Predicting Macro-Learning Performance Using Structural Regularity

Anton Gustafsson, Jendrik Seipp, Elliot Gestrin

Department of Computer and Information Science
Linköping University, Sweden
antgu175@student.liu.se, jendrik.seipp@liu.se, elliot.gestrin@liu.se

Abstract

Macro-learning is a technique in automated planning in which action sequences are compiled into higher-level actions, or macros. While it can improve planning performance by reducing search depth, its effectiveness is highly domain-dependent and it can even degrade performance. This makes it important to determine when macro-learning is beneficial. We introduce *structural regularity*, a metric for predicting macro-learning performance by capturing recurring structural patterns in plans through repeated action subsequences and parameter reuse, without relying on object identities. As part of this work, we compile and label a curated dataset of 47 planning domains from 10 macro-learning studies, comprising 84 data points across different planners and configurations based on observed macro-learning impact. We evaluate structural regularity on this dataset using leave-one-out cross-validation. Our metric fits the training data well, but shows limited generalization to unseen domains, with performance above a uniform random baseline but below a majority-class baseline, the latter partly driven by class imbalance in the dataset. Our results suggest that structural regularity is a step toward predicting when macro-learning should be applied.

1 Introduction

Macro-learning is a well-established technique in automated planning, where frequently occurring action sequences are compiled into higher-level actions, or macros (e.g., Botea et al. 2005; Coles and Smith 2007; Asai and Fukunaga 2015). When successful, macros can significantly reduce search depth and improve planning efficiency. However, their effectiveness is highly domain-dependent. In some domains, macro-learning leads to substantial speedups, while in others it introduces overhead by increasing the branching factor and the size of grounded representations, ultimately degrading performance (Alarnaouti, Baryannis, and Vallati 2023).

This variability presents a fundamental challenge: we currently lack reliable and widely applicable methods for predicting when macro-learning will be beneficial. As a result, applying macro-learning often involves a costly trial-and-error process in which computational resources are spent learning and evaluating macros that may ultimately be ineffective.

In this paper, we address this problem by introducing a predictive metric for macro-learning performance. Our key idea is that macro-learning is effective in domains where plans exhibit recurring structural patterns that can be exploited. We formalize this intuition through *structural regularity*, a metric that quantifies the extent to which action sequences and parameter usage patterns repeat across plans.

A central challenge in capturing such patterns is that object identities differ across problem instances, making direct object-based comparisons infeasible, whereas considering only action names is often too coarse to capture meaningful structure. To address this, our metric focuses on how parameters are reused across actions, capturing partially grounded patterns that remain consistent across instances.

We define structural regularity by combining three components: (i) *parameter flow*, which captures how parameters propagate across action tuples, (ii) a *similarity score* that compares such flows, and (iii) an aggregation scheme that measures how frequently similar structures occur within and across plans. At the domain level, we normalize this measure to account for the number of sampled plans.

To evaluate our approach, we collect data from 10 macro-learning papers and classify 47 planning domains according to whether macro-learning has a positive, neutral, or negative impact, yielding 84 data points across different planners and configurations. We then compute structural regularity for each domain and use leave-one-out cross-validation to tune its parameters and classification thresholds. Our results indicate that structural regularity captures signals correlated with macro-learning performance. While the metric fits the training data well, it generalizes only weakly to unseen domains, achieving classification accuracy above a uniform random baseline but below a majority-class baseline on the test data, the latter being partly driven by class imbalance in the dataset. This highlights both the promise of the metric and the challenges posed by limited data and variability in macro-learning outcomes.

Beyond the metric itself, we contribute this consolidated dataset of macro-learning outcomes across domains, enabling systematic evaluation of predictive approaches in this setting.

2 Background

We consider classical planning, where the environment is fully observable, actions are deterministic, and the objective is to find a sequence of actions that transforms an initial

state into a goal state (Ghallab, Nau, and Traverso 2004). Such planning tasks are typically expressed in the Planning Domain Definition Language (PDDL), which uses a *domain* and a *problem*.

A *domain* is a tuple $\mathcal{D} = \langle P, A \rangle$ where P is a set of lifted Boolean propositional predicates and A is a set of lifted actions. Each *action* is in turn a tuple $a = \langle \textit{name}(a), \textit{par}(a), \textit{pre}(a), \textit{eff}(a) \rangle$, where $\textit{name}(a)$ is a unique identifier and $\textit{par}(a)$ is a tuple of parameters, while $\textit{pre}(a)$ and $\textit{eff}(a)$ are logical formulas over P specifying when an action is applicable and its effect, respectively.

A *problem* is a tuple $\mathcal{P} = \langle \mathcal{D}, O, I, G \rangle$ consisting of the associated domain $\mathcal{D}$, the set of *objects* O, each of which is a unique identifier, the initial state I and a goal condition G. An action $a \in A$ can further be *grounded* by replacing $\textit{par}(a)$ with objects from O.

