of spanner j, with domain $\text{dom}(spanner_j) = \{loc_i\}_{i=0}^{m} \cup \{agent\}$.
- $usable_j$: The usability status of spanner j, with domain $\text{dom}(usable_j) = \{yes, no\}$.
- nut_j: The state of nut j, with domain $\text{dom}(nut_j) = \{tightened, loose\}$.

Initially, the agent starts at loc_0, all spanners are positioned at some location (not carried by the agent), all spanners are assumed to be usable, and all nuts are loose. The available operators are moving forward, picking up a spanner, and tightening a nut.

Seipp et al. [14] introduce a DDA heuristic for the Spanner domain and demonstrated that its correlation complexity is 2. In this chapter, we extend their work by analysing three additional variants: the UDDA, the ∞-DDA, and the PDDA heuristics. We show that there is no UDDA heuristic for the Spanner domain (Theorem 5). Furthermore, we prove that the ∞-DDA and the PDDA correlation complexity remains 2 (Theorems 6 and 7).

6.1 Non-Existence of UDDA

In this section, we show that no heuristic function can satisfy the UDDA property in the Spanner domain. The key insight lies in the existence of dead-end states, states from which no sequence of actions leads to a goal. These states violate the core UDDA requirement that every non-goal state must have a successor with strictly lower heuristic value. We formalize this argument in the following theorem.

Theorem 5. *There exists no UDDA heuristic for Spanner.*

Proof. We can demonstrate this by presenting a dead-end state where no valid action can lead to a goal state. Consider a Spanner planning task Π with two locations, one spanner and one nut. Suppose the agent is at loc_2, while the spanner is at loc_1. In this state s, the agent is unable to move forward, pick up the spanner, or tighten the nut (since they are not carrying the spanner). As a result, no applicable operator exists, meaning s has no successors. Consequently, no heuristic function can satisfy the UDDA property in the Spanner domain, proving that a UDDA heuristic does not exist for this domain.

□

6.2 Lower Bound Results

In the following, we establish a lower bound on the PDDA correlation complexity and use this result to derive a corresponding lower bound for the ∞-DDA correlation complexity.

Lemmas 13. *The PDDA correlation complexity of Spanner is at least 2.*

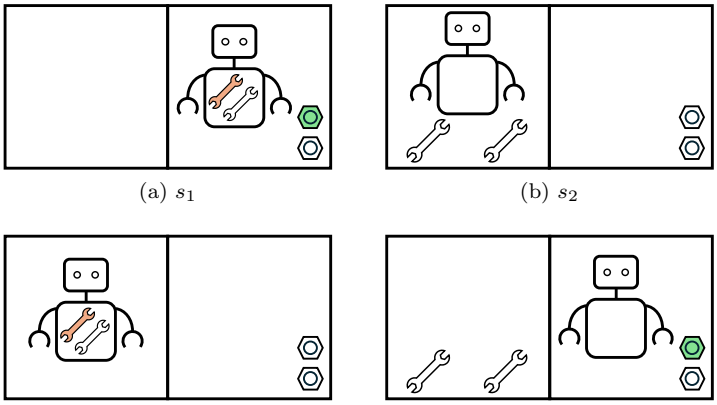
Proof. We prove by contradiction that the PDDA correlation complexity of Spanner is at least 2 by using a witnessing quartet. Assume for contradiction that there exists a 1-dimensional potential function h_{pot} (defined as in Definition 6) satisfying the PDDA property.

Consider a Spanner planning task Π with 2 locations, 2 spanners, and 2 nuts. The initial state I is given by:

$$I = \{\langle agent, loc_0\rangle, \langle spanner_1, loc_0\rangle, \langle spanner_2, loc_0\rangle, \langle usable_1, yes\rangle, \langle usable_2, yes\rangle, \langle nut_1, loose\rangle, \langle nut_2, loose\rangle\}.$$

Since any plan that reaches the goal must pass through a state where one spanner is unusable and one nut is already tightened, at least one such state s_1 must be assigned a finite heuristic value. Otherwise, the goal would be unreachable, contradicting the assumption that the task is solvable.

Assume w.l.o.g. that spanner 1 was used to tighten nut 1 in s_1. This assumption is justified by the symmetry of the problem, as relabelling the spanners and nuts would yield an equivalent case.



(a) s_1 (b) s_2

(c) s_3 (d) s_4

Figure 6.2: Four example states from a Spanner task with two locations, two nuts, and two spanners. Spanners that are no longer usable are shown in red, and tightened nuts are displayed in green.

Now, consider the following four states represented in Figure 6.2:

$$s_1 : \{\langle agent, loc_1\rangle, \langle spanner_1, agent\rangle, \langle spanner_2, agent\rangle, \langle usable_1, no\rangle, \langle usable_2, yes\rangle, \langle nut_1, tightened\rangle, \langle nut_2, loose\rangle\}$$

$$s_2 : \{\langle agent, loc_0\rangle, \langle spanner_1, loc_0\rangle, \langle spanner_2, loc_0\rangle, \langle usable_1, yes\rangle, \langle usable_2, yes\rangle, \langle nut_1, loose\rangle, \langle nut_2, loose\rangle\}$$

$$s_3 : \{\langle agent, loc_0\rangle, \langle spanner_1, agent\rangle, \langle spanner_2, agent\rangle, \langle usable_1, no\rangle, \langle usable_2, yes\rangle, \langle nut_1, loose\rangle, \langle nut_2, loose\rangle\}$$

$$s_4 : \{\langle agent, loc_1\rangle, \langle spanner_1, loc_0\rangle, \langle spanner_2, loc_0\rangle, \langle usable_1, yes\rangle, \langle usable_2, yes\rangle, \langle nut_1, tightened\rangle, \langle nut_2, loose\rangle\}$$

From the previous argument, we know that s_1 must be assigned a finite heuristic value. Since s_2 is the initial state, it too must have a finite heuristic value. State s_3 is unsolvable because only one spanner is usable, while both nuts are still loose. As it is impossible to tighten both nuts in this state, the goal cannot be reached from s_3, so it must be assigned an infinite heuristic value. Similarly, s_4 is also unsolvable. Although the agent is at the final location, it lacks a spanner to tighten the nuts, making it impossible to reach the goal. Therefore, s_4 must also be assigned an infinite heuristic value.
This results in the following constraints on h_{pot2}:

$$h_{pot2}(s_1) \leq 0, \quad h_{pot2}(s_2) \leq 0, \quad h_{pot2}(s_3) > 0, \quad h_{pot2}(s_4) > 0.$$

We now define the variable partition $\{W, M\}$ as follows:

$$W = \{usable_1, usable_2, spanner_1, spanner_2\}, \quad M = \{agent, nut_1, nut_2\}.$$

We claim that $\langle [s_4, s_2, s_3, s_1], [W, M] \rangle$ forms a witnessing quartet for h_{pot2} as per Definition 9. We verify the conditions:

- $h_{pot2}(s_4) > h_{pot2}(s_2)$: State s_4 is unsolvable, while s_2 is solvable.
- $s_4^W = s_2^W$: In both states, the spanners are located at loc_0 and are usable.
- $s_4^M = s_1^M$: The agent is at loc_1; nut_1 is tightened and nut_2 is loose in both states.
- $h_{pot2}(s_3) \geq h_{pot2}(s_1)$: State s_3 is unsolvable, while s_1 is solvable.
- $s_3^W = s_1^W$: In both states, the agent holds both spanners, $spanner_1$ is not usable and $spanner_2$ is usable.
- $s_3^M = s_2^M$: The agent is at loc_0, and both nuts are loose in both states.

Thus, all conditions for a witnessing quartet are satisfied. By Theorem 1, the existence of such a witnessing quartet implies that $\dim(h_{pot2}) \geq 2$. This contradicts our initial assumption that the PDDA correlation complexity of Spanner is less than 2. Therefore, we conclude that the PDDA correlation complexity of Spanner is at least 2.

□

We have established a lower bound for the PDDA correlation complexity. Since every ∞-DDA heuristic can be transformed into a PDDA heuristic (Lemma 3), the same bound holds for ∞-DDA. Thus, we arrive at the following corollary.

Corollary 13.1. *The ∞-DDA correlation complexity of Spanner is at least 2.*

6.3 Upper Bound Results

Next, we derive an upper bound on the ∞-DDA correlation complexity and use it to establish a corresponding bound for the PDDA correlation complexity.

Lemmas 14. *The ∞-DDA correlation complexity of Spanner is at most 2.*

Proof. We prove the lemma by constructing a heuristic h of dimension 2 that satisfies the conditions of an ∞-DDA heuristic, ensuring that the ∞-DDA correlation complexity of Spanner is at most 2. Let Π be a Spanner task with m locations, N spanners, and n nuts.
The heuristic h forces the agent to pick up every spanner and to use each spanner for a specific nut. The heuristic function is defined by the following weight function:

$$
\begin{aligned}
&w(\{\langle agent, loc_i \rangle\}) = m - i && \text{for all } i \in \{0, \ldots, m\} \\
&w(\{\langle spanner_j, loc_i \rangle\}) = 1 && \text{for all } i \in \{0, \ldots, m\},\ j \in \{1, \ldots, N\} \\
&w(\{\langle nut_l, loose \rangle\}) = 1 && \text{for all } l \in \{1, \ldots, n\} \\
&w(\{\langle nut_l, loose \rangle, \langle usable_l, no \rangle\}) = \infty && \text{for all } l \in \{1, \ldots, n\} \\
&w(\{\langle agent, loc_i \rangle, \langle spanner_j, loc_k \rangle\}) = \infty && \text{for all } i \in \{0, \ldots, m\},\ j \in \{1, \ldots, N\}, \\
& && \qquad k \in \{1, \ldots, i-1\}
\end{aligned}
$$

Now, we will verify that this heuristic function satisfies the ∞-DDA properties, ensuring the ∞-DDA correlation complexity is at most 2.

The initial state has a finite value, i.e. $I \in S_{\mathbf{fin}}$:
Let I be the initial state. We show that $h(I) \in \mathbb{R}$, meaning it has a finite heuristic value: Since the agent always starts at loc_0, there is no loc_k with $k < 0$, ensuring that for all $i = 0,\ j \in \{1, \ldots, n\},\ k \in \{0, \ldots, i-1\}$: $\langle spanner_j, loc_k \rangle \notin I$. Additionally because initially all spanners are usable, it holds that for all $j \in \{1, \ldots, n\} : \langle usable_j, no \rangle \notin I$. From this, we can conclude that the initial state is in S_{fin}.

For all non-goal states with a finite heuristic value there exists a successor with a lower heuristic value, i.e. $\forall s \in (S_{\mathbf{fin}} \setminus G) \exists s' \in \mathbf{succ}(s) : h(s') < h(s)$:
To prove that, we make a case distinction over s:

1) The agent is at location loc_i where $i \in \{0, \ldots, m-1\}$:

   a) Spanner j is lying at loc_i (usable or non-usable) where $j \in \{1, \ldots, N\}$:
   Picking up spanner j only affects the variable assignment of $spanner_j$, changing it from loc_i to *agent*. This updates the state by removing the fact $\langle spanner_j, loc_i \rangle$ and adding $\langle spanner_j, agent \rangle$.

   Since $\langle spanner_j, loc_i \rangle \notin s$ after the pickup, its associated weight is removed from the heuristic sum. Whether the weight $w(\langle agent, loc_i \rangle, \langle spanner_j, loc_k \rangle)$ contributes to the heuristic depends on $k \in \{0, \ldots, i-1\}$. Because k is always less than i, it refers only to spanners at earlier locations and is therefore unaffected by changes to a spanner at loc_i.

   Consequently, picking up a spanner reduces the heuristic value by the weight associated with the fact $\langle spanner_j, loc_i \rangle$, which is 1:

   $$h(s') = h(s) - 1 < h(s).$$

   b) No spanner is present at loc_i:
   Moving forward from loc_i to loc_{i+1} only affects the agent's position, changing the variable assignment of *agent* from loc_i to loc_{i+1}. This transition updates the state by removing the fact $\langle agent, loc_i \rangle$ and adding the fact $\langle agent, loc_{i+1} \rangle$.
   This change affects two types of weights:

   First, the weight of $w(\{\langle agent, loc_i \rangle\}) = m - i$ disappears as the fact is removed, while $w(\{\langle agent, loc_{i+1} \rangle\}) = m - (i+1)$ is added as the new fact is introduced.

   Second, this update could potentially influence weights that depend on both the agent's position and the presence of a spanner at an earlier location. The relevant weights in this case are $w(\{\langle agent, loc_i \rangle, \langle spanner_j, loc_{k_i} \rangle\})$ and $w(\{\langle agent, loc_{i+1} \rangle, \langle spanner_j, loc_{k_{i+1}} \rangle\})$, where $j \in \{1, \ldots, N\}$ and $k_x \in \{0, \ldots, x-1\}$.

Since the current state s is in $(S_{\text{fin}} \setminus G)$, we know that no spanner is present at any location loc_{k_i} where $k_i < i$. Furthermore, by our case distinction, there is no spanner at loc_i either. This allows us to extend the statement further: since there is no spanner at loc_i or any earlier location, it follows that no spanner can be present at any $loc_{k_{i+1}}$ with $k_{i+1} < i + 1$. Therefore, the pair $\{\langle agent, loc_{i+1}\rangle, \langle spanner_j, loc_{k_{i+1}}\rangle\}$ is not a subset of s. As a result, no infinite weight is added to the sum, and the heuristic value remains finite.

This results in a change in the heuristic value by $-(m-i)+(m-(i+1)) = -1$:

$$h(s') = h(s) - 1 < h(s).$$

2) The agent is at location loc_m:

   a) Spanner j is lying at loc_m (usable or non-usable) where $j \in \{1, \ldots, N\}$:
   Picking it updates the state the same way as in Item 1a, therefore decreasing the heuristic value by 1:
   $$h(s') = h(s) - 1 < h(s).$$

   b) No spanner is present at loc_m:
   Since the state s lies in $(S_{\text{fin}} \setminus G)$, at least one nut remains loose, otherwise s would be a goal state. Moreover, because $h(s) < \infty$, the following pairs cannot be subsets of s:

   $$\{\langle nut_j, loose\rangle, \langle usable_j, no\rangle\} \text{ and } \{\langle agent, loc_m\rangle, \langle spanner_j, loc_k\rangle\} \text{for } k < m.$$

   The exclusion of the first pair ensures that every loose nut has its corresponding usable spanner available. Consequently, the agent is always able to tighten a nut when necessary. The exclusion of the second pair ensures that any spanner located at an earlier position must already have been picked up by the agent. This ensures that all necessary tools are carried forward and available for use when needed.

   When the agent tightens a nut using its corresponding spanner, two variable assignments are updated. First, nut_l transitions from *loose* to *tightened*. Second, $usable_l$ changes from *yes* to *no*.

   The only variable assignment directly affected by this is $\langle nut_l, loose\rangle$, which lets the corresponding weight get removed from the heuristic sum. As we established earlier, the feature $\{\langle nut_l, loose\rangle, \langle usable_l, no\rangle\}$ is absent in state s. Since the tightening operator removes the fact $\langle nut_l, loose\rangle$, this feature remains absent in the successor state s', meaning that the corresponding weight does not contribute to the heuristic sum.
   Consequently, tightening a nut decreasing the heuristic value by 1:
   $$h(s') = h(s) - 1 < h(s).$$

In each case, the heuristic value decreases, meaning that for every non-goal state s, there exists a successor state s' such that $h(s') < h(s)$.