A *plan* π for a problem $\mathcal{P} = \langle \cdot, \cdot, I, G \rangle$ is a sequence of grounded actions $\pi = \langle a_1, a_2, \ldots, a_N \rangle$ such that a_1 is applicable in I, a_n is applicable in the state after applying a_{n-1} and G holds after a_N.

The *makespan* of a plan is the length thereof when adjacent actions are executed in parallel unless their preconditions or effects clash. We refer to Bäckström (1998) for further details.

In this work, we use PDDL domains and problems to generate plans, but our metric operates solely on the resulting sequences of grounded actions and is largely independent of the underlying representation and the considered fragments.

Macro-learning extends classical planning by introducing *macros*, i.e., sequences of actions treated as single higher-level actions. These are typically learned from previously solved plans and can reduce search depth. However, macros may also increase the branching factor and grounding size, which can degrade performance. As a result, their performance is highly domain-dependent.

3 Related Work

Macro-action learning and related reformulation techniques have been extensively studied (e.g., Armano, Cherchi, and Vargiu 2004; Botea et al. 2005; Coles and Smith 2007; Chrpa 2010; Dulac et al. 2013; Chrpa, Vallati, and McCluskey 2014; Asai and Fukunaga 2015; Chrpa and Siddiqui 2015; Hofmann, Niemueller, and Lakemeyer 2017; Chrpa and Vallati 2019; Castellanos-Paez, Rombourg, and Lalanda 2021; Alarnaouti, Baryannis, and Vallati 2023). These approaches aim to improve planner performance by transforming the representation of planning problems or domains. However, they do not explicitly characterize structural properties of planning domains themselves. In contrast, our work provides a quantitative perspective on domain structure, enabling comparison and categorization based on empirical regularities in macro-learning behavior.

Next, we contrast our structural regularity metric with a subset of the most closely related approaches that extract macros from plans.

Botea et al. (2005) construct macro-actions by identifying sequences of actions that are linked through shared parameters, i.e., actions that refer to the same objects across consecutive steps. These sequences are generalized into macros by preserving the variable mappings between actions. While this captures dependencies between adjacent actions based on shared objects, it is limited to pairwise or local interactions. In contrast, our notion of parameter flow models how parameters are reused across longer action sequences, enabling the detection of higher-order structural patterns that are not restricted to immediate co-occurrence.

Castellanos-Paez, Rombourg, and Lalanda (2021) propose ERA, a sequential pattern mining approach that extracts macro-actions from grounded plans by identifying frequent subsequences with consistent parameter-binding structures. Their method encodes full binding patterns and requires exact matches across occurrences, including cases where non-adjacent actions are permitted via a gap parameter. In contrast, our approach operates on partial parameter-binding structures, enabling the detection of recurring patterns even when full binding configurations differ. This allows capturing more general forms of repetition beyond exact sequence matching.

Dulac et al. (2013) propose an approach based on n-gram analysis of action sequences, where frequent contiguous sequences of actions are generalized into macro-actions. Their method relies on statistical frequency and heuristic filtering to identify useful macros, but requires exact matching of action sequences and does not account for partial similarity in parameter-binding structure. In contrast, our approach defines a similarity-based measure over partial parameter bindings, enabling the detection of recurring structure beyond exact sequence repetition.

To the best of our knowledge, no prior work proposes a metric that measures the degree of structural repetition within a planning domain. Our proposed approach introduces several novel components. First, instead of tracking object identities, it captures how parameter positions propagate across actions, enabling abstraction over specific objects and cross-problem analysis. Second, it defines a similarity measure that allows partial matching of parameter-binding structures rather than requiring exact correspondence. Finally, these similarities are aggregated into a domain-level score that reflects the overall structural regularity of the domain.

4 Structural Regularity

To estimate when macro-learning is beneficial, we develop a metric that estimates the utility of learning macros for a planning domain. Intuitively, if a domain does not contain meaningful macro structure, then macro discovery and application may simply add overhead without improving performance. Our *structural regularity* metric aims to capture macro-relevant structure of arbitrary length by identifying repeating subsequences in and between plans.

A key challenge is that object identities differ across problem instances within the same domain. As a result, objects cannot be directly matched between plans from different problems, making fully object-based comparisons infeasible.

At the same time, considering only action names is often too coarse to capture meaningful structure. For example, in domains with a single action schema, all plans may appear

identical at the level of action names, despite exhibiting different structural patterns in how objects are manipulated.

To enable comparisons across grounded plans from different problem instances, we abstract away object identities and instead track how parameters are reused across actions. Specifically, rather than recording which objects appear in a plan, we record how parameter positions correspond across actions. For example, consider two actions $x(a, b, c)$ and $y(c, b, a)$. Instead of tracking the objects themselves, we record that the first parameter of x corresponds to the third parameter of y, the second to the second, and the third to the first. This yields the index correspondences $\{\langle 1, 3\rangle, \langle 2, 2\rangle, \langle 3, 1\rangle\}$. In this way, each occurrence of parameter reuse is represented as a mapping between parameter indices. This representation is invariant to object identities and allows structural patterns to be compared across different problem instances.