Since we have shown that for every state s in S_{fin}, there exists a successor state where the heuristic decreases, we conclude that the ∞-DDA correlation complexity of Spanner is at most 2. □

We have established that the ∞-DDA correlation complexity is at most 2. Since every ∞-DDA heuristic can be transformed into a PDDA heuristic (Lemma 3), the same upper bound applies to the PDDA correlation complexity. This leads to the following corollary.

Corollary 14.1. *The PDDA correlation complexity of Spanner is at most 2.*

6.4 Exact Correlation Complexities

Building on the lower and upper bounds established in the previous lemmas and corollaries, we can now draw the key conclusion of this chapter: the ∞-DDA and PDDA correlation complexity of Spanner is 2.

Theorem 6. *The ∞-DDA correlation complexity of Spanner is 2.*

Proof. From Corollary 13.1, we know that the ∞-DDA correlation complexity of Spanner is at least 2. Furthermore, Lemma 14 establishes that the ∞-DDA correlation complexity of Spanner is at most 2. Together, these results imply that the ∞-DDA correlation complexity of Spanner is exactly 2. □

Theorem 7. *The PDDA correlation complexity of Spanner is 2.*

Proof. From Lemma 13, we know that the PDDA correlation complexity of Spanner is lower bounded by 2. Furthermore, Corollary 14.1 establishes that the PDDA correlation complexity is at most 2. Together, these results imply that the PDDA correlation complexity of Spanner is exactly 2. □

7
Logistics

The *Logistics* domain [9] is a well-established benchmark in classical planning, modelling the problem of transporting packages across a network of cities using trucks and airplanes. Each package has a specified origin and destination, and the planner must devise a sequence of actions to deliver all packages to their destinations using available transport resources.

To define the domain, we introduce some important sets and functions:
Let L be a finite set of *locations*, C a finite set of *cities*, P a finite set of *packages*, T a finite set of *trucks*, and A a finite set of *airplanes*. The function $city : L \to C$ assigns each location a city. The function $airport : C \to L$ designates exactly one location in each city as its airport. The function $dest : P \to L$ specifies the goal location for each package.

The domain is formally defined using the following variables:

- $package \in P$: location of a package with $\text{dom}(package) = L \cup T \cup A$
- $airplane \in A$: location of an airplane with $\text{dom}(airplane) = \{airport(c) \mid c \in C\}$
- $truck \in T$: location of a truck with $\text{dom}(truck) = \{package \in P \mid city(l) = city(I[truck])\}$

Each package is initially located at a specific location, and the goal is for it to arrive at a designated location, potentially in another city.

Trucks can move between any two locations within the same city, while airplanes can fly between airports in different cities. Packages are loaded into or unloaded from trucks when both are at the same location, and similarly, transferred to or from airplanes only at airports. Intercity delivery requires a package to be transported by truck to a local airport, flown to the destination city's airport, and then delivered by truck to its final destination. Using these actions, a planner must coordinate both local and intercity transport to deliver all packages to their goals.

In this chapter, we show that for Logistics domain, the DDA, the UDDA, the ∞-DDA and the PDDA correlation complexity is 2.

7.1 Lower Bound Results

In the following, we establish a lower bound on the DDA and PDDA correlation complexity and use the second result to derive a corresponding lower bound for the UDDA and the ∞-DDA correlation complexity.

Lemmas 15. *The DDA correlation complexity of Logistics is at least 2.*

Proof. Consider the Logistics planning task Π described in Figure 7.1. To achieve the goal, the truck must first drive from loc_2 to loc_1, load the package, and then return to loc_2 to unload it.

The drive actions from loc_2 to loc_1 and from loc_2 to loc_1 are both necessary and form a pair of inverse operators. The claim follows by Theorem 5 of Seipp et al. [14], which states that if a planning task has two critical operators that are inverses of each other, then its DDA correlation complexity is at least 2. □

The proof of Lemma 15 shows that the DDA correlation complexity of Logistics is at least 2 by identifying critical inverse operators that must be executed sequentially. However, to extend this analysis to the PDDA correlation complexity, we need a more structured understanding of how state transitions unfold during the plan execution.

This is where the concept of landmarks, as introduced in Section 2.4, becomes crucial. State landmarks provide a structured framework for analysing dependencies and ordering constraints among key intermediate states, enabling a principled discussion of the minimum correlation complexity required for PDDA.

Lemmas 16. *The PDDA correlation complexity of Logistics is at least 2.*

Proof. We assume for contradiction that the PDDA correlation complexity of Logistics is less than 2. This would imply the existence of a potential heuristic function h_{pot} of dimension 1 that satisfies the PDDA properties.

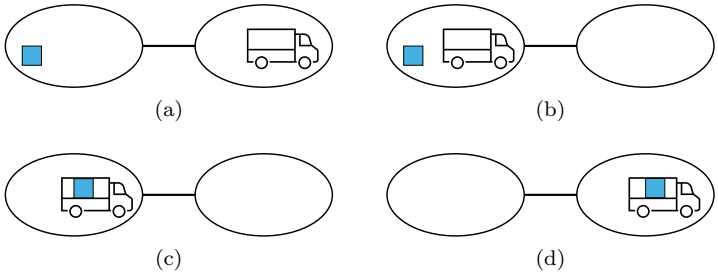


Figure 7.1: Four example states from a logistics task Π with two locations, loc_1 and loc_2, within a single city, one truck, and one package. The initial state I is given by $I = \{\langle package, loc_1\rangle, \langle truck, loc_2\rangle\}$. The goal is to deliver the package from loc_1 to loc_2.

Consider the Logistics planning task Π described in Figure 7.1.

We show that every valid plan must pass through the four distinct states illustrated in Figure 7.1 during the process of completing the task:

(a) The initial state, where the package is at loc_1 and the truck is at loc_2.

(b) The truck has moved to loc_1, but the package is still on the ground.

(c) The package has been loaded into the truck, but the truck is still at loc_1.

(d) The truck has reached loc_2, but the package is still inside the truck.

To reach the goal from state c, we have to go through state d, since the truck must arrive at loc_2 carrying the package. This makes d a landmark of c. Similarly, to reach the goal from b, we must go through c, because the package must be loaded into the truck before it can be moved. Thus, c is a landmark of b. To reach the goal from a, we must go through b, since the truck needs to move to loc_1 to pick up the package. Therefore, b is a landmark of a.

By transitivity, d is a landmark of c, which is a landmark of b, which is a landmark of a, so b, c, and d are all landmarks of a. Since a is the initial state, it follows that b, c and d must appear in every valid plan. We can now further analyse the order in which these landmarks must be achieved. Because each landmark depends on the previous one, any valid plan must visit these states in a specific sequence. This implies a strict ordering: $last(a) < last(b) < last(c) < last(d)$, where *last(s)* denotes the time index of the last occurrence of state s in any valid plan.

Since these four states are part of every valid plan, the potential heuristic value h_{pot2} must be positive for all four states. Therefore, we turn our attention to h_{pot1}. We will now construct a witnessing quartet for h_{pot1}, as defined in Definition 9.

Let us denote these four states as a, b, c, d, respectively:

$$a : \{\langle package, loc_1\rangle, \langle truck, loc_2\rangle\}, \qquad b : \{\langle package, loc_1\rangle, \langle truck, loc_1\rangle\},$$
$$c : \{\langle package, truck\rangle, \langle truck, loc_1\rangle\}, \qquad d : \{\langle package, truck\rangle, \langle truck, loc_2\rangle\}.$$

Given that the PDDA property requires that every non-goal state has a successor with a strictly lower heuristic value, it follows that every valid plan must descend strictly with respect to h_{pot1}. Combined with the ordering of last occurrences among the landmarks, this implies:

$$h_{\text{pot1}}(a) > h_{\text{pot1}}(b) > h_{\text{pot1}}(c) > h_{\text{pot1}}(d)$$

We partition the variables as follows:

$$W = \{package\} \text{ and } M = \{truck\}.$$

Now we verify the quartet conditions:

- We already showed that $h_{pot1}(a) > h_{pot2}(b)$ and $h_{pot1}(c) > h_{pot2}(d)$.

- In both a and b, the package is at loc_1, satisfying $a^W = b^W$.
- In both a and d, the truck is at loc_2, satisfying $a^M = d^M$.
- In both c and d, the package is inside the truck, satisfying $c^W = d^W$.
- In both b and c, the truck is at loc_1, satisfying $b^M = c^M$.

All the conditions of a witnessing quartet are satisfied. The contradiction follows via the quartet criterion (Theorem 1), which says that if there exists a witnessing quartet for a potential heuristic h, then the dimension of h is at least 2.

□

The previous lemmas show that the DDA and PDDA correlation complexities of Logistics are both at least 2, indicating that solving even simple Logistics tasks requires capturing correlations across at least two dimensions. According to the structure illustrated in Figure 4.4, the correlation complexities of ∞-DDA and UDDA are never smaller than those of DDA and PDDA. This means that the lower bound of 2 established for DDA and PDDA must also apply to ∞-DDA and UDDA, as their complexity is always at least as high.
We now formalize this insight in the following corollaries.

Corollary 16.1. *The ∞-DDA correlation complexity of Logistics is at least 2*

Corollary 16.2. *The UDDA correlation complexity of Logistics is at least 2.*

7.2 Upper Bound Results

In this section, we first derive an upper bound on the UDDA correlation complexity and use it to establish a corresponding bound for the DDA, ∞-DDA and PDDA correlation complexity.

Lemmas 17. *The UDDA correlation complexity of Logistics is at most 2.*

Proof. To prove the lemma, we construct a two-dimensional heuristic h that satisfies the UDDA condition, thereby showing that the UDDA correlation complexity of the Logistics domain is at most 2. The heuristic is inspired by the framework introduced by Francès Medina et al. [3], particularly the concepts they propose for the Logistic domain.

We define weights over selected binary features that relate packages to trucks, airplanes, and locations. These weights reflect the estimated closeness of a package to its goal, factoring in both its current position and the accessibility and placement of trucks or airplanes.
To formalize this, we define a function $\mathit{the\_truck} : C \to T$ that selects one designated truck per city, representing the preferred vehicle for intra-city transport. Vice versa, the variable $\mathit{the\_airplane} \in A$ representing the primary vehicle for inter-city package transport.

We define h with the following weights. Each weight reflects a distinct qualitative state of a package, ordered from most to least favourable in terms of delivery progress:

1. The package is in a truck at its destination:
   $w(\{\langle package, truck\rangle, \langle truck, dest(package)\rangle\}) = 1$
   for all $package \in P,\ truck \in T$

2. The package is in a truck in the correct city, but not at the destination:
   $w(\{\langle package, truck\rangle, \langle truck, loc\rangle\}) = 2$
   for all $package \in P,\ truck \in T,\ loc \in L$
   with $city(loc) = city(dest(package)),\ loc \neq dest(package)$

3. The package is on the ground in the correct city with the preferred truck available:
   $w(\{\langle package, loc\rangle, \langle the\_truck(city(loc)), loc\rangle\}) = 3$
   for all $package \in P,\ loc \in L$
   with $city(loc) = city(dest(package)),\ loc \neq dest(package)$

4. The package is on the ground in the correct city, but without the preferred truck available:
   $w(\{\langle package, loc_a\rangle, \langle the\_truck(city(loc_a)), loc_b\rangle\}) = 4$
   for all $package \in P,\ loc_a, loc_b \in L$
   with $loc_a \neq loc_b,\ city(loc_a) = city(loc_b) = city(dest(package)),\ loc_a \neq dest(package)$

5. The package is in an airplane at the correct city's airport:
   $w(\{\langle package, airplane\rangle, \langle airplane, airport(city(dest(package)))\rangle\}) = 5$
   for all $package \in P,\ airplane \in A$

6. The package is in an airplane in the wrong city:
   $w(\{\langle package, airplane\rangle, \langle airplane, airport(c)\rangle\}) = 6$
   for all $package \in P,\ airplane \in A,\ c \in C$
   with $c \neq city(dest(package))$

7. The package is on the ground at an airport in the wrong city, with the preferred airplane present:
   $w(\{\langle package, airport(c)\rangle, \langle the\_airplane, airport(c)\rangle\}) = 7$
   for all $package \in P,\ airplane \in A,\ c \in C$
   with $c \neq city(dest(package))$

8. The package is on the ground at an airport in the wrong city, without the preferred airplane:
   $w(\{\langle package, airport(c_1)\rangle, \langle the\_airplane, airport(c_2)\rangle\}) = 8$
   for all $package \in P,\ c_1, c_2 \in C$
   with $c_1 \neq c_2,\ c_1 \neq city(dest(package))$

9. The package is in a truck at the wrong city's airport:
   $w(\{\langle package, truck\rangle, \langle truck, airport(c)\rangle\}) = 9$
   for all $package \in P,\ truck \in T,\ c \in C$
   with $c \neq city(dest(package))$

10. The package is in a truck at a non-airport location in the wrong city:
    $w(\{\langle package, truck\rangle, \langle truck, loc\rangle\}) = 10$
    for all $package \in P,\ truck \in T,\ loc \in L$
    with $city(loc) \neq city(dest(package)),\ loc \neq airport(city(loc))$

11. The package is on the ground in a non-airport location in the wrong city, with the preferred truck there:
    $w(\{\langle package, loc\rangle, \langle the\_truck(city(loc)), loc\rangle\}) = 11$
    for all $package \in P,\ loc \in L$
    with $city(loc) \neq city(dest(package)),\ loc \neq airport(city(loc))$

12. The package is on the ground at a non-airport location in the wrong city and no truck available:
    $w(\{\langle package, loc_a\rangle, \langle the\_truck(city(loc_a)), loc_b\rangle\}) = 12$
    for all $package \in P,\ loc_a, loc_b \in L$
    with $loc_a \neq loc_b,\ city(loc_a) = city(loc_b) \neq city(dest(package)),\ loc_a \neq airport(city(loc))$

By construction, any arbitrary package $package \in P$ is either delivered to its goal location or belongs to exactly one of the previously defined cases 1) to 12). We now proceed to verify the descending condition of UDDA, which requires that for all non-goal states $s \in (S \setminus G)$, there exists a successor $t \in \text{succ}(s)$ such that $h(t) < h(s)$. To establish this property, we make a case distinction over s:

Since s is not a goal state, there must be at least one package that has not yet reached its target location.We will now consider the possible actions in a specific order, where each step is only considered if none of the previous conditions apply. That is, we first attempt (a); if (a) is not possible, then we attempt (b); if (b) is not possible, then (c), and so on. We begin by examining cases where unloading a vehicle is possible.

(a) If a package is currently in case 1, unloading it places the package directly at its goal location, reducing the heuristic value by 1.

(b) In the situation where a package is in case 5, unloading it places the package on the ground. If the preferred truck is present, this action transitions the package to case 3, resulting in a heuristic reduction of 2. If the preferred truck is not present, the package transitions instead to case 4, reducing the heuristic by 1.

(c) When a package is in case 9, unloading changes its state based on the presence of the preferred airplane. If the airplane is available, the package moves to case 7, decreasing the heuristic by 2. If the airplane is absent, it transitions to case 8, which reduces the heuristic by 1.

Next, we consider all cases where loading a package into a vehicle is possible:

(d) If a package is currently in case 3, loading it into the preferred truck moves it to case 2, reducing the heuristic value by 1.

(e) Similarly, if a package is in case 7, loading it into the airplane transitions it to case 6, decreasing the heuristic value by 1.

(f) Finally, if a package is in case 11, loading it into the truck moves it to case 10, which also reduces the heuristic value by 1.

Since all potential load and unload actions have been addressed, we can safely assume that no further such operations are possible. Consequently, any further progress towards the goal state must involve moving a vehicle to enable further improvements.

(g) In the case where a package is in case 2, driving the truck to the package's goal destination moves it to case 1, resulting in a heuristic reduction of 1. This action can also cause packages in case 4 to transition to case 3, in case 10 to transition to case 9, and in case 12 to transition to case 11, each reducing the heuristic by an additional 1, guaranteeing a total reduction of at least 1.

(h) If a package is in case 4, driving the preferred truck to its location moves it to case 3, reducing the heuristic by 1. Additionally, this action may cause packages from case 10 to move to case 9 and from case 12 to move to case 11, each reducing the heuristic value by 1.

(i) When a package is in case 6, flying the airplane to the airport in the correct city for the package moves it to case 5, decreasing the heuristic value by 1. This action can also shift packages from case 8 to case 7, further reducing the heuristic by 1.

(j) If a package is in case 8, flying the preferred airplane to the package's current location moves it to case 7, reducing the heuristic by 1.

(k) For a package in case 10, driving the truck to the airport moves it to case 9, again decreasing the heuristic by 1.

(l) Finally, if a package is in case 12, driving the preferred truck to the package's location transitions it to case 11, reducing the heuristic by 1.

This analysis covers all possible transitions and demonstrates that for every non-goal state s, there is always a successor state t such that $h(t) < h(s)$, thereby satisfying the descending condition of UDDA. With this, we conclude that the UDDA correlation complexity of Logistics is at most 2.

□

The previous lemma demonstrates that the UDDA correlation complexity of Logistics is at most 2. According to the structure illustrated in Figure 4.4, the correlation complexities of DDA, ∞-DDA, and PDDA are never greater than that of UDDA. Therefore, the upper bound of 2 established for UDDA also holds for these variants, ensuring that their correlation complexities do not exceed this limit.
We now formalize this result in the following corollaries.

Corollary 17.1. *The DDA correlation complexity of Logistics is at most 2.*

Corollary 17.2. *The ∞-DDA correlation complexity of Logistics is at most 2.*

Corollary 17.3. *The PDDA correlation complexity of Logistics is at most 2.*

7.3 Exact Correlation Complexities

Building on the lower and upper bounds established in the previous lemmas and corollaries, we can now draw the key conclusion of this chapter: the DDA, UDDA, ∞-DDA and PDDA correlation complexity of Logistics are 2.

Theorem 8. *The DDA, UDDA, ∞-DDA and PDDA correlation complexity of Logistics is 2.*

Proof. By Lemma 15 and 16 and Corollary 16.1 and 16.2, we establish that the correlation complexity of Logistics for each DDA variant, specifically, the DDA, UDDA, ∞-DDA, and PDDA, is bounded below by 2. Furthermore, the corresponding upper bounds are confirmed by Lemma 17 and Corollary 17.1 to 17.3, each demonstrating that the correlation complexity does not exceed 2. Since the lower and upper bounds are both precisely 2, it follows that the correlation complexity of Logistics for all considered DDA variants is exactly 2. □

8

Linear Termes

The Termes domain is inspired by the Harvard TERMES project [11], which explores how robot swarms, modeled after termites, can build complex structures by carrying and stacking blocks. Termites are known for constructing large mounds cooperatively, and the TERMES robots emulate this behavior to build three-dimensional structures. These robots can climb on structures and must often build ramps to reach higher areas, introducing significant planning complexity.

The domain, originally introduced by Koenig and Kumar [7] as a benchmark for cooperative multi-agent planning, captures key challenges such as limited workspace, long sequences of actions, and the necessity for precise coordination. While the full domain involves multiple agents handing off blocks and avoiding collisions, the IPC 2018 benchmark version[1] simplifies this to a single robot that moves, picks up blocks, and places them while respecting structural constraints.

In this work, we focus on a simplified *linear variant* of the Termes domain. The environment is a one-dimensional grid of $m + 1$ fields arranged in a straight line. A single robot starts at position 0 (the depot), where it picks up blocks, and it must build a tower of height $n \leq m$ at position m (the final field). Each field i has an associated height variable $\mathit{field}_i \in 0, \ldots, n$.

The domain is formally defined using the following variables:

- *robot*: the robot's current position, $\mathrm{dom}(\mathit{robot}) = \{0, \ldots, m\}$
- field_i: height of field i, $\mathrm{dom}(\mathit{field}_i) = \{0, \ldots, n\}$
- *hand*: whether the robot is carrying a block, $\mathrm{dom}(\mathit{hand}) = \{\mathit{clear}, \mathit{block}\}$

All fields are initially at height zero, and the robot starts at field 0 with an empty hand. The goal is to construct a tower of fixed height n at the final field, with all other fields remaining

[1] https://ipc2018-classical.bitbucket.io/domains.html

at height zero and the robot ending at field 0 with empty hands.

In the Linear Termes domain, the robot performs three fundamental actions: moving, picking up blocks, and placing blocks. Movement is restricted to adjacent fields where the height difference between the current position and the target field does not exceed one block. To place a block, the robot must be carrying one and can only place it on a neighbouring field whose height matches the robot's current field height exactly. Conversely, picking up a block requires the robot's hand to be empty and the neighbouring field to be precisely one block higher than its current position. Additionally, the depot, located at the first field in the line, serves as a special point for block exchange. When standing on the depot, the robot can pick up a block directly from it, loading blocks for transport. Similarly, if carrying a block, the robot can place it back onto the depot, effectively returning blocks to the source. This simpler version still keeps the main challenges of the original domain, like building step-by-step and carrying only one block at a time. Because of this, it works well for testing single-robot planning methods.

In this chapter, we show that no UDDA heuristic exists for the Linear Termes domain. Furthermore, we prove that both the ∞-DDA and PDDA correlation complexities have a lower bound of 2 and an upper bound of 3 in this domain.

8.1 Non-Existence of UDDA

In this section, we show that no heuristic function can satisfy the UDDA property in the Linear Termes domain. The key insight lies in the presence of dead-end states, states from which no sequence of actions can reach the goal. Such states violate the fundamental UDDA requirement that every non-goal state must have at least one successor with a strictly lower heuristic value. We formalize this argument in the following theorem.

Theorem 9. *There exists no UDDA heuristic for Linear Termes.*

Proof. Consider a Linear Termes instance Π with $m = 3$ fields. Let the robot be at position 2 (the last field), which has height 0. Assume the middle field (field 1) has height 2, forming a tall tower between the depot (field 0) and field 2. The robot is not carrying any block.

In this configuration, the robot cannot move left from field 2 to field 1, as the height difference (2 blocks) exceeds the maximum climbable threshold. It also cannot move right, since it is already at the last field. Furthermore, the robot cannot pick up a block because both neighbouring fields (in front and behind) are either inaccessible due to height constraints or do not exist. It also cannot place a block, as its hand is empty. Therefore, no heuristic function can satisfy the UDDA condition in the Linear Termes domain. □

8.2 Lower Bound Results

In the following, we establish a lower bound on the PDDA correlation complexity and use the result to derive a corresponding lower bound for the ∞-DDA correlation complexity.

Lemmas 18. *The PDDA correlation complexity of Linear Termes is at least 2.*

Proof. We assume for contradiction that the PDDA correlation complexity of Linear Termes is less than 2. Then there exists a potential heuristic function h_{pot} of dimension 1 that satisfies the PDDA properties.

Consider a Linear Termes planning task Π with $m + 1 = 3$ fields and a goal height of $n = 2$. Initially all fields are at height 0, and the robot is located at field 0 with an empty hand. The goal is that the last field (field 2) is height 2, all other fields are height 0, and the robot is located at field 0 with an empty hand.

To achieve this goal, the robot must pick up at least three blocks from the depot at field 0: two to build the tower at field 2 and one to use as a ramp at field 1. After the tower is constructed, the robot must remove the ramp, which involves handing at least one block back to the depot. Therefore, the plan necessarily includes at least two pickup actions and at least one hand over action at the depot.

Now consider the effect of a pickup action at the depot. Regardless of the current configuration, picking up a block only changes the value of the variable *hand* from *free* to *block*, with no other variables affected. Let s be the state before the action and s' the state after. Since h_{pot} satisfies the PDDA property, the following inequality must hold:

$$h(s) - h(s') = w(\{\langle \mathit{hand}, \mathit{block}\rangle\}) - w(\{\langle \mathit{hand}, \mathit{free}\rangle\}) < 0,$$

which implies:

$$w(\{\langle \mathit{hand}, \mathit{block}\rangle\}) < w(\{\langle \mathit{hand}, \mathit{free}\rangle\}).$$

Now consider the reverse operation, handing over a block to the depot. This action changes the variable *hand* from *block* to *free*, again affecting no other variables. Let s be the state before the action and s' the state after. Applying the PDDA property again, we obtain:

$$h(s) - h(s') = w(\{\langle \mathit{hand}, \mathit{free}\rangle\}) - w(\{\langle \mathit{hand}, \mathit{block}\rangle\}) < 0,$$

which implies:

$$w(\{\langle \mathit{hand}, \mathit{free}\rangle\}) < w(\{\langle \mathit{hand}, \mathit{block}\rangle\}).$$

Combining both inequalities yields:

$$w(\{\langle \mathit{hand}, \mathit{free}\rangle\}) < w(\{\langle \mathit{hand}, \mathit{block}\rangle\}) < w(\{\langle \mathit{hand}, \mathit{free}\rangle\}),$$

which is a contradiction.

Therefore, our initial assumption must be false. No one-dimensional potential heuristic function satisfying the PDDA property can exist for this task. Hence, the PDDA correlation complexity of Linear Termes is at least 2. □

The previous lemmas show that the PDDA correlation complexity of Linear Termes is at least 2. According to the structure illustrated in Figure 4.4, the ∞-DDA correlation complexity is

never smaller than those of PDDA. This means that the lower bound of 2 established for PDDA must also apply to ∞-DDA, as its complexity is always at least as high.

Corollary 18.1. *The ∞-DDA correlation complexity of Linear Termes is at least 2.*

MiniZinc-Based Exploration of the ∞-DDA Property

While we have proven that the correlation complexity is at least 2, establishing a lower bound of 3 remains an open challenge. To explore this, we implemented a MiniZinc [10] model encoding the constraints that any two-dimensional potential heuristic satisfying the ∞-DDA property must fulfil for a given planning task. By formalizing these constraints, the MiniZinc model effectively characterizes the space of all heuristics of correlation complexity 2 that comply with ∞-DDA for a particular planning problem. If the model finds no solution, this indicates that no such heuristic exists for that task, thus serving as a counterexample that demonstrates the insufficiency of correlation complexity 2 heuristics. Since the existence of a single counterexample is enough to prove that a lower bound of 3 is necessary, this approach provides a practical and rigorous way to investigate the problem. In the next subsections, we detail the MiniZinc encoding and present the results of our computational experiments conducted on specific planning tasks.

Formal Model

The core goal is to find integer weights $w_{a,b}$ that define a quadratic heuristic function h over features of states, such that the heuristic satisfies the ∞-DDA property. These weights are variables in our MiniZinc constraint model.

Formally, let S be the set of all possible states of a specific Linear Termes planning task Π with $m+1$ fields and a goal height of n. Let A be the set of atomic features. The full feature set, including both atomic features and their pairwise conjunctions, is defined as:

$$F = A \cup (A \times A).$$

We define $G \subseteq S$ as the set of goal states. The transition relation $succ \subseteq S \times S$ encodes which states can be reached from others via a valid action. For each feature $f \in F$, an indicator function $\phi_f : S \to \{0,1\}$ specifies whether a feature f is present in a given state. All of these, G, $succ$, and the collection of functions $\{\phi_f\}_{f \in F}$, are given as constants within our MiniZinc model.

For each state $s \in S$, we introduce a variable $h(s)$ representing the heuristic value at s. These variables are constrained by the following relation in our model:

$$h(s) = \sum_{a \in A} w_{a,a}\, \phi_a(s) \;+\; \sum_{\substack{a,b \in A \\ a<b}} w_{a,b}\, \phi_a(s)\, \phi_b(s),$$

where the coefficients $w_{a,b} \in \mathbb{R}$ are also variables.
Additionally, for each pair $(a,b) \in A \times A$, we introduce a binary variable $w_\infty[a,b] \in \{0,1\}$ that indicates whether the combination of atomic features a and b causes the heuristic value to be

infinite. Using these variables, we define an auxiliary variable $h(s) \in \mathbb{Z}$ for each state $s \in S$, constrained as:

$$h_\infty(s) = \sum_{(a,b)\in A\times A} \phi_a(s) \cdot \phi_b(s) \cdot w_\infty[a,b].$$

Furthermore, $h_\infty \geq 1$ if any such infinite feature pair is present in state s.
To satisfy the ∞-DDA property, the model enforces the following constraint for every state $s \in S$:

$$h_\infty(s) \geq 1 \quad \text{or} \quad s \in G \quad \text{or} \quad \exists s' \in S \text{ such that } (s,s') \in \mathit{succ},\ h_\infty(s') = 0, \text{ and } h(s') < h(s).$$

Finally, the initial state $s_1 \in S$ is constrained to have a finite heuristic value:

$$h_\infty(s_1) = 0.$$

These constraints formally capture the ∞-DDA property: the initial state is finite, and every non-goal, finite-valued state must have a strictly improving successor.

Dataset Generation

We generated the input data for our MiniZinc model using a Python script that enumerates all possible states of a Linear Termes planning task Π with m+1 fields and a goal height of n. Each state is defined by the vector of stack heights, robot position, and hand status. For each state, the script computes a Boolean feature vector capturing these details, identifies goal states based on the configuration of the blocks and the robot's status, and constructs the successor relation by simulating valid robot actions such as moving, picking up, and placing blocks. The resulting data, including feature vectors, successor relations, and goal indicators, is exported in `.dzn` format for use within the MiniZinc solver.

Computational Results

The rapid growth of the state space $|\mathcal{S}|$ significantly limits the size of problem instances we can analyse computationally. In our experiments, the largest instance tested consisted of 5 fields with a goal height of 3 (see Table 8.1). As shown in the table, the computational time increases substantially with larger state spaces corresponding to higher field counts and goal heights. Despite this limitation, our experiments found no violations of the ∞-DDA property under quadratic heuristics. This supports the hypothesis that correlation complexity 2 heuristics may be sufficient, leaving a formal lower bound of 3 an open question for further research.

8.3 Upper Bound Results

To derive an upper bound on the ∞-DDA correlation complexity, we have to introduce several partial heuristics. For that, we will divide the Linear Termes problem into 3 subproblems: (1) fundamental structural constraints, (2) the construction of a ramp, and (3) the destruction of a ramp. Each of these subproblems can be further divided into smaller components, each giving rise to its own sub-heuristic.