4.1 Problem-Level Structural Regularity.

We first define structural regularity at the problem level to capture repeated patterns within individual plans, before extending it to domain-level aggregation. We do this through three components: parameter flow, similarity score, and aggregation across tuples.

We capture the notion of parameter reuse through the concept of *parameter flow*, which records how parameters are shared between actions within a tuple.

Definition 1. Let $\langle a_1, \ldots, a_n\rangle$ be an action tuple, i.e., a contiguous subsequence of a plan. For each action a_i, let $\mathit{par}(a_i) = \langle \mathit{par}(a_i)_1, \ldots, \mathit{par}(a_i)_{k_i}\rangle$ denote its parameters.

The *parameter flow* of the tuple is defined as $T_{a_1,\ldots,a_n} = (\sigma, F)$, where $\sigma = \langle \mathit{name}(a_1), \ldots, \mathit{name}(a_n)\rangle$ is the sequence of action names, and

$$F = \left\{ \langle i, j\rangle \in \{1, \ldots, k_1\} \times \{1, \ldots, k_n\} \;\middle|\; \mathit{par}(a_1)_i = \mathit{par}(a_n)_j \right\}$$

is the set of indices for all shared parameters between the first and last actions of the tuple.

The parameter flow of an action tuple captures both its sequence of action names and the parameter reuse between the first and last actions. Any change to the actions in the tuple alters σ, making the resulting flows incomparable. Similarly, changes to the parameters of the first or last actions affect F, reflecting different patterns of parameter reuse.

In this sense, σ encodes the structural pattern of actions, while F captures how parameters are propagated across the tuple. This combination captures both the structural pattern of actions and how parameters are reused, enabling comparisons across plans without relying on object identities, which are not directly comparable across different problem instances.

While σ captures the full sequence of action names, F records parameter correspondences only between the first and last actions. This design avoids redundancy: correspondences between contiguous actions are already captured by shorter tuples when aggregating over all tuple lengths. As a result, longer tuples encode only additional, non-local parameter reuse, rather than duplicating structure that has already been accounted for.

Example 1. Let "$-$" denote a parameter that is distinct at each occurrence, i.e., it does not match any other parameter in the plan.

1. $T_{x(a,b),\, y(a,-)} = (\langle x, y\rangle, \{\langle 1, 1\rangle\})$. The first parameter of x reappears as the first parameter of y.
2. $T_{y(a,-),\, z(-,b)} = (\langle y, z\rangle, \emptyset)$. No parameter is shared between the first and last actions.
3. $T_{x(a,b),\, y(a,-),\, z(b,a)} = (\langle x, y, z\rangle, \{\langle 1, 2\rangle, \langle 2, 1\rangle\})$. Both parameters of x reappear in z, but with swapped positions.

To calculate structural regularity for a domain, we compare parameter flows induced by action tuples within plans. However, parameter flows may match only partially, as some parameter correspondences may align while others differ. To account for this, we define a similarity score between two parameter flows that measures the extent of their structural agreement.

Definition 2. Let $T_{a_1,\ldots,a_n} = \langle \sigma, F\rangle$ and $T_{b_1,\ldots,b_m} = \langle \sigma', F'\rangle$ be two parameter flows. Let $\alpha \in [0, 1]$ be a tunable parameter controlling the fraction of parameter correspondences that must match for the similarity score to attain 1. We define their *similarity score* as

$$S_\alpha\left(T_{a_1,\ldots,a_n}, T_{b_1,\ldots,b_m}\right) = \begin{cases} \min\left(1,\ \frac{1+|F\cap F'|}{1+\left\lceil \alpha \cdot \min\left(|\mathit{par}(a_1)|,\ |\mathit{par}(a_n)|\right)\right\rceil}\right) & \text{if } \sigma = \sigma', \\ 0 & \text{otherwise.} \end{cases}$$

If the action sequences differ ($\sigma \neq \sigma'$), we define the similarity score as zero. In the case where the action sequences are identical ($\sigma = \sigma'$), similarity increases with the number of shared parameter correspondences, normalized by the maximum number of correspondences that can be matched, given by $\min(|\mathit{par}(a_1)|, |\mathit{par}(a_n)|)$. This bound can be exceeded only when the same object is reused across multiple parameters within a single action, which is highly unusual in planning.

The normalization term $\min(|\mathit{par}(a_1)|, |\mathit{par}(a_n)|)$ reflects the maximum number of parameter correspondences that can occur between the first and last actions, which is bounded by the smaller of their arities. Since $\sigma = \sigma'$ implies that the action sequences of the two parameter flows are identical, the normalization term is identical for both flows and can therefore be computed from either one.

The parameter $\alpha \in [0, 1]$ controls how many correspondences are required to achieve maximal similarity. Smaller values of α reduce this requirement, lowering the penalty for tuples with many parameters.