Height					
	3			10m 9s	2h 20m 39s
	2		5s 340ms	46s 227ms	15m 1s
	1	679ms	2s 537ms	5s 457ms	32s 881ms
		2	**3**	**4**	**5**
		Fields			

Table 8.1: Time taken to solve the MiniZinc model for different numbers of fields and tower goal heights. Each cell reports the computational time corresponding to the specified height and field count.

Later, we will combine these sub-heuristics in two different ways. The first approach involves assigning a scalar weight to each heuristic and summing them to form a composite heuristic. The second approach introduces a new operation on feature sets, called feature multiplication, that modifies heuristics based on additional features before combining them. Details on these methods will be explained later.

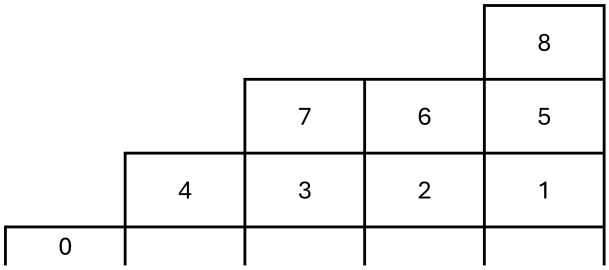


Figure 8.1: Illustration of a Linear Termes task consisting of five fields. The first field (left) serves as the depot, and the final field (right) has a goal height of three blocks. The robot is not depicted, as the figure represents only the final configuration of the completed ramp. Numbers on the blocks indicate the intended construction order of the ramp.

8.3.1 General Constraints

The first partial heuristic, denoted h_1, encodes fundamental structural constraints of the task. It ensures that fields become progressively higher as their distance from the depot increases, preserving the ramp-like structure. Building is only allowed layer by layer, as illustrated in Figure 8.1, meaning the next layer can only be started once the previous one is complete, except when the tower at the end is already finished, in which case it is allowed to stand independently. Furthermore, the height difference between adjacent fields must not exceed one block, again with the exception of a completed tower. Finally, the robot is never allowed to move onto the last field. If the tower there is more than one block higher than the previous field, the robot would be unable to take any further actions when positioned on it, resulting in a dead-end. Moreover, standing on the last field is never required, as placing a block on it can always be done from the second-to-last field.

These constraints are encoded in the following weight function w_1, which defines the heuristic h_1.

1. Each field has a maximum allowed height:
   $w_1(\{\langle \mathit{field}_i, h_i\rangle\}) = \infty$
   for all $i \in \{0,\dots,m\},\ h_i \in \{0,\dots,n\}$ with $h_i > i$

2. Only the last field (field m) is allowed to reach the goal height n, all other fields must remain strictly below it:
   $w_1(\{\langle \mathit{field}_i, n\rangle\}) = \infty$
   for all $i \in \{0,\dots,m-1\}$.

3. Fields further away from the depot must not be lower than fields that are near the depot:
   $w_1(\{\langle \mathit{field}_i, h_i\rangle, \langle \mathit{field}_j, h_j\rangle\}) = \infty$
   for all $i,j \in \{0,\dots,m\},\ h_i, h_j \in \{0,\dots,n\}$ with $i < j$ and $h_i > h_j$

4. The robot must never stand on the last field:
   $w_1(\{\langle \mathit{robot}, m\rangle\}) = \infty$

5. A neighbouring field (except the last field) may be at most one block higher:
   $w_1(\{\langle \mathit{field}_i, h_i\rangle, \langle \mathit{field}_{i+1}, h_{i+1}\rangle\}) = \infty$
   for all $i \in \{0,\dots,m-2\}$, $h_i, h_{i+1} \in \{0,\dots,n\}$ with $h_{i+1} \geq h_i + 2$.

6. A new layer cannot be started until the current one is completed (except the last field):
   $w_1(\{\langle \mathit{field}_i, h_i\rangle, \langle \mathit{field}_j, h_j\rangle\}) = \infty$
   for all $i,j \in \{0,\dots,m-1\}$, $h_i, h_j \in \{0,\dots,n\}$ with $i < j$, $h_i < i$, and $h_j \geq h_i + 2$.

7. The last field may be at most one block higher than the second-last, unless the goal height is reached:
   $w_1(\{\langle \mathit{field}_{m-1}, h_{m-1}\rangle, \langle \mathit{field}_m, h_m\rangle\}) = \infty$
   for all $h_{m-1}, h_m \in \{0,\dots,n\}$ with $h_{m-1} + 2 \leq h_m < n$.

8. The last field is treated as part of a layer and must not exceed earlier incomplete fields by more than one block, unless the goal height is reached:
   $w_1(\{\langle \mathit{field}_i, h_i\rangle, \langle \mathit{field}_m, h_m\rangle\}) = \infty$
   for all $i \in \{0,\dots,m-1\}$, $h_i, h_m \in \{0,\dots,n\}$ with $h_i < i$, and $h_i + 2 \leq h_m < n$.

We now establish two key properties of the heuristic function h_1: for states with a finite heuristic value, (i) walking to any neighbouring field, excluding the last field, is always possible, and (ii) picking up or handing over a block at the depot is always possible when the robot is positioned on field 1. Both actions preserve the finiteness of the heuristic value.

Lemmas 19. *Let s be a non-goal state such that $h_1(s)$ is finite. Then the following actions are always possible and preserve the finiteness of the heuristic value:*

(i) Walking to any neighbouring field, excluding the final field.

(ii) Picking up or handing over a block at the depot when the robot is positioned on field 0.

Proof. We consider each action in turn.

(i) Walking:

In states where h_1 is finite, none of the features listed in Item 5 of w_1 are active. This implies that the height difference between any two adjacent fields, excluding those involving the last field, is at most one. Furthermore, since $s \in S_{\text{fin}}$, the weight function w_1 ensures that the robot is not positioned on the last field. As a result, the robot can move left or right, provided a neighbouring field exists that is not the last field.

Additionally, since walking only changes the robot's position, it can activate at most one feature with infinite weight, specifically, the one associated with stepping onto the final field. However, such cases are excluded in the lemma. Therefore, every other walking action contributes only finite terms to the heuristic value.

(ii) Depot interaction:

The variable *hand* does not occur in any feature with infinite weight. Additionally, the depot is always located at field 0 and remains at height 0. Therefore, whenever the robot is positioned on field 0, it can safely pick up or hand over a block without causing the heuristic value to become infinite. Hence, the action preserves the finiteness of the heuristic value.

In both cases, the permitted actions preserve the finiteness of the heuristic value.

□

8.3.2 Construction Phase

The second subproblem addresses the construction of the ramp, specifically focusing on how the robot moves and where it places blocks. The core idea is to define a building sequence such that each field, depending on its current height, influences the robot by pulling it towards a specific target position that supports ongoing ramp construction. This behaviour is modelled using a moving weight function defined over pairs of robot positions and field heights. The goal is to steer the robot to the most effective position for placing the next block.

For that, we define a sequence of ramp construction steps, as illustrated in Figure 8.1. The ramp is built layer by layer, and within each layer, from back to front, resulting in a well-defined, sequential order of building steps.
We formalize this by defining the set of planned building steps as

$$\mathcal{B} = \{(i, h) \mid i \in \{0, \ldots, m\},\, h \in \{1, \ldots, i\}\}$$

which reflects the assumption that each field i must ultimately reach height i. We then define a bijective function

$$f : \mathcal{B} \to \{0, \ldots, |\mathcal{B}|\}$$

that maps each field–height pair to a unique building step index b. The function f encodes our desired build sequence: progressing layer by layer, and within each layer, from the back towards the depot. Since f is bijective, we can also recover the corresponding field–height pair using its inverse $f^{-1}(b)$.

The interpretation is that when the ramp at field i has reached height h, it means that the $f(i,h)$-th construction step has been completed. This completion then triggers the robot's attention to move on to the next step, $f(i,h)+1$, ensuring sequential progress.

Moving Heuristic

To guide the robot effectively during construction, we break down the global movement objective into localized components. Each field, based on its current state, defines its own heuristic that pulls the robot towards a position that would enable progress at that specific location. These local moving heuristics form the foundation of the robot's movement logic and are later combined into a global guidance function.

Local Moving Heuristic

To formalize this, let $\mathcal{F}$ be a set of state features, and let $F \in \mathcal{F}$ be a single state feature such that $\langle \text{robot}, i\rangle \notin F$ for all $i \in \{0,\dots,m\}$. We define a local moving heuristic $h^{t,F}$ via the weight function:

$$w^{t,F}(F \cup \{\langle \text{robot}, i\rangle\}) = |t-i|, \quad \text{for all } i \in \{0,\dots,m\},$$

where $t \in \mathbb{N}$ is a fixed target position and i denotes the robot's current position.

Each of these heuristics, when considered individually, changes by ± 1 depending on whether the robot moves towards or away from the preferred target. This behaviour arises directly from the structure of the weight functions, which is defined as absolute distances. We specify this in the following lemma:

Lemmas 20. *Let $\mathcal{F}$ be a set of state features, let $F \in \mathcal{F}$ be a single state feature such that $\langle robot, i\rangle \notin F$ for all $i \in \{0,\dots,m\}$, and let $h^{t,F}$ be a local moving heuristic with preferred field $t \in \{0,\dots,m-1\}$.*
Assume that:

- *s and s' are two states such that the robot moves from field i to $i' = i \pm 1$,*
- *The feature $F \in \mathcal{F}$ is present in both s and s' (i.e. $F \in s$ and $F \in s'$).*

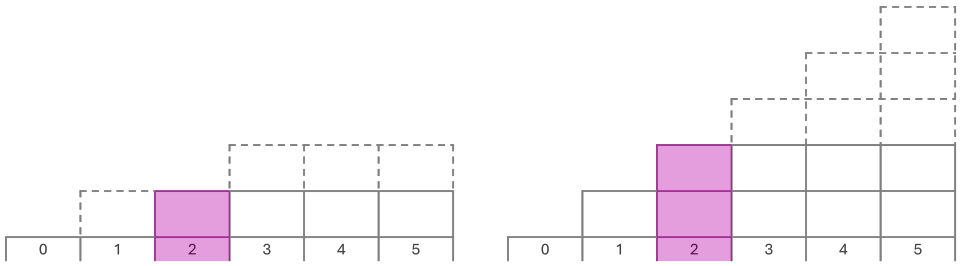
Then the heuristic difference satisfies:

$$h^{t,F}(s') - h^{t,F}(s) = \begin{cases} -1 & \text{if the robot moves towards the target } t \\ 1 & \text{otherwise} \end{cases}$$

Proof. Let s be a state in which the robot is positioned at field i, and let s' be its successor state where the robot has moved to $i' = i \pm 1$. By the definition of the heuristic $h^{t,F}$, we have

$$\begin{aligned} h^{t,F}(s') - h^{t,F}(s) &= w(F \cup \{\langle \text{robot}, i'\rangle\}) - w(F \cup \{\langle \text{robot}, i\rangle\}) \\ &= |t-i'| - |t-i|. \end{aligned}$$

It follows from a simple case distinction that a move bringing the robot one step closer to the target decreases the heuristic value by 1, while a move away from the target increases it by 1. □



(a) Case 1: Field 2 has not yet reached its target height of 2. Regardless of the state of other fields, the robot is encouraged to move to field 0 and place a block on field 1.

(b) Case 2: Field 2 has reached its target height of 2. Again independent of the state of other fields, the robot is now encouraged to move to field 4 and place a block on field 5.

Figure 8.2: A Linear Termes planning task Π with 6 fields and a goal height of 5. The system observes field 2 in two different states to illustrate how its height affects the local moving heuristic. In both cases, the system considers only the state of field 2, which is shaded to highlight its independence from the rest of the configuration. Dashed blocks represent positions where blocks may or may not have been placed.

To guide the robot's movement, we introduce a family of local moving heuristics $\{h^{\uparrow}_{j,h_j}\}$, defined for each field index $j = 0, \ldots, m$ and corresponding height $h_j = 0, \ldots, n$. Each $h^{\uparrow}_{j,h_j}$ directs the robot towards a target field based only on the height of field j, independent of other fields. This behaviour is illustrated in Figure 8.2 and defined as follows:

- Case 1: Field j is below the target height, i.e. $h_j < j$:
  $h^{\uparrow}_{j,h_j} = h^{(j-2),\{\langle \mathit{field}_j, h_j \rangle\}}$, meaning that for the next construction step, the robot needs to place a block on field $j - 1$.
- Case 2: Field j has reached its target height, i.e. $h_j = j$:
  $h^{\uparrow}_{j,j} = h^{(m-1),\{\langle \mathit{field}_j, j \rangle\}}$, indicating that for the next construction step, the robot must place a block on the final field m.

Returning to the depot

A similar principle applies when the robot is not carrying a block: it should return to the depot located at field 0. This behaviour is captured by a local moving heuristic $h_{\text{depot}} = h^{0,\{\langle \mathit{hand}, \mathit{free} \rangle\}}$.

Global Weighted Moving Heuristic

The local moving heuristics described earlier allow each field to influence the robot's movement based on the field's current construction state. However, these influences act simultaneously and independently, which can lead to conflicting signals about where the robot should go next. To resolve this and guide the robot decisively towards the most relevant part of the ramp, we introduce a priority weighting scheme.

Each field-height pair $(i, h) \in \mathcal{B}$ is assigned a weight of $2^{f(i,h)}$. This assignment, together with the statement from Lemma 20 that each local moving heuristic changes by at most ± 1 when the robot moves by one field, guarantees that the influence of the last construction step always outweighs the combined influence of all previous steps. As a result, the robot's moving heuristic becomes unambiguous: when carrying a block, only a single field can dominate the decision, providing a clear and consistent guide for movement.

When the robot is not holding a block, the depot return heuristic (introduced earlier) should override any construction-related preferences. To enforce this behaviour, we define a special parameter:

$$\mathit{walking\_back} = 2^{|\mathcal{B}|+1}$$

This value exceeds the total combined pull from all building step weights defined above, ensuring that returning to the depot always dominates when the robot is empty-handed.

To formally define a weighted moving heuristic, we introduce two basic operations on potential heuristics: addition and scalar multiplication. These allow us to compose multiple heuristic components into a single unified potential heuristic.

Definition 10 (Addition of Potential Heuristics)**.** Let h_1 and h_2 be potential heuristics with respective feature sets $\mathcal{F}_1$ and $\mathcal{F}_2$, and corresponding weight functions $w_1 : \mathcal{F}_1 \to \mathbb{R}$ and $w_2 : \mathcal{F}_2 \to \mathbb{R}$. We define the sum $h_{\text{sum}} = h_1 + h_2$ as a new potential heuristic over the unified feature set $\mathcal{F}_{\text{sum}} = \mathcal{F}_1 \cup \mathcal{F}_2$, with the combined weight function $w_{\text{sum}} : \mathcal{F}_{\text{sum}} \to \mathbb{R}$ given by:

$$w_{\text{sum}}(F) = \begin{cases} w_1(F), & \text{if } F \in \mathcal{F}_1 \setminus \mathcal{F}_2, \\ w_2(F), & \text{if } F \in \mathcal{F}_2 \setminus \mathcal{F}_1, \\ w_1(F) + w_2(F), & \text{if } F \in \mathcal{F}_1 \cap \mathcal{F}_2. \end{cases}$$

In addition to combining heuristics additively, we may also want to control their relative influence. For this, we introduce scalar multiplication.

Definition 11 (Scalar Multiplication of Potential Heuristics)**.** Let h be a potential heuristic with feature set $\mathcal{F}$ and weight function $w : \mathcal{F} \to \mathbb{R}$, and let $a \in \mathbb{R}$ be a scalar. The scaled heuristic h_{scaled} is defined by multiplying each weight by a, i.e.

$$w_{\text{scaled}}(F) = a \cdot w(F) \quad \text{for all } F \in \mathcal{F}.$$

Then the corresponding heuristic value for a state s is:

$$h_{\text{scaled}}(s) = \sum_{F \in \mathcal{F}} w_{\text{scaled}}(F)[F \subseteq s] = a \cdot \sum_{F \in \mathcal{F}} w(F)[F \subseteq s] = a \cdot h(s).$$

With these operations in place, we can now define the global weighted moving heuristic as a linear combination of the depot return signal and the sequence-guided construction heuristics:

$$h^{\uparrow}_{\text{walk}} = \mathit{walking\_back} \cdot h^{\uparrow}_{\text{depot}} + \sum_{i=0}^{m} \sum_{h=0}^{n} 2^{f(i,h)} \cdot h^{\uparrow}_{i,h} \tag{1}$$

To illustrate the effect of this global weighted moving heuristic in practice, we now consider a concrete example. The following scenario demonstrates how the exponential weighting guides the robot's behaviour on a small ramp, showing how each field–height pair contributes to the movement decision depending on the current state of construction.

Example 2 (Construction Guidance on a 4-Field Ramp)**.** Consider a ramp with $m + 1 = 4$ fields and a goal height of $n = 3$. The resulting exponential weights assigned to each field–height pair are:

$h = 3$	0	0	0	64
$h = 2$	0	0	32	16
$h = 1$	0	8	4	2
$a_{i,h}$	$i = 0$	$i = 1$	$i = 2$	$i = 3$

The entry in each cell represents the weight $2^{f(i,h)}$ for the field–height pair (i, h). For example, the weight for field 1 having height 1 is $2^3 = 8$.
Now assume the current ramp state is:

$$(h_0, h_1, h_2, h_3) = (0, 1, 1, 2).$$

The next valid construction step, according to the desired build order (layer by layer, back to front), is to place the second block on field 2 (i.e., complete height 2 at $i = 2$). From the weight matrix, the field–height pair $(3, 2)$ has a weight of $2^4 = 16$, which is currently the highest relevant active weight in the construction sequence. The robot's global moving heuristic will therefore, when carrying a block, be dominated by the local moving heuristic for this next step, $h_{3,2}$, pulling it towards field 1 to place a block on field 2.

This example illustrates how the exponential weighting guides the robot towards the next construction step. The following lemma formalizes this intuition by showing that moving towards the field where construction will continue is always feasible and results in a decreased heuristic value.

Lemmas 21. *Let s be a non-goal state with a finite h_1 heuristic value. Let $h^{\uparrow}_{walk}$ be the weighted moving heuristic defined by the weight function in Eq.* (1)*.*
Assume the robot is currently carrying a block. Then there exists a field p such that the next construction step is to place a block onto field $p + 1$. When not already positioned on field p, then moving towards field p is a feasible action that does not incur an infinite penalty from h_1. Moreover, this movement results in a strictly decreased heuristic value $h^{\uparrow}_{walk}$ in the resulting state s'.

Proof. By Lemma 19, walking to a neighbouring field, excluding the final field, is always feasible in states with finite h_1, and such a move does not introduce an infinite penalty in the heuristic.

Since s is not a goal state, at least one field is incomplete. Because $h_1(s) < \infty$, it follows from the construction of h_1 that all building steps up to some j are complete. Let this j be the index of the last completed building step. By definition, the next step is to place a block

onto the field associated with building step j. Let $f^{-1}(j) = (i, h_i)$ denote the field–height pair corresponding to the j-th building step. We define the field as

$$p = \begin{cases} m-1, & \text{if } h_i = i, \\ i-1, & \text{otherwise.} \end{cases}$$

In either case, the next placement is at field $p+1$. Note that the term $\mathit{walking\_back} \cdot w^{\uparrow}_{\text{depot}}$ does not contribute to the heuristic difference here because the robot is carrying a block, so the depot-related weight remains constant during this move. From Lemma 20, we know that taking a single step changes each local moving heuristic by -1 when walking towards its target field and +1 when walking away from it. In the worst-case scenario, moving towards p increases the heuristic values for all $k < j$ by +1 and decreases the heuristic value for index j by -1. The total change in the weighted moving heuristic is then:

$$h^{\uparrow}_{\text{walk}}(s') - h^{\uparrow}_{\text{walk}}(s) = -2^j + \sum_{k=0}^{j-1} 2^k = -2^j + (2^j - 1) = -1 < 0.$$

Therefore, the weighted moving heuristic strictly decreases after moving towards p, completing the proof. □

Having shown the benefit of moving towards the next construction site, we now investigate the implications of completing the construction step by placing the block.

Lemmas 22. *Let s be a non-goal state with a finite h_1 heuristic value. Let $h^{\uparrow}_{walk}$ be the weighted moving heuristic defined by the weight function in Eq.* (1).
Assume the robot is currently carrying a block and is positioned on field p such that placing a block onto field $p+1$ is a feasible action that preserves the finiteness of h_1, and the resulting state s' satisfies

$$h^{\uparrow}_{walk}(s') - h^{\uparrow}_{walk}(s) < \mathit{walking\_back} \cdot m.$$

Proof. By Lemma 21, there exists a field p such that placing a block onto $p+1$ corresponds to the next valid construction step. Let $q := p+1$. Since h_1 assigns infinite cost only to states that violate the intended construction sequence, performing this step ensures that h_1 remains finite.

Placing the block onto field q modifies two variable assignments. Specifically, it changes the assignment of the variable *hand* from *block* to *free* and updates field_q from h_q to $h_q + 1$. These changes affect the heuristic value h_{walk} as follows:

$$\begin{aligned} h^{\uparrow}_{\text{walk}}(s') - h^{\uparrow}_{\text{walk}}(s) &= w^{\uparrow}_{\text{walk}}(\{\langle \mathit{robot}, p\rangle, \langle \mathit{hand}, \mathit{clear}\rangle\}) \\ &\quad + w^{\uparrow}_{\text{walk}}(\{\langle \mathit{robot}, p\rangle, \langle \mathit{field}_q, h_q + 1\rangle\}) \\ &\quad - w^{\uparrow}_{\text{walk}}(\{\langle \mathit{robot}, p\rangle, \langle \mathit{field}_q, h_q\rangle\}) \\ &= \mathit{walking\_back} \cdot p + 2^{f(q,h_q+1)} \cdot |t_1 - p| - 2^{f(q,h_q)} \cdot |(q-2) - p| \\ &< \mathit{walking\_back} \cdot p + 2^{f(q,h_q+1)} \cdot |t_1 - p| \end{aligned}$$

where $t_1 \in \{q-2, m-1\}$ depends on the updated construction target.
We consider two cases:

1) $h_q + 1 = q$:
Then the construction step is complete, and the next target field becomes $t_1 = m - 1$. Since $p \leq m - 1$, we have $|m - 1 - p| = m - 1 - p$. Also, by definition,

$$\mathit{walking\_back} = 2^{|\mathcal{B}|+1}, \quad \text{and} \quad f(q, h_q + 1) \leq |\mathcal{B}|,$$

so $2^{f(q,h_q+1)} < \mathit{walking\_back}$. Hence:

$$\begin{aligned} h^{\uparrow}_{\text{walk}}(s') - h^{\uparrow}_{\text{walk}}(s) &< \mathit{walking\_back} \cdot p + 2^{f(q,h_q+1)} \cdot (m - 1 - p) \\ &< \mathit{walking\_back} \cdot p + \mathit{walking\_back} \cdot (m - 1 - p) \\ &= \mathit{walking\_back} \cdot (m - 1) \end{aligned}$$

2) $h_q + 1 < q$:
Then the target field remains $t_1 = q - 2$, and $|q - 2 - p| = |p - 1 - p| = 1$ (since $q = p + 1$). Thus:

$$\begin{aligned} h^{\uparrow}_{\text{walk}}(s') - h^{\uparrow}_{\text{walk}}(s) &< \mathit{walking\_back} \cdot p + 2^{f(q,h_q+1)} \\ &< \mathit{walking\_back} \cdot p + \mathit{walking\_back} \\ &= \mathit{walking\_back} \cdot (p + 1) \end{aligned}$$

Since $p \leq m - 1$, it follows that $p + 1 \leq m$, and hence in both cases:

$$h^{\uparrow}_{\text{walk}}(s') - h^{\uparrow}_{\text{walk}}(s) < \mathit{walking\_back} \cdot m.$$

□

Block Placing Heuristic

To explicitly encourage the agent to place blocks and thereby make measurable progress towards the goal, we define a block placement heuristic $h^{\uparrow}_{\text{block}}$ that rewards state transitions which increase the height of any field.
Let h_j denote the current height of field j, and recall that the goal height of field j is j. The number of blocks still required to complete field j is thus $j - h_j$. Based on this observation, we define $h^{\uparrow}_{\text{block}}$ with the block placement weight function:

$$w^{\uparrow}_{\text{block}}(\{\langle \mathit{field}_j, h_j \rangle\}) = j - h_j$$

for all $j = 0, \ldots, m$ and $h_j = 0, \ldots, j$. This heuristic captures the value of having placed a block and reflects progress towards completing the desired ramp profile. We now analyse how this heuristic behaves under different actions by the robot. We begin by establishing that robot movement alone does not affect the heuristic value h_{block}.

Lemmas 23. *Let s be a non-goal state with a finite h_1 heuristic value. Let $h^{\uparrow}_{block}$ be the block placing heuristic. Then any action in which the robot moves between fields does not change the heuristic value of $h^{\uparrow}_{block}$.*

Proof. When the robot moves, the only variable that changes is the robot's position. Since the weight function $w^{\uparrow}_{\text{block}}$ depends solely on the field heights h_j, and not on the robot's location, such a move does not change the value of $h^{\uparrow}_{\text{block}}$. □

Next, we show that placing a block reduces the heuristic value $h^{\uparrow}_{\text{block}}$, thus making measurable progress towards the goal state.

Lemmas 24. *Let s be a non-goal state with a finite h_1 heuristic value. Let $h^{\uparrow}_{block}$ be the weighted block placing heuristic.*
Suppose the robot is carrying a block. Then placing the block on any field that has not yet reached its goal height decreases the value of $h^{\uparrow}_{block}$ by exactly 1.

Proof. Placing a block affects only the height of a single field. Let this field be field j, and suppose its height increases from h_j to $h_j + 1$ after the block is placed. Let s' denote the resulting state. Then the change in the heuristic value is:

$$\begin{aligned} h^{\uparrow}_{\text{block}}(s') - h^{\uparrow}_{\text{block}}(s) &= w^{\uparrow}_{\text{block}}(\{\langle \mathit{field}_j, h_j + 1\rangle\}) - w^{\uparrow}_{\text{block}}(\{\langle \mathit{field}_j, h_j\rangle\}) \\ &= (j - (h_j + 1)) - (j - h_j) \\ &= -1. \end{aligned}$$

Hence, the heuristic value $h^{\uparrow}_{\text{block}}$ decreases by exactly 1. □

Placing a block transitions the agent to a state where its hand is empty. As a result, the walking-back heuristic (which encourages returning to the depot to pick up a new block) may increase the overall heuristic value. To ensure that placing a block is always an overall favourable action, we scale the block placement heuristic by a sufficiently large constant such that the gain from placing a block outweighs any penalty incurred by being empty-handed.
Specifically, we define:

$$\textit{placing-block} = \textit{walking-back} \cdot (m + 1)$$

where $m + 1$ is the number of fields. This guarantees that placing a block is always more rewarding than walking back to the depot, regardless of the robot's position.

We now define the complete weighted block placing heuristic:

$$h_{\text{placing}} = \textit{placing\_block} \cdot h_{\text{block}} \tag{2}$$

This scaled heuristic guides the agent to prioritize placing blocks in a way that guarantees consistent progress towards the goal configuration.

Construction Heuristic

We can now introduce the main heuristic

$$\begin{aligned} h_2 &= h^{\uparrow}_{\text{placing}} + h^{\uparrow}_{\text{walk}} \\ &= \textit{placing\_block} \cdot h^{\uparrow}_{\text{block}} + \textit{walking\_back} \cdot h^{\uparrow}_{\text{depot}} + \sum_{i=0}^{m} \sum_{h=0}^{n} 2^{f(i,h)} \cdot h^{\uparrow}_{i,h} \end{aligned}$$

for constructing a ramp.

Analysis of Successor States

We now analyse the behaviour of the heuristic $h_\uparrow = h_1 + h_2$ in states where the robot is either carrying a block or not. For every possible configuration, we prove that there exists a successor state whose heuristic value is strictly lower than that of the current state. Our approach involves a thorough case-by-case analysis, distinguishing scenarios based on the robot's position as well as whether it is carrying a block. Note that, by Lemma 21, when the robot is carrying a block, there is exactly one preferred field it should move towards to reduce the heuristic. We denote this field as field p.

Case 1: The robot is carrying a block and is on the target field p

First, we show that when the robot is positioned on the designated target field p, placing the block directly continues the ramp construction. Due to the construction of the weight matrix, this placement is encouraged, and no infinite weights are introduced in the process.

Lemmas 25. *Let s be a state in which the robot is carrying a block and is located at field p. Then, there exists a successor state s' of s such that $h_c(s') < h_c(s)$.*

Proof. By Lemma 21, there exists a field p such that placing a block at $q := p + 1$ constitutes the next valid construction step. Since h_1 assigns infinite cost only to states violating the construction sequence, placing the block at position q results in a successor state s' for which $h_1(s')$ remains finite, i.e., $h_1(s') < \infty$.

We now analyse the change in the h_2 component of the heuristic. By Lemma 24, placing a block decreases h_{block} by 1. Additionally, by Lemma 22, the act of placing a block increases h_{walk} by at most $m \cdot \mathit{walking\_back}$. Thus, the net change in h_2 is:

$$\begin{aligned} h_2(s') - h_2(s) &= h^\uparrow_{\text{placing}}(s') + h^\uparrow_{\text{walk}}(s') - (h^\uparrow_{\text{placing}}(s) + h^\uparrow_{\text{walk}}(s)) \\ &= (h^\uparrow_{\text{placing}}(s') - h^\uparrow_{\text{placing}}(s)) + (h^\uparrow_{\text{walk}}(s') - h^\uparrow_{\text{walk}}(s)) \\ &< -\mathit{placing\_block} + m \cdot \mathit{walking\_back} \end{aligned}$$

Recall the definition

$$\mathit{placing\_block} = \mathit{walking\_back} \cdot (m + 1),$$

we substitute:

$$\begin{aligned} h_2(s') - h_2(s) &< -\mathit{walking\_back} \cdot (m + 1) + m \cdot \mathit{walking\_back} \\ &= -\mathit{walking\_back} < 0 \end{aligned}$$

Hence, $h_2(s') < h_2(s)$, and since h_c is composed of h_1 and h_2 (with h_1 finite and unchanged), it follows that:

$$h_c(s') < h_c(s)$$

as required.