Example 2. We illustrate the similarity score on a range of parameter flows and assume $\alpha = 1$ for simplicity, i.e., all parameter correspondences must match to achieve a score of 1.

1. **Many parameters, high similarity.**

$$T_1 = (\langle x, y, z\rangle,\ \{\langle 1, 1\rangle, \langle 2, 2\rangle, \langle 3, 3\rangle\})$$
$$T_2 = (\langle x, y, z\rangle,\ \{\langle 1, 1\rangle, \langle 2, 2\rangle\})$$

Assume first and last actions (x and z) have arity 3.

$$S_1\big(T_1, T_2\big) = \frac{1+2}{1+3} = \frac{3}{4}.$$

2. **Many parameters, low similarity.**

$$T_1 = (\langle x, y\rangle,\ \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle, \langle 4,4\rangle\})$$
$$T_2 = (\langle x, y\rangle,\ \{\langle 1,2\rangle\})$$

Assume first and last actions (x and y) have arity 4.

$$S_1\big(T_1, T_2\big) = \frac{1+0}{1+4} = \frac{1}{5}.$$

3. **High similarity but mismatched actions.**

$$T_1 = (\langle x, y, z\rangle,\ \{\langle 1,1\rangle, \langle 2,2\rangle\})$$
$$T_2 = (\langle x, w, z\rangle,\ \{\langle 1,1\rangle, \langle 2,2\rangle\})$$

Since $\sigma \neq \sigma'$, we have

$$S_1\big(T_1, T_2\big) = 0.$$

4. **Different arities.**

$$T_1 = (\langle x, y\rangle,\ \{\langle 1,1\rangle, \langle 2,2\rangle\})$$
$$T_2 = (\langle x, y\rangle,\ \{\langle 1,1\rangle\})$$

Assume x has arity 3 and y has arity 4. Then

$$S_1\big(T_1, T_2\big) = \frac{1+1}{1+3} = \frac{2}{3}.$$

5. **Zero-arity actions.**

$$T_1 = (\langle x, y\rangle,\ \emptyset)$$
$$T_2 = (\langle x, y\rangle,\ \emptyset)$$

Assume that x or y has arity 0. Then

$$S_1\big(T_1, T_2\big) = \frac{1+0}{1+0} = 1.$$

6. **Duplicate parameters (capped similarity).**

$$T_1 = (\langle x, y\rangle,\ \{\langle 1,1\rangle, \langle 1,2\rangle, \langle 2,1\rangle, \langle 2,2\rangle\})$$
$$T_2 = (\langle x, y\rangle,\ \{\langle 1,1\rangle, \langle 1,2\rangle, \langle 2,1\rangle, \langle 2,2\rangle\})$$

Assume first and last actions (x and y) have arity 2.

$$\frac{1 + |F \cap F'|}{1 + \min(|\mathit{par}(x)|, |\mathit{par}(y)|)} = \frac{1+4}{1+2} = \frac{5}{3} > 1.$$

In the presence of duplicate parameters across positions (e.g., $y(a, a)$), the number of correspondences may exceed the normalization term; the outer $\min(1, \cdot)$ ensures the score remains bounded.

We define *problem-level structural regularity* by aggregating similarity scores across all pairs of action tuples of equal length within a plan.

Let $\pi = \langle a_1, \ldots, a_{|\pi|}\rangle$ be a plan. For $i \in \{1, \ldots, |\pi| - n + 1\}$, let

$$T^\pi(i, n) = T_{a_i, \ldots, a_{i+n-1}}$$

denote the parameter flow of the action tuple of length n starting at position i.

Definition 3. Let $\beta \geq 0$ be a tunable parameter controlling the importance of longer action tuples. We define the *problem-level structural regularity* of a plan π as

$$\mathit{structreg}_{\alpha,\beta}(\pi) = \sum_{n=2}^{|\pi|} \sum_{1 \leq i < j \leq |\pi| - n + 1} n^\beta \cdot S_\alpha\big(T^\pi(i, n),\ T^\pi(j, n)\big).$$

This score aggregates the similarity of all action tuples in the plan, with tuples of length n weighted by n^β, where $\beta \in [0, \infty)$. The parameter β controls how strongly longer tuples are emphasized. When $\beta = 0$, all tuple lengths contribute equally, recovering an unweighted formulation.

Example 3. We assume $\alpha = 1$ and $\beta = 0$ for simplicity. Let "$-$" denote an action that is distinct at each occurrence, i.e., it does not match any other action in the plan. We illustrate the problem-level structural regularity on plans with zero-arity actions, where $S_\alpha \in \{0, 1\}$, simplifying the analysis.

In this setting, each contribution can be expressed in terms of two quantities: the number of distinct (sub)tuples k and the number of occurrences f, resulting in $k \cdot \binom{f}{2}$.