□

Case 2: The robot is carrying a block but is at a different field *field* i, such that $i \neq p$

Next, we handle the situation where the robot is not yet at the target field p. In this case, walking towards field p reduces the distance to the next placement site. We show that this movement leads to a successor state with strictly lower heuristic value.

Lemmas 26. *Let s be a state in which the robot is carrying a block and is located at field i with $i \in \{0, \ldots, m-1\}$ such that $\mathit{field}_i \neq \mathit{field}\ p$. Then there exists a successor state s' of s for which $h_c(s') < h_c(s)$.*

Proof. By Lemma 19, taking a walking step to a neighbouring field (excluding the final one) is always feasible in any state where $h_1 < \infty$, and this step does not introduce an infinite penalty. Thus, we focus on the change in the h_2 component, specifically the weighted moving heuristic $h^{\uparrow}_{\text{walk}}$, which depends on the robot's position.

Since $\mathit{field}_i \neq \mathit{field}_p$, the robot is not yet at its preferred destination field p. According to Lemma 21, taking a step towards field p decreases the heuristic value by at least 1. Therefore, by executing such a move, the robot reaches a successor state s' with $h_c(s') < h_c(s)$. □

Case 3: The robot is not carrying a block and is located at field_i with $i \in \{1, \ldots, m-1\}$

In this case, the robot is returning to the depot without a block. We show that stepping one field back (from *field* i to *field* $i+1$) leads to a strictly lower heuristic value.

Lemmas 27. *Let s be a state in which the robot is not carrying a block and is located at field i with $i \in \{1, \ldots, m-1\}$. Then, there exists a successor state s' of s such that $h_c(s') < h_c(s)$.*

Proof. By Lemma 19, taking a walking step to a neighbouring field (excluding the final one) is always feasible in any state where $h_1 < \infty$, and this step does not introduce an infinite penalty. Thus, we focus on the change in the h_2 component, specifically the weighted moving heuristic $h^{\uparrow}_{\text{walk}}$, which depends on the robot's position.

Since s is not a goal state, at least one field is incomplete. Moreover, $h_1(s) < \infty$ implies that all building steps up to some $j \leq |\mathcal{B}|$ are complete. Let this j be the index of the last completed building step.

When the robot takes a single step, the only variable that changes is its position. According to Lemma 20, each local moving heuristic can change by at most ± 1. Consider the worst case: moving towards the depot increases the heuristic values for all $k \leq j$ by $+1$, while the value for $h^{\uparrow}_{\text{depot}}$ decreases by -1. The resulting change in the weighted heuristic is then:

$$h_c(s') - h_c(s) = -\mathit{walking\_back} + \sum_{k=0}^{j} 2^k$$

By definition, $\mathit{walking\_back} = 2^{|\mathcal{B}|+1}$, and since $j < |\mathcal{B}| + 1$, we have:

$$h_c(s') - h_c(s) = -2^{|\mathcal{B}|+1} + \sum_{k=0}^{j} 2^k = -2^{|\mathcal{B}|+1} + (2^{j+1} - 1) < 0$$

Therefore, the weighted moving heuristic strictly decreases when the robot takes one step towards the depot with a free hand, completing the proof

□

Case 4: The robot is not carrying a block and is located at the depot (*field* 0)
Finally, we consider the case where the robot has returned to the depot and is ready to pick up a new block. We show that this action strictly decreases the heuristic value.

Lemmas 28. *Let s be a state in which the robot is not carrying a block and is located at field 0. Then, there exists a successor state s' of s such that $h_c(s') < h_c(s)$.*

Proof. Since the depot is always at height 0, the robot can always pick up a block at this location. This action updates the variable *hand* from *empty* to *block*. Because the variable *hand* does not appear in any feature of the weight function w_1, we can focus solely on the weights from w_2. The only weight in w_2 involving the *hand* variable is $w^{\uparrow}_{\text{depot}}$, which becomes inactive because the atom $\langle \textit{hand}, \textit{empty} \rangle$ is no longer part of the successor state s'. As a result, the heuristic value decreases by *walking_back* and, importantly, remains finite. □

8.3.3 Destruction Phase

The third subproblem addresses the deconstruction of the ramp. The destruction phase reverses the construction process by removing blocks in reverse order. Key differences in the heuristic framework include:

- Each local moving heuristic $h^{\downarrow}_{j,h_j}$ encourages the robot to move to field $j-1$ (instead of $j+1$ or $m-1$ during construction).
- The depot heuristic $h^{\downarrow}_{\text{depot}}$ applies when the robot carries a block, directing it back to field 0 (whereas in construction it applied when the robot's hands were empty).
- Higher fields incur greater penalties than lower ones, reversing the block placing heuristic used in construction.

The combined heuristic

$$h_3 = h^{\downarrow}_{\text{picking}} + h^{\downarrow}_{\text{walk}} = \textit{picking\_block} \cdot h^{\downarrow}_{\text{block}} + \textit{walking\_back} \cdot h^{\downarrow}_{\text{depot}} + \sum_{i=0}^{m} \sum_{h=0}^{n} 2^{f(i,h)} \cdot h^{\downarrow}_{i,h}$$

guides the robot's movements and block pickups to guarantee progress during deconstruction. For full formal definitions, lemmas, and proofs, see Appendix B.

8.3.4 Linear Termes Heuristic

To derive an upper bound on the ∞-DDA correlation complexity, we introduce a new operation on potential heuristics, called *feature multiplication*. This operation augments an existing feature set $\mathcal{F}$ by conjoining each feature in $\mathcal{F}$ with an additional feature $\tilde{F}$. Intuitively, this corresponds to creating a new heuristic that evaluates each original feature in the context of $\tilde{F}$. We denote this operation by $\tilde{F} \circledast h_{\text{pot}}$.

Definition 12 (Feature Multiplication)**.** Let Π be a planning task, $\mathcal{F}$ be a set of state features of Π, $\tilde{F}$ be a single feature of Π, $w : \mathcal{F} \to \mathbb{R} \cup \{\infty\}$, and let h_{pot} be the potential heuristic with features $\mathcal{F}$ and weight function w.
The *feature-multiplied heuristic* $\tilde{F} \circledast h_{\text{pot}}$ is defined as the potential heuristic over the set of atoms $\tilde{\mathcal{F}} = \{F \cup \tilde{F} \mid F \in \mathcal{F}\}$ with weight function $w'(F \cup \tilde{F}) = w(F)$. Then for any state s:

$$\tilde{F} \circledast h_{\text{pot}}(s) = \sum_{F \in \mathcal{F}} w(F) \cdot [(F \cup \tilde{F}) \subseteq s].$$

where $[\cdot]$ denotes the indicator function. If $F \cup \tilde{F}$ contains contradictory assignments to the same variable, then no state s can satisfy it, i.e. $(F \cup \tilde{F}) \subseteq s$ is false for all s. In such cases, the corresponding weight can be safely treated as zero without affecting the heuristic value.

Lemmas 29. *Let h_{pot} be a potential heuristic defined over a feature set $\mathcal{F}$ of dimension d, and let $\tilde{F}$ be a feature over a single variable. Then the feature-multiplied heuristic $\tilde{F} \circledast h_{pot}$ has dimension of at most $d + 1$. If the variable in $\tilde{F}$ does not already occur in a feature $F \in \mathcal{F}$, then the dimension is exactly $d + 1$.*

Proof. By definition, the feature-multiplied heuristic $\tilde{F} \circledast h_{\text{pot}}$ is constructed by forming new set of atoms $F' = F \cup \tilde{F}$ for each $F \in \mathcal{F}$. The dimension of a feature is defined as the number of distinct variables it includes.

If the variable in $\tilde{F}$ does not occur in F, then F' contains $d+1$ variables. Therefore, the resulting heuristic includes features of dimension $d + 1$, and the heuristic as a whole has dimension $d + 1$.

On the other hand, if the variable in $\tilde{F}$ already occurs in F, then F' may contain contradictory assignments. In such cases, the set of atoms F' is no more consistent, therefore no state can satisfy both variable assignments, and hence it contributes nothing to the heuristic value. These sets can be omitted or assigned weight zero without changing the heuristic.

Thus, all consistent features F' have dimension of at most $d + 1$, and only those contribute to the heuristic. The upper bound is reached exactly when $\tilde{F}$ introduces a new variable not present in any $F \in \mathcal{F}$. □

To ensure that the heuristic values remain positive throughout the whole plan, we introduce an offset heuristic, called h_{offset}, while the tower has not yet reached its goal height. Once the tower is complete, this value is removed, effectively resetting the scale and guiding the agent towards deconstructing the ramp. It is defined over the weight function

$$w_{\text{offset}}(\{\langle \mathit{field}_m, h_m \rangle\} = \mathit{offset}$$

for all $h_m < n$. The *offset* accounts for the maximum possible cost incurred during the transition from the construction phase to the required deconstruction phase. This transition models a worst-case configuration in which the heuristic must overestimate any potential gain from placing the final block on the tower, thereby completing the construction phase. By doing so,

it ensures that the heuristic continues to guide the agent correctly into the necessary deconstruction phase.
To conservatively estimate this transition cost, we assume the following worst-case configuration:

- All fields $j \in \{0, \dots, m\}$ are at the height $h_j = n - 1$, with each local moving heuristic pulling the robot towards the first field weighted like the last building step,
- The robot is positioned at field $m - 1$ with empty hands.

Under this assumptions, the *offset* is defined as:

$$\begin{aligned} \mathit{offset} = 1 &+ \mathit{walking\_back} \cdot (m-1) \\ &+ m \cdot \mathit{placing\_block} \cdot (n-1) \\ &+ m \cdot 2^{|\mathcal{B}|} \cdot (m-1) \end{aligned}$$

Combining the components introduced above, we define the final heuristic h as a composition of several potential heuristics modulated by feature multiplication. Formally, the heuristic is given by

$$h = h_1 + \langle \mathit{field}_m, n \rangle \circledast h_3 + h_{\text{offset}} + \sum_{h_m=0}^{n-1} \langle \mathit{field}_m, h_m \rangle \circledast h_2$$

We can now establish an upper bound on the ∞-DDA correlation complexity for Linear Termes.

Lemmas 30. *The ∞-DDA correlation complexity of Linear Termes is at most 3.*

Proof. We prove that the heuristic function h satisfies the ∞-DDA properties and that its maximum feature dimension is bounded by 3.

The heuristic is defined as

$$h = h_1 + \langle \mathit{field}_m, n \rangle \circledast h_3 + h_{\text{offset}} + \sum_{h_m=0}^{n-1} \langle \mathit{field}_m, h_m \rangle \circledast h_2.$$

The component h_1, encoding hard constraints, is defined over features of at most two variables and therefore has dimension 2. h_{offset} is defined over just the last fields height, it therefore has dimension 1. Both h_2 and h_3 are potential heuristics composed of local movement features, each defined over exactly two variables: the robot's position and the height of a field. For example, h_3 includes weights of the form

$$w^{\downarrow}_{j,h_j}\left(\{\langle \mathit{robot}, i \rangle, \langle \mathit{field}_j, h_j \rangle\}\right) = |(j-1) - i|,$$

defined for all $j \in \{0, \dots, m-1\}$. These features do not involve the final field m. When feature multiplication is applied with $\langle \mathit{field}_m, h_m \rangle$, the resulting conjunctions introduce a third variable, the height of field m, which was not previously present in any feature.

By Lemma 29, multiplying a heuristic of dimension 2 with a feature over a new variable yields a heuristic of dimension exactly 3. Consequently, both feature-multiplied components

($\langle \mathit{field}_m, n\rangle \circledast h_3$ and $\langle \mathit{field}_m, h_m\rangle \circledast h_2$) have dimension 3. Since h_1 remains of dimension 2 and h_{offset} remains of dimension 1, the overall heuristic h has maximum feature dimension 3.

We now proceed to show that h satisfies the ∞-DDA property.

The initial state has a finite value, i.e. $I \in S_{\mathbf{fin}}$:
Let I denote the initial state, where all fields have height zero, the robot is on field 0, and its hand is empty. We must show that $h(I) < \infty$. Since all features in h_{offset}, h_2 and h_3 have finite weights by definition, it suffices to verify that no features from h_1, which can contribute an infinite value, are present in the initial state.

The weights in w_1 represent structural constraints. These include conditions such as preventing field heights from exceeding their position index, ensuring that field heights do not decrease from left to right, and prohibiting the robot from standing on the final field, height differences between neighbouring fields of more than one, and placing blocks on a layer, when the layer beneath is not completed yet. All of these constraints are trivially satisfied in the initial state: all field heights are zero, and the robot is located at the first field. Therefore, no feature from h_1 applies, and $h(I) < \infty$.

For all non-goal states with a finite heuristic value there exists a successor with a lower heuristic value, i.e. $\forall s \in (S_{\mathbf{fin}} \setminus G)\exists t \in \mathbf{succ}(s) : h(t) < h(s)$:
Due to the feature multiplication in the heuristic definition, and the way w_{offset} is defined for all $h_m < n$, only the components h_1, h_2, and h_{offset} contribute to the overall heuristic value h during the construction phase (i.e. when $h_m < n$). Consequently, to verify the descending and dead-end-avoiding properties, it suffices to consider these components. In the deconstruction phase (when $h_m = n$), the heuristic value of h depends solely on h_1 and h_3.
As established in Lemma 25 to 28, for every non-goal state in the construction phase (i.e., for all $h_m < n$ with finite h_1), there exists a successor state in which h_2 strictly decreases. Likewise, Lemma 34 to 37 demonstrate that in the deconstruction phase, for every non-goal state with finite h_1, there exists a successor in which h_3 strictly decreases.

It remains to show that placing the final block on field m results in a decrease of the heuristic value:

$$h(s') - h(s) = \langle \mathit{field}_m, n\rangle \circledast h_3(s') - \langle \mathit{field}_m, n-1\rangle \circledast h_2(s) - h_{\text{offset}}(s)$$

By construction, *offset* is chosen such that

$$\langle \mathit{field}_m, n\rangle \circledast h_3(s') < \mathit{offset}$$

i.e. the heuristic value in the deconstruction phase is strictly smaller than the contribution from the *offset* alone.
Moreover, since $h_2(s) \geq 0$ by non-negativity of the potential heuristic components, we conclude:

$$h(s') < h(s)$$

Hence, placing the final block results in a strict decrease in the heuristic value.
This completes the proof that the heuristic satisfies the ∞-DDA property.

□

The previous lemma demonstrates that the ∞-DDA correlation complexity of Linear Termes is at most 3. According to the structure illustrated in Figure 4.4, the PDDA correlation complexity is never greater than that of ∞-DDA. Therefore, the upper bound of 3 established for ∞-DDA also holds for the PDDA correlation complexity.

Corollary 30.1. *The PDDA correlation complexity of Linear Termes is at most 3.*

8.4 Discussion and Extensions

This section presents two developments that refine and extend the original framework. First, we replace the exponential weight function with a recursive alternative that preserves the heuristic's intended prioritization while reducing numerical growth. Second, we reconnect the Linear Termes approach with the original two-dimensional Termes system by introducing a fixed build path that linearizes the 2D workspace. In doing so, the 1D heuristic retains its structural properties even when applied in a richer two-dimensional context.

Recursive Weight Matrix

Since the weights $2^{f(j,h_j)}$ grow too quickly for practical computation, we have instead worked with alternative weights defined recursively. This substitution provides a more manageable growth rate while preserving the desired prioritization of tasks in the construction sequence. We achieve this behaviour with the following recursive definition:

$$a_{i,h} = \begin{cases} 1 & \text{if } i = 0 \text{ and } h = 0 \\ \sum_{j>i} a_{j,h} + \sum_{j<i} a_{j,h-1} + \sum_{j<h-1} a_{j,j} + 1 & \text{if } i \geq h \text{ and } h > 0 \\ 0 & \text{otherwise} \end{cases}$$

This recursion strategically prioritizes the current entry $a_{i,h}$ over three distinct groups of other entries, ensuring the robot consistently follows the intended construction logic:

- The first sum ensures that fields to the right with the same height are outweighed by $a_{i,h}$, encouraging the robot to complete the current layer from back to front.
- The second sum ensures that fields to the left at height $h-1$ are outweighed by $a_{i,h}$, reflecting that the lower layer has already been completed and construction should now proceed upward.
- The third sum ensures that diagonal entries from previously completed layers, those that guide the robot towards initiating the next layer, are also surpassed by $a_{i,h}$, so that higher layers are only prioritized after the current one is fully built.

To illustrate the difference in growth, consider the following example:

Example 3 (Grow rate of different weights)**.** Consider a Linear Termes planning task Π with $m+1=10$ fields and a goal height of $n=9$.
Using the exponential weighting scheme it becomes:

$$2^{f(9,9)} = 3.52 \times 10^{13}$$

By contrast, under our recursively defined weights, the final weight value is:

$$a_{9,9} = 1.04 \times 10^{13}$$

The difference between these two values is:

$$2^{f(9,9)} - a_{9,9} = 2.48 \times 10^{13}$$

This substantial gap shows that the recursive weights grow more slowly and are therefore more manageable numerically.

We show in Appendix C that these recursively defined weights still fulfil the necessary heuristic descent properties. The adapted proof follows the same core ideas as the one used for the exponential weights, confirming that the robot continues to make progress towards the goal.
The required adjustments to parameters such as *walking_back*, *placing_block* and *offset* are redefined analogously using the recursive weights as follows:

$$\mathit{walking\_back} = 1 + \sum_{i=0}^{n} a_{i,i}, \qquad \mathit{placing\_block} = \mathit{walking\_back} \cdot m$$

To conservatively estimate *offset*, we assume the following worst-case configuration:

- All fields $j \in \{0, \ldots, m-1\}$ are at their maximum possible height $h_j = \max(j, n)$,
- The robot is positioned at field $m-1$ with empty hands.

Under this assumptions, the *offset* is defined as:

$$\begin{aligned}
\mathit{offset} = 1 &+ \mathit{walking_back} \cdot (m-1) \\
&+ \sum_{j=0}^{m-1} \mathit{placing_block} \cdot [j < n] \cdot j \\
&+ \sum_{j=0}^{m-1} \mathit{placing_block} \cdot [j \geq n] \cdot (n-1) \\
&+ \sum_{j=0}^{m-1} 2^{f(j,j)} \cdot [j < n] \cdot |(j-1) - (m-1)| \\
&+ \sum_{j=0}^{m-1} 2^{f(j,n-1)} \cdot [j \geq n] \cdot |(j-1) - (m-1)|
\end{aligned}$$

These updated definitions ensure that the intended behaviour, returning to the depot when empty-handed and preferring to place blocks over walking, remains intact under the new weighting scheme.

Generalization to 2D

The Linear Termes approach can be naturally extended from 1D to 2D environments where a single tower must be built at a designated goal field. In this setting, the 2D construction area is represented as an $m \times k$ grid, with a fixed path predefined from the depot to the target tower. This path captures the desired construction sequence and serves as a linearization of the 2D problem. Each field along the path is assigned a unique index from 1 to n, where $n < m \cdot k$ is the maximum tower height. These indices then replace the role of the 1D field positions in the original heuristic framework. This approach enables the same layer of control over build order and path planning in two dimensions, while preserving the simplicity and provable properties of the 1D method. Moreover, this approach generalizes to environments of arbitrary shape, provided there exists a path of length n that defines the build order.

9
Experiments

This chapter presents an empirical validation of the theoretical properties of the heuristics developed in this work, specifically those satisfying the UDDA, ∞-DDA, and PDDA conditions. The goal is not to evaluate the heuristics performance or compare them to alternative methods, but to verify that their practical behaviour aligns with theoretical guarantees. Because these heuristics are designed to be strictly descending, ensuring that each state has a successor with a lower heuristic value, solution discovery becomes trivial when a solution exists. Consequently, conventional performance metrics such as search time or plan optimality are not informative in this context.

All experiments were performed using a modified version of Fast Downward [5], extended to support domain-specific multi-dimensional potential heuristics. The modifications affected both the heuristic evaluation and the search infrastructure. Experiments were run on a machine with an Intel(R) Core(TM) i7-8550U CPU @ 1.80 GHz, 16.0 GB RAM, under Ubuntu 24.04. The hardware and software setup was kept constant throughout.

Our validation strategy consisted of two components:

a) **Search behaviour along the solution path:** We ran an eager greedy best-first search using our heuristics as the evaluation function. For each problem instance, we recorded two key metrics: the number of expanded states and the length of the plan. According to our theoretical results, if a heuristic satisfies the desired properties and correctly guides the search, the number of expanded states should match the plan length plus one, accounting for the initial state and each action along the path.

b) **Property verification over the full state space:** We implemented an exhaustive algorithm to verify whether the heuristic satisfies the formal criteria for UDDA and ∞-DDA. This algorithm systematically explores the state space by enumerating all possible variable assignments, visiting all reachable and unreachable states. To test the UDDA property, we check whether every state has at least one successor with a strictly lower heuristic value. For verifying the ∞-DDA property, we restrict this condition to states with finite heuristic

values. If this condition is met for all such states, the corresponding property is considered satisfied.

Note that in cases where the ∞-DDA and PDDA correlation complexities were the same, we only tested the ∞-DDA heuristic. This is because the PDDA heuristic can be directly transformed from the ∞-DDA heuristic without altering its behaviour. For domains where the correlation complexities differed, we tested both heuristics independently.

Blocksworld (IPC 2000):
For our experiments in the *Blocksworld* domain, we used the version provided in the AI-Basel Downward benchmark collection[2]. To simplify the structure and reduce potential ambiguity, we adapted the domain by removing the robotic hand component, resulting in a single-arm variant where blocks can be picked up and placed without intermediate holding actions. In all tested instances, both the ∞-DDA and PDDA heuristics resulted in search behaviour consistent with theoretical expectations: the number of expanded states exactly matched the plan length plus one, indicating strictly descending progress along the solution path with no detours.

For full state space verification, we selected small instances from the *Blocksworld* domain to allow for exhaustive analysis. In every tested instance, the heuristic satisfied the ∞-DDA property. For the PDDA heuristic, the verification was not completed due to insufficient time.

Spanner (IPC 2011 Learning Dataset):
To evaluate search behaviour in the *Spanner* domain, we selected problem instances from the IPC 2011 learning track. In all tested instances, the number of expanded states matched the plan length plus one, exactly as predicted by our theoretical analysis, confirming that the search followed the path without unnecessary exploration.

For full state space verification, we generated custom *Spanner* instances with reduced state space sizes, allowing exhaustive analysis. In every tested instance, the heuristic satisfied the ∞-DDA property. This confirms that the heuristic maintained the required descending behaviour across the entire space.

Logistics (IPC 1998):
To evaluate our approach in the *Logistics* domain, we used problem instances from the IPC 1998 benchmark set provided in the AI-Basel Downward benchmark collection[2]. In all tested instances, the number of expanded states exactly matched the plan length plus one, demonstrating that the heuristic consistently guided search along a path without detours or backtracking.

To enable complete state space verification, we tested on a custom small *Logistics* instance. In this reduced setting, the heuristic satisfied the UDDA property. This confirms that the

[2] https://github.com/aibasel/downward-benchmarks

heuristic consistently guided progress towards the goal and avoided dead ends across the entire domain.

Termes (IPC 2018):
In the Termes domain, we evaluated our heuristic on custom instances of the linear variant introduced earlier. Because the heuristic assigns rapidly increasing weights, the state space size grows with the grid size, even in relatively small instances. This growth restricts exhaustive analysis to only the smaller problem configurations.

Despite this limitation, in all tractable instances we tested, the heuristic satisfied the ∞-DDA property. This confirms that the heuristic maintained the desired descending behaviour across the finite space. Moreover, the number of expanded states during the search matched the plan length plus one, indicating that the search followed a path without detours or unnecessary exploration.

10 Conclusion

In this thesis, we have analysed the hierarchy and correlation complexities of four DDA heuristic variants, DDA, ∞-DDA, UDDA, and PDDA, across several planning domains: Blocksworld, Spanner, Logistics, and a modified version of Termes. Our results demonstrate that the relationship between these heuristics varies depending on domain structure.

Among the heuristic variants studied, UDDA is the most general: every UDDA heuristic can be transformed into an equivalent DDA, ∞-DDA, or PDDA heuristic, implying that their correlation complexities are always bounded above by that of UDDA. Notably, through concrete examples such as the modified Gray code, we demonstrated that the more specialized variants (DDA, ∞-DDA, and PDDA) can achieve strictly lower correlation complexity than UDDA in certain planning tasks. Furthermore, every ∞-DDA heuristic can be transformed into a PDDA heuristic. However, DDA is incomparable with both ∞-DDA and PDDA, reflecting fundamental structural differences between these variants.

These theoretical insights were then supported by empirical evaluations across domains. In Blocksworld, we observed that ∞-DDA requires a correlation complexity of 3, while DDA and PDDA achieve a lower complexity of 2. This provides a concrete, practical example where one heuristic variant (∞-DDA) relies on strictly more information than the others (DDA and PDDA) to resolve the same planning problem. This distinction highlights a meaningful difference in how these heuristics interpret and utilize structural information in a domain.

In the Spanner domain, we showed that no UDDA heuristic exists. Nevertheless, DDA, ∞-DDA, and PDDA all attain the same correlation complexity of 2, consistent with previous results for DDA [14]. For the Logistics domain, all four variants yield a correlation complexity of 2, indicating that the choice among these DDA heuristics becomes inconsequential in structurally simpler domains.

In the modified Termes domain, we established a lower bound of 2 and an upper bound of 3 for both ∞-DDA and PDDA, with no exact value yet determined. Preliminary attempts to prove a strict lower bound of 3 using MiniZinc were inconclusive due to time constraints and will be

pursued as future work. The open nature of this result suggests potential for further insight into the complexity behaviour of these heuristics in spatial and construction-based domains.

Finally, we implemented each of the heuristic variants across the studied domains and verified their satisfaction of the different DDA properties. These implementations served to validate our theoretical hierarchy and complexity results. For all variants except PDDA, we conducted both a search-based evaluation and a full verification of the corresponding property over the entire state space. For PDDA, due to time constraints, we limited our evaluation to search-based behaviour along solution paths.

Future Work

There are several directions in which this work could be extended. One important aspect is the investigation of the lower bound of the correlation complexity in the linear Termes domain. While we have established an upper bound, determining whether this bound is tight would provide a more complete understanding of the informational requirements in this domain.

Additionally, it would be valuable to generalize the heuristic developed for the linear version of Termes to the full Termes domain. Such a generalization would enable a more comprehensive evaluation of the heuristic's applicability, and could offer insights into how increasing structural complexity within the domain affects the correlation complexity required by different heuristic variants.

Finally, investigating a wider range of planning domains could clarify whether the observed differences in correlation complexity among DDA variants are specific to certain domain properties or occur more broadly. Such an expanded study would enhance our understanding of the relationship between domain structure and heuristic behaviour.

Bibliography

[1] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11:625–656, 1995.

[2] Simon Dold and Malte Helmert. Higher-dimensional potential heuristics: lower bound criterion and connection to correlation complexity. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 151–161, 2024.

[3] Guillem Francès Medina, Augusto B Corrêa, Cedric Geissmann, and Florian Pommerening. Generalized potential heuristics for classical planning. In *International Joint Conferences on Artificial Intelligence*, 2019.

[4] Frank Gray. Pulse code communication. US Patent 2632058, March 1953.

[5] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[6] Malte Helmert, Silvan Sievers, Alexander Rovner, and Augusto B. Corrêa. On the complexity of heuristic synthesis for satisficing classical planning: Potential heuristics and beyond. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling*, pages 124–133, 2022.

[7] Sven Koenig and T.K. Satish Kumar. A case for collaborative construction as testbed for cooperative multi-agent planning. In *Scheduling and Planning Applications woRKshop (SPARK)*, page 10, 2017.

[8] Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[9] Drew M McDermott. The 1998 AI planning systems competition. *AI magazine*, 21(2): 35–35, 2000.

[10] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[11] Kirstin Petersen, Radhika Nagpal, and Justin Werfel. Termes: An autonomous robotic system for three-dimensional collective construction. In *Robotics: science and systems VII*, volume 27, pages 257–265. MIT press Cambridge, CA, USA, 2011.

[12] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 3335 – 3341, 2015.

[13] Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the Sixth European Conference on Planning*, pages 174–182, 2001.

[14] Jendrik Seipp, Florian Pommerening, Gabriele Röger, and Malte Helmert. Correlation complexity of classical planning domains. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 3242–3250, 2016.

[15] John Slaney and Sylvie Thiébaux. Blocks world revisited. In *Artificial Intelligence*, volume 125, pages 119–153, 2001.

[16] Lin Zhu and Robert Givan. Landmark extraction via planning graph propagation. In *ICAPS Doctoral Consortium*, pages 156–160, 2003.

A

MiniZinc Model Code

```
include "state_data.dzn";

int: num_fields = 5;
int: max_height = 3;
set of int: FIELDS = 1..num_fields;
set of int: HEIGHTS = 0..max_height;
set of int: HAND = 0..1;

array[FIELDS] of var HEIGHTS: heights;
var FIELDS: robot_pos;
var HAND: carrying;

int: num_vars = num_fields + 2;
int: num_states = (max_height+1)^num_fields * num_fields * 2;
int: num_atoms = num_fields*(max_height+1) + num_fields + 2;
set of int: ATOMS = 1..num_atoms;
set of int: STATES = 1..num_states;

int: MAX_W = 10;
array[ATOMS,ATOMS] of var 0.0..MAX_W: w;
array[ATOMS, ATOMS] of var 0..1: w_inf;

array[STATES,ATOMS] of int: features;
array[STATES] of var float: h;
array[STATES] of var int: h_inf;
array[STATES] of int: goal;
array[STATES, STATES] of int: succ;
```

```
% heuristic constraint
constraint
  forall(s in STATES)(
    h[s] =
      sum(a in ATOMS)(w[a,a] * features[s,a]) +
      sum(a,b in ATOMS where a < b)
      (w[a,b] * features[s,a] * features[s,b])
  );

% inf propagation: if any pair with infinite weight
% is active in state s, then inf[s] > 0
constraint
  forall(s in STATES)(
    h_inf[s] =
      sum(a in ATOMS)(w_inf[a,a] * features[s,a]) +
      sum(a,b in ATOMS where a < b)
      (w_inf[a,b] * features[s,a] * features[s,b])
  );

constraint
  forall(a, b in ATOMS)(
    w_inf[a,b] == 1 -> w[a,b] == 0
  );

% initial state
constraint h_inf[1]==0;

% infinity dda property
constraint
  forall(s in STATES)(
    h_inf[s]>=1 \/
    goal[s]==1 \/
    exists(s2 in STATES)(
    succ[s,s2]==1 /\
    h_inf[s2]==0 /\
    h[s2] < h[s]
    ));

solve satisfy;
```

B

Linear Termes: Destruction Phase

The third subproblem addresses the deconstruction of the ramp. The destruction phase mirrors the construction phase but proceeds in reverse: blocks are removed rather than placed. Each field i (except the last) encourages the robot to move to field $i-1$, from where it can pick up a block at i. The same parameters and heuristic structures are reused, with only directional logic inverted.