1. $\pi = \langle x, y, z, -, -, -, x, y, z, -, -\rangle$.
There are 3 distinct tuples ($\langle x, y\rangle$, $\langle y, z\rangle$, $\langle x, y, z\rangle$), each occurring 2 times:

$$\mathit{structreg}_{1,0}(\pi) = 3 \cdot \binom{2}{2} = 3.$$

2. $\pi = \langle x, y, z, w, -, -, x, y, z, w, -\rangle$.
There are 6 distinct subtuples of $\langle x, y, z, w\rangle$, each occurring 2 times:

$$\mathit{structreg}_{1,0}(\pi) = 6 \cdot \binom{2}{2} = 6.$$

3. $\pi = \langle x, y, -, x, y, -, x, y, -, x, y\rangle$.
There is 1 distinct tuple ($\langle x, y\rangle$), occurring 4 times:

$$\mathit{structreg}_{1,0}(\pi) = 1 \cdot \binom{4}{2} = 6.$$

4. $\pi = \langle x, y, z, -, x, y, z, -, x, y, z\rangle$.
There are 3 distinct tuples ($\langle x, y\rangle$, $\langle y, z\rangle$, $\langle x, y, z\rangle$), each occurring 3 times:

$$\mathit{structreg}_{1,0}(\pi) = 3 \cdot \binom{3}{2} = 9.$$

More generally, the contribution of a repeated action tuple depends on the tuple length n, the number of occurrences f, and the similarity score S between pairs of parameter flows induced by its occurrences.

4.2 Domain-Level Structural Regularity

We extend structural regularity to the domain level to capture recurring patterns across multiple plans.

At the domain level, we first construct a concatenated plan from a set of plans corresponding to solved instances of the

domain. The parameter flow, similarity score, and aggregated similarity score are defined exactly as in the problem-level setting and applied directly to this concatenated plan. We then introduce a domain-level normalization to account for the number of sampled plans, ensuring that the resulting measure is not biased by the size of the plan set.

We perform a domain-level analysis that captures recurring grounded structure across multiple instances of a domain. In practice, it is not possible to consider all plans of a domain, as the number of problems and corresponding plans may be unbounded. Instead, we assume access to a finite set of solved instances and their corresponding plans.

To capture domain-level structure, it is not sufficient to compute structural regularity independently for each plan and aggregate the results, as this would ignore patterns that occur across different plans. By concatenating plans, parameter flows induced by tuples from different instances can be compared directly, enabling the metric to capture structural repetition both within and across plans.

Let $\pi_1, \ldots, \pi_n$ denote a set of plans obtained from solved instances of a domain. We construct a combined plan by concatenating these plans while preventing artificial subplans from spanning instance boundaries.

To this end, we introduce a set of barrier actions $\theta_1, \ldots, \theta_{n-1}$, where each θ_i is a unique identifier that does not occur elsewhere in any plan. These barrier actions ensure that any tuple containing a barrier is unique and therefore cannot contribute to structural regularity.

Definition 4. Let $\pi_1, \ldots, \pi_n$ be plans. The *concatenated plan* is defined as

$$\pi_{dom} = \pi_1 \circ \theta_1 \circ \pi_2 \circ \theta_2 \circ \cdots \circ \theta_{n-1} \circ \pi_n,$$

where each θ_i is a distinct barrier action not occurring in any π_j.

We define *domain-level structural regularity* by applying problem-level structural regularity to the concatenated plan and normalizing it with respect to the number of sampled plans.

The parameter flow, similarity score, and problem-level structural regularity $\mathit{structreg}_{\alpha,\beta}(\cdot)$ are defined as in Section 4.1 and apply directly to the concatenated plan π_{dom}.

Definition 5. Let n be the number of plans concatenated into π_{dom}, and let $\gamma \geq 0$ be a tunable parameter controlling the strength of the normalization with respect to the number of plans. The *domain-level structural regularity* is defined as

$$\mathit{structreg}^{dom}_{\alpha,\beta,\gamma} = \frac{\mathit{structreg}_{\alpha,\beta}(\pi_{dom})}{n^{\gamma}}.$$

This normalization compensates for the increase in repeated tuples when aggregating more plans. Scaling by n^{γ} ensures that the measure reflects the structural properties of the domain rather than the number of sampled plans.

4.3 Parameters and Interpretation

The parameters α, β, and γ control the sensitivity of the similarity score, the relative importance of longer action tuples, and the strength of normalization, respectively.

The underlying design of the metric, capturing parameter flow and aggregating repeated action tuples, is motivated by the structural properties of plans and is therefore largely domain-independent. In contrast, the parameters α, β, and γ control how strictly structural similarity is enforced and how strongly different components of the metric are weighted. As such, their optimal values depend on how structural regularity correlates with macro performance in practice and cannot be fully determined a priori.

In addition to parameter selection, interpreting the resulting structural regularity values requires defining thresholds that distinguish between domains where macro-learning yields low or high performance. In our experiments, we calibrate both parameter values and interpretation thresholds based on their correlation with observed macro-planning performance.

4.4 Computation

This section describes an efficient algorithm for computing domain-level structural regularity. The algorithm takes as input a set of plans from a domain and outputs the structural regularity, a scalar value estimating the expected impact of macro-learning on the domain.

While the definitions in the previous sections provide a direct formulation of the metric, a naive implementation would be computationally prohibitive. We therefore present an equivalent but optimized computation pipeline.

The pipeline consists of the following steps: preprocessing, identifying repeating tuples, parameter flow calculations and aggregated similarity score.

Preprocessing. In preprocessing, the input plans are concatenated into a single sequence, as described previously, with barrier actions inserted between consecutive plans.

Identifying Repeating Tuples. In this step, we consider only action names, ignoring parameters. We identify all repeating action tuples of length at least 2. For each tuple, we record the set of starting indices at which it occurs. Tuples that occur only once, or that contain a barrier symbol, are discarded.

To avoid enumerating all possible tuples, we exploit the fact that any repeating tuple of length n must share its prefix of length $n-1$ with another occurrence. We therefore construct repeating tuples incrementally: tuples of length n are generated by extending repeating tuples of length $n-1$, retaining only those extensions whose next action symbol also matches at the corresponding indices. Tuples of length 1 are computed only to initialize the incremental procedure and are not included in subsequent computations. Algorithm 1 demonstrates this computation.

Example 4. Consider the sequence of action names

$$\langle x,\ y,\ z,\ x,\ x,\ y,\ z \rangle.$$

The repeating action tuples are:

1. $n = 1:\ (x : [0, 3, 4],\ y : [1, 5],\ z : [2, 6])$
2. $n = 2:\ (xy : [0, 4],\ yz : [1, 5])$
3. $n = 3:\ (xyz : [0, 4])$
4. $n = 4:\ \emptyset$

Algorithm 1 Identification of repeating action tuples

Data: Action sequence $\langle a_0, \ldots, a_{n-1} \rangle$ with barrier symbols -1
Result: List of repeating tuples grouped by length
$R \leftarrow \emptyset$ `// Results grouped by tuple length`
$O \leftarrow$ map from action symbols to occurrence indices
for $i \leftarrow 0$ **to** $n-1$ **do**
  if $a_i \neq -1$ **then**
    append i to $O[a_i]$
$C \leftarrow \{O[s] \mid s \in dom(O), |O[s]| > 1\}$
`// Repeating tuples of length 1`
if $C = \emptyset$ **then**
  return $\emptyset$
$L \leftarrow 1$
while $C \neq \emptyset$ **do**
  $N \leftarrow \emptyset$ `// Next level (tuples of length` $L+1$`)`
  foreach *position list* $P \in C$ **do**
    $B \leftarrow$ empty map
    foreach $p \in P$ **do**
      $q \leftarrow p + L$
      if $q < n$ *and* $a_q \neq -1$ **then**
        append p to $B[a_q]$
    foreach *list* $P' \in B$ **do**
      if $|P'| > 1$ **then**
        append P' to N
  if $N = \emptyset$ **then**
    break
  append N to R
  $C \leftarrow N$
  $L \leftarrow L + 1$
return R

Parameter Flow Calculations. For each action tuple of length $n \geq 2$ and each of its recorded starting indices p, we extract the grounded actions at positions p and $p + n - 1$ in the concatenated plan and compute their parameter flow. The list of starting indices associated with each tuple is thus replaced by a list of parameter flow structures.

A naive implementation would recompute parameter flows for every occurrence of every tuple. Since parameter flow depends only on the parameters of the first and last actions, we instead memoize these computations. In repetitive domains, many tuples share identical parameter configurations, so each distinct pair is processed once and reused in constant time.

Aggregated Similarity Score. For each action tuple, we compute the similarity score between all pairs of associated parameter flows. These scores are summed and weighted by the tuple length, yielding the aggregated similarity score.

To avoid evaluating all $\binom{k}{2}$ pairs explicitly, we group identical parameter flows and count their occurrences. Since the similarity score depends only on the flow values, we can compute contributions once per distinct pair of flows and weight them by the number of corresponding occurrences. This yields an equivalent result while significantly reducing the number of required comparisons. As the number of distinct flow values is typically small, this reduces the complexity from quadratic in the number of occurrences to quadratic in the number of distinct flows, with an additional linear pass to construct the counts.

The final aggregated similarity score is obtained by summing these contributions across all tuples and weighting each by n^β. The resulting value is then normalized, yielding the domain-level structural regularity value as defined previously.

Complexity. Let $N = |\pi_{dom}|$ be the length of the concatenated plan. The preprocessing step is linear in N, and the identification of repeating tuples is quadratic in N.

Parameter flow computation is linear in the number of tuple occurrences, with memoization ensuring constant-time reuse for repeated parameter configurations.