Moving Heuristic

As before, we break down the global objective into local goals.

Local Moving Heuristic

We define a family of local heuristics $\{h^{\downarrow}_{j,h_j}\}$ for all $j = 0, \ldots, m-1$ and $h_j = 0, \ldots, n$, just as in construction. The key difference is that each field j now encourages the robot to move to $j-1$ (rather than $j+1$), enabling the robot to pick up blocks in reverse order. The corresponding weight function is:

$$w^{\downarrow}_{j,h_j}\left(\{\langle \mathit{robot}, i\rangle, \langle \mathit{field}_j, h_j\rangle\}\right) = |(j-1) - i|.$$

As in the construction phase, these weights satisfy the property from Lemma 20.

Returning to the depot

The behaviour when carrying a block remains the same: the robot should return to the depot at field 0. We reuse the same depot-directed heuristic $h^{\downarrow}_{\text{depot}}$, which is now defined over:

$$w^{\downarrow}_{\text{depot}}(\{\langle \mathit{robot}, i\rangle, \langle \mathit{hand}, \mathit{block}\rangle\}) = i.$$

This again satisfies the conditions in Lemma 20 since $i = |0 - i|$ reflects the distance to the depot.

Global Weighted Moving Heuristic

Since the ramp is deconstructed in exactly the reverse order of its construction, we can reuse the same weight structure for the local movement heuristics. This leads to the global moving heuristic:

$$h^{\downarrow}_{\text{walk}} = \mathit{walking\_back} \cdot h^{\downarrow}_{\text{depot}} + \sum_{i=0}^{m} \sum_{h=0}^{n} 2^{f(i,h)} \cdot h^{\downarrow}_{i,h} \tag{1}$$

To ensure progress during deconstruction, we first show that the robot can always move towards the next field where a block should be picked up, and that this move strictly decreases the weighted moving heuristic without violating any constraints.

Lemmas 31. *Let s be a non-goal state with a finite h_1 heuristic value. Let $h^{\downarrow}_{walk}$ be the weighted moving heuristic defined by the weight function in Eq.* (1)*.*
Assume the robot is not carrying a block. Then there exists a field p such that the next deconstruction step is to pick up a block from field $p+1$, and moving towards field p is a feasible action that does not incur an infinite penalty from h_1. Moreover, this movement results in a strictly decreased heuristic value $h^{\downarrow}_{walk}$ in the resulting state s'.

Proof. Since s is not a goal state and $h_1(s) < \infty$, some portion of the ramp remains to be deconstructed. Let j denote the index of the last completed construction step. Then all steps with index $j' > j$ have already been undone. By definition, the next block to remove corresponds to construction step j. Let $f^{-1}(j) = (i, h_i)$, and set $p := j - 1$. Then the next pickup is from field $p + 1$.

Because the robot is not carrying a block, the depot term in $h^{\downarrow}_{\text{walk}}$ is unchanged. By Lemma 19, moving to p is feasible and does not violate the constraints of h_1. By Lemma 20, this move decreases the j-th local term by 1 and may increase all earlier ones by 1. Thus:

$$h^{\downarrow}_{\text{walk}}(s') - h^{\downarrow}_{\text{walk}}(s) = -2^j + \sum_{k=0}^{j-1} 2^k = -2^j + (2^j - 1) = -1 < 0.$$

Therefore, the heuristic strictly decreases after moving towards p, as required. □

Building on the guaranteed progress from moving towards the pickup field, we now establish an upper bound on the heuristic increase caused by actually picking up the block.

Lemmas 32. *Let s be a non-goal state with a finite h_1 heuristic value. Let $h^{\downarrow}_{walk}$ be the weighted moving heuristic.*
If the robot is empty-handed, it is positioned on field p, then picking up a block from field $p+1$ is a valid action that preserves the finiteness of h_1, and the resulting state s' satisfies:

$$h^{\downarrow}_{walk}(s') - h^{\downarrow}_{walk}(s) < \mathit{walking\_back} \cdot m.$$

Proof. By Lemma 31, there exists a field p such that picking up a block from $p+1$ corresponds to the next valid deconstruction step. Let $q := p + 1$. Since h_1 assigns infinite cost only to states that violate the intended deconstruction sequence, performing this step ensures that h_1

remains finite.

Picking the block from field q modifies two variable assignments. Specifically, it changes the assignment of the variable *hand* from *free* to *block*, and updates field_q from h_q to $h_q - 1$. These changes affect the heuristic value $h^{\downarrow}_{\text{walk}}$ as follows:

$$\begin{aligned} h^{\downarrow}_{\text{walk}}(s') - h^{\downarrow}_{\text{walk}}(s) &= w^{\downarrow}_{\text{walk}}(\{\langle \mathit{robot}, p\rangle, \langle \mathit{hand}, \mathit{block}\rangle\}) \\ &\quad + w^{\downarrow}_{\text{walk}}(\{\langle \mathit{robot}, p\rangle, \langle \mathit{field}_q, h_q - 1\rangle\}) \\ &\quad - w^{\downarrow}_{\text{walk}}(\{\langle \mathit{robot}, p\rangle, \langle \mathit{field}_q, h_q\rangle\}) \\ &= \mathit{walking\_back} \cdot p + 2^{f(q,h_q-1)} \cdot |p - 1 - p| - 2^{f(q,h_q)} \cdot |p - p| \\ &= \mathit{walking\_back} \cdot p + 2^{f(q,h_q-1)}. \end{aligned}$$

Since by definition

$$\mathit{walking\_back} = 2^{|\mathcal{B}|+1}, \quad \text{and} \quad f(q, h_q - 1) \leq |\mathcal{B}|,$$

we conclude that $2^{f(q,h_q-1)} < \mathit{walking\_back}$. Thus:

$$h^{\downarrow}_{\text{walk}}(s') - h^{\downarrow}_{\text{walk}}(s) < \mathit{walking\_back} \cdot (p + 1).$$

Since $p \leq m - 1$, we have:

$$h^{\downarrow}_{\text{walk}}(s') - h^{\downarrow}_{\text{walk}}(s) < \mathit{walking\_back} \cdot m.$$

□

Block Pickup Heuristic

To mirror the construction phase, we define a block pickup heuristic that measures remaining deconstruction work. Since the goal height for each field is 0, each field j contributes h_j to the heuristic:

$$w^{\downarrow}_{\text{block}}(\{\langle \mathit{field}_j, h_j\rangle\}) = h_j$$

for all $j = 0, \ldots, m - 1$ and $h_j = 0, \ldots, j$.
As with construction, we now show that robot movement alone does not affect this heuristic, while picking up a block decreases it.

Lemmas 33. *Let s be a non-goal state with a finite h_1 heuristic value. Then:*

(i) Any action in which the robot moves between fields does not change the value of $h^{\downarrow}_{\mathit{block}}$.

(ii) Picking up a block reduces the heuristic value by exactly 1.

Proof. (i) The weight function depends only on field heights, not on the robot's position. Therefore, pure movement does not affect $h^{\downarrow}_{\text{block}}$.

(ii) Picking up a block reduces h_j by 1. The resulting change in the heuristic is:

$$h^{\downarrow}_{\text{block}}(s') - h^{\downarrow}_{\text{block}}(s) = (h_j - 1) - h_j = -1.$$

□

As in the construction phase, to ensure that picking up a block remains favourable despite an increase in movement cost, we scale the heuristic:

$$\mathit{picking\_block} = \mathit{walking\_back} \cdot (m+1)$$

Finally, we define the weighted block pickup heuristic:

$$h^{\downarrow}_{\text{picking}} = \mathit{picking\_block} \cdot h^{\downarrow}_{\text{block}}$$

Destruction Heuristic

We can now introduce the main destruction heuristic

$$h_3 = h^{\downarrow}_{\text{picking}} + h^{\downarrow}_{\text{walk}} = \mathit{picking\_block} \cdot h^{\downarrow}_{\text{block}} + \mathit{walking\_back} \cdot h^{\downarrow}_{\text{depot}} + \sum_{i=0}^{m} \sum_{h=0}^{n} 2^{f(i,h)} \cdot h^{\downarrow}_{i,h}.$$

Analysis of Successor States

We now analyse the behaviour of the heuristic in states where the robot is either carrying a block or not. For every possible configuration, we prove that there exists a successor state whose heuristic value is strictly lower than that of the current state. Our approach involves a thorough case-by-case analysis, distinguishing scenarios based on the robot's position as well as whether it is carrying a block or not.

Before delving into these cases, we first establish the existence of a particular field p that the robot is always encouraged to move towards when not carrying a block. When the robot occupies field p, it is guaranteed that grabbing a block from the adjacent field $p+1$ is possible. Importantly, both moving towards p and grabbing a block from $p+1$ avoid adding an infinite weight to the heuristic value.

Case 1: The robot is carrying a block and is positioned at *field* i with $i \in \{1, \ldots, m-1\}$

First, we show that when the robot is carrying a block and is positioned on a field other than the depot, the robot is encouraged to move to the depot to hand over the block.

Lemmas 34. *Let s be a state in which the robot is carrying a block and is located at field i with $i \in \{2, \ldots, m-1\}$. Then, there exists a successor state s' of s such that $h_3(s') < h_3(s)$.*

Proof. By Lemma 19, walking to a neighbouring field, excluding the final field, is always feasible in states with finite h_1, and such a move does not introduce an infinite penalty in the heuristic.

Since s is not a goal state, at least one field is incomplete. Moreover, $h_1(s) < \infty$ implies that all building steps up to some $j \leq |\mathcal{B}|$ are complete. Let this j be the index of the last completed building step.
Consider the worst case where moving towards the depot increases the heuristic values for all $k \leq j$ by $+1$ and decreases the heuristic value corresponding to $\langle \mathit{hand}, \mathit{block} \rangle$ by -1. The change

in the heuristic is then:

$$h_3(s') - h_3(s) = -\mathit{walking\_back} + \sum_{k=0}^{j} 2^k$$

Recall the definition of *walking_back*:

$$\mathit{walking\_back} = 2^{|\mathcal{B}|+1}$$

By definition $j < |\mathcal{B}| + 1$, this concludes

$$h_3(s') - h_3(s) = -2^{|\mathcal{B}|+1} + \sum_{k=0}^{j} 2^k = -2^{|\mathcal{B}|+1} + 2^{j+1} - 1 < 0$$

That is, the heuristic strictly decreases when the robot walks one step towards the depot carrying a block. □

Case 2: The robot is carrying a block and is positioned at *field* 0

Next, we handle the situation where the robot is carrying a block and is positioned at *field* 0. We show that handing over the block leads to a successor state with a strictly lower heuristic value.

Lemmas 35. *Let s be a state in which the robot is carrying a block and is located at field 0. Then, there exists a successor state s' of s such that $h(s') < h(s)$.*

Proof. The robot can always hand over a block at the depot, since its height remains 0. This action updates the variable *hand* from *block* to *empty*. As *hand* does not occur in any feature of w_1, we focus on w_3. The only weight in w_3 involving *hand* is $w^{\downarrow}_{\text{depot}}$, which becomes inactive because the atom $\langle \mathit{hand}, \mathit{block} \rangle$ is removed from the successor state s'. As a result, the heuristic value decreases by *walking_back*. □

Case 3: The robot is not carrying a block and is on the target field p

Next, we handle the situation where the robot is not carrying a block and is positioned at field p.

Lemmas 36. *Let s be a state in which the robot is not carrying a block and is located at field p. Then, there exists a successor state s' of s such that $h(s') < h(s)$.*

Proof. By Lemma 31, there is a field p such that picking up a block at $q := p + 1$ is the next valid deconstruction step, ensuring $h_1(s') < \infty$.

Picking up a block decreases h_{block} by 1 (Lemma 33) and increases $h^{\downarrow}_{\text{walk}}$ by less than $m \cdot$ *walking_back* (Lemma 32), so:

$$h_3(s') - h_3(s) < -\mathit{picking\_block} + m \cdot \mathit{walking\_back}.$$

Since $\mathit{picking\_block} = \mathit{walking\_back} \cdot (m + 1)$, this yields

$$h_3(s') - h_3(s) < -\mathit{walking\_back} < 0.$$

With h_1 unchanged and finite, we conclude

$$h_3(s') < h_3(s),$$

as required. □

Case 4: The robot is not carrying a block but is at a different field *field* i, with $i \neq p$

Next, we handle the situation where the robot is not carrying a block and is positioned at *field* i, with $i \neq p$.

Lemmas 37. *Let s be a state in which the robot is not carrying a block and is located at field i with $i \in \{0, \ldots, m-1\}$. Then, there exists a successor state s' of s such that $h_3(s') < h_3(s)$.*

Proof. By Lemma 19, taking a walking step to a neighbouring field (excluding the final one) is always feasible in any state where $h_1 < \infty$, and this step does not introduce an infinite penalty. Thus, we focus on the change in the h_2 component, specifically the weighted moving heuristic $h^{\downarrow}_{\text{walk}}$, which depends on the robot's position.

Since $\mathit{field}_i \neq \mathit{field}_p$, the robot is not yet at its preferred destination field p. According to Lemma 31, taking a step towards field p decreases the heuristic value. Therefore, by executing such a move, the robot reaches a successor state s' with $h_3(s') < h_3(s)$. □

C

Linear Termes: Refined Proof Using Recursive Weights

In this appendix, we outline the necessary modifications to Lemma 21 to incorporate the recursive definition of the weights $a_{i,h}$.
First, we replace the statement on the weighted walking heuristic in the lemma by:

Let $h^{\uparrow}_{walk}$ be the weighted walking heuristic defined using the recursive weights $a_{i,h}$.
Next, update the argument about the heuristic difference in the proof as follows:
Moving one step towards field p decreases the walking distance for (i, h) by 1, which reduces the heuristic by exactly $a_{i,h}$. Simultaneously, this move may increase the walking distances for earlier building steps. However, only those building steps, whose associated field–height pair feature is active in s, contribute to the heuristic.
Given the recursive definition of $a_{i,h}$,

$$a_{i,h} = \sum_{j>i} a_{j,h} + \sum_{j<i} a_{j,h-1} + \sum_{j<h-1} a_{j,j} + 1$$

we know that $a_{i,h}$ is strictly larger than the total weight of all unfinished building steps from previous layers that could possibly increase during this move. That is, it outweighs every earlier term in the heuristic whose associated field–height pair feature is active in s.
Therefore, the net change in the heuristic is strictly negative:

$$h^{\uparrow}_{\text{walk}}(s') - h^{\uparrow}_{\text{walk}}(s) < 0$$

These adjustments ensure that the proof correctly reflects the weighting scheme described by the recursive definition.