For aggregation, grouping identical flows reduces the complexity from quadratic in the number of occurrences per tuple to quadratic in the number of distinct flow values, which is typically small.

Overall, the computation is dominated by the number of repeating tuples and their occurrences. In domains with high structural regularity, where many tuples repeat frequently, this quantity can be large. However, these are precisely the cases where memoization and flow grouping are most effective, allowing substantial reuse and reducing the average computational cost. In practice, these computations are fast and evaluating all 33 training domains takes approximately 260 seconds.

5 Experiments

Structural regularity introduces tunable parameters α, β, and γ that control how structural patterns are identified and aggregated. To evaluate and calibrate the metric, we cast the problem as a ternary classification task. Given a structural regularity score, we predict whether macro-learning has a negative, neutral, or positive impact.

Concretely, we map structural regularity scores to labels in $\{-1, 0, 1\}$ using two learned thresholds, which partition the score space into three regions. Both the structural regularity parameters and the thresholds are selected using leave-one-domain-out cross-validation. We evaluate performance using a confusion matrix and report classification accuracy.

Our goal is not to predict an absolute ground truth, but to align with empirical observations reported in the literature. As such, the evaluation reflects agreement with prior studies rather than definitive correctness.

5.1 Dataset Construction

We construct a dataset by aggregating results from 10 macro-learning studies (Armano, Cherchi, and Vargiu 2004; Botea et al. 2005; Coles and Smith 2007; Chrpa 2010; Dulac et al. 2013; Chrpa, Vallati, and McCluskey 2014; Asai and Fukunaga 2015; Chrpa and Siddiqui 2015; Hofmann, Niemueller, and Lakemeyer 2017; Chrpa and Vallati 2019). Each data point corresponds to a tuple consisting of a domain, a study, and a planner or configuration, resulting in 84 data points.

Each data point is assigned a label in $\{-1, 0, 1\}$ based on the reported impact of macro-learning. A label of 1 indicates

improved performance, -1 indicates degraded performance, and 0 indicates no substantial effect.

These labels are derived from qualitative descriptions in the literature and therefore involve a degree of interpretation. In particular, the distinction between -1 and 0, as well as between 0 and 1, can be ambiguous, whereas the distinction between clearly positive and clearly negative outcomes is typically more pronounced.

Rather than resolving disagreements between studies, we retain multiple labels for the same domain when they arise, reflecting variability across planners and configurations. We treat each $\langle$domain, study, configuration$\rangle$ tuple as a separate data point. However, structural regularity is computed once per domain and reused across all associated data points. During evaluation, predictions are made at the domain level and compared against all corresponding labels, where a prediction is evaluated against each associated label independently, such that consistency with the set of reported outcomes is rewarded. Thus, each domain-level prediction contributes multiple entries to the confusion matrices when multiple labels are associated with that domain.

All domains are drawn from planning competition benchmark sets, spanning multiple editions of the International Planning Competition (IPC) (Long et al. 2000; Bacchus 2001; Long and Fox 2003; Hoffmann and Edelkamp 2005; Fern, Khardon, and Tadepalli 2011; Linares López, Celorrio, and Olaya 2015; Vallati et al. 2015). When studies use instance generators, we instead use the corresponding testing instances of the Learning Track to remain representative and reproducible. The full dataset is provided in Appendix A.

For each domain, we solve a set of planning instances using the satisficing *LAMA* planner (Richter and Westphal 2010). We convert the resulting plans into makespan maximal form using *Mimir* (Ståhlberg 2023) when possible.[1] This re-orders the actions such that objects are more likely to repeat in subsequent actions, emulating the behavior of macro actions. Out of 1579 tasks, 1373 were solved and 1271 were made makespan maximal, each with 8 GiB memory limits and 30 minutes time out on an Intel Xeon Gold 6130 CPU. The structural regularity calculations used a memory limit of 90 GB and no time limit, though computations were quick. Of the 84 domain-track data points, 79 were used in our experiments. The remaining domains were excluded because no instances were successfully solved.

5.2 Training and Evaluation

We tune the parameters of structural regularity by minimizing a weighted classification loss under leave-one-domain-out cross-validation on a training set. Each fold excludes all data points associated with a single domain, ensuring that evaluation is performed on unseen domains. The dataset is partitioned into training and test sets at the domain level, such that all data points associated with a given domain appear in the same split, preventing information leakage. The partition is constructed by randomly shuffling domains (with a fixed random seed for reproducibility) and greedily assigning domains to the test set until it contains approximately one quarter of the data points for each label class. This ensures that the class distribution is approximately preserved across splits, yielding roughly 75% training and 25% test data overall.

Threshold-Based Classification. Let $s \in \mathbb{R}$ denote the structural regularity score of a domain. We introduce thresholds $t_1 < t_2$ that partition the score space into three regions:

$$\hat{y}(s) = \begin{cases} -1 & \text{if } s < t_1, \\ 0 & \text{if } t_1 \leq s < t_2, \\ 1 & \text{if } s \geq t_2. \end{cases}$$

Candidate thresholds are defined as midpoints between consecutive values in a sorted list of structural regularity scores. All valid candidate threshold pairs (t_1, t_2) with $t_1 < t_2$ are evaluated exhaustively, ensuring that the optimal thresholds are found with respect to the training objective.

Weighted Ordinal Loss. There is a class imbalance in our dataset, likely due to a reporting bias in the literature, where negative results are less frequently reported than positive or neutral ones. As a result, instances with label -1 are underrepresented.

To account for both this imbalance and the ordinal nature of the labels, we define a weighted penalty function. Let N_i denote the number of instances with label i. We define the class-dependent weight function $w : \{-1, 0, 1\} \to \mathbb{R}_{>0}$ as

$$w(y) = \frac{\max(N_{-1}, N_0, N_1)}{N_y}.$$

Additionally, for the predicted label $\hat{y}$ and the true label y, we define the penalty function as

$$\textit{penalty}(\hat{y}, y) = w(y) \cdot |\hat{y} - y|$$

This formulation ensures that misclassifications on underrepresented classes are penalized more heavily and that errors between -1 and 1 incur a larger penalty than errors involving the neutral class.

Parameter Selection. To search over hyperparameters, we define a discrete grid spanning $\alpha \in [0.0, 1.0]$ (step 0.1), $\beta \in [0.00, 3.00]$ (step 0.25), and $\gamma \in [0.00, 3.00]$ (step 0.25), giving $11 \cdot 13^2 = 1859$ configurations in total.

Each configuration is evaluated as follows. First, structural regularity scores are computed for all 33 training domains, a one-time process taking 213 seconds. Next, we perform leave-one-domain-out cross-validation, fitting thresholds on all but one domain and measuring performance on the held-out domain. The total loss for a configuration is the sum of penalties across all folds, with each configuration completing in 0.15–0.16 seconds.

After selecting the parameter configuration with the lowest total loss ($\alpha = 0.6$, $\beta = 0.0$, and $\gamma = 1.75$), thresholds are re-estimated using the full training set and are used for final evaluation ($t_1 = 146.77$, $t_2 = 105182.16$). Further parameter selection results are provided in Appendix B.

[1] Since the makespan-maximizing algorithm in Mimir does not support derived predicates, plans from PSR, Philosophers, and Optical Telegraphs could not be makespan maximized.

	Training						Testing					
	Structural Regularity			Theoretical Optimum			Structural Regularity			Theoretical Optimum		
True label	Pred. -1	Pred. 0	Pred. 1	Pred. -1	Pred. 0	Pred. 1	Pred. -1	Pred. 0	Pred. 1	Pred. -1	Pred. 0	Pred. 1
-1	3	7	1	7	3	1	0	2	1	3	0	0
0	0	12	3	0	12	3	1	3	1	0	4	1
1	0	18	16	0	3	31	1	5	5	0	0	11

Table 1: Confusion matrices on training and test data for the training-tuned structural regularity classifier and the theoretical optimum. Since some domains have multiple disparate labels, due to macro-learning approaches sometimes performing well and sometimes poorly, it is not possible to get perfect classification on either dataset.

The large gap between the thresholds reflects the combinatorial growth of structural regularity, which leads to substantially larger scores in domains with strong repetition. The optimal value $\beta = 0.0$ indicates that weighting longer tuples more heavily does not improve predictive performance, suggesting that shorter and more frequent structural patterns are more informative than longer ones. The intermediate value $\alpha = 0.6$ reflects a balance between strict and permissive matching of parameter correspondences. Finally, $\gamma = 1.75$ is close to 2, corresponding to approximately quadratic normalization in the number of plans.

Training Results. We evaluate performance on the training set using the selected parameters and thresholds. The training confusion matrix is reported in Table 1. This yields a classification accuracy of 0.52 over 60 data points, which is above a uniform random baseline of 0.33 but slightly below a majority-class baseline of 0.57. The latter is inflated by class imbalance in the dataset, and structural regularity is not intended to reproduce this effect. Due to conflicting labels for the same domain across different studies, perfect classification is not achievable under this evaluation setting for a domain-level predictor. The maximum attainable accuracy under this evaluation setting on the training data is $50/60 = 0.83$, and the corresponding confusion matrix is reported in Table 1. Comparing the two confusion matrices, we observe that the model captures the neutral class well and identifies a subset of positive and negative instances, but fails to recover many strongly labeled cases, instead assigning them to the neutral class. Overall, this indicates that the model fits a meaningful portion of the structure in the training data, while leaving a considerable fraction of possibly separable signal unexplained. Appendix C shows that these observations are stable across training-test splits.

Test Results. Using the tuned parameters and thresholds, we evaluate performance on the held-out test set. The associated confusion matrix is reported in Table 1. This yields a classification accuracy of 0.42, which is above the uniform random baseline of 0.33, but below the majority-class baseline of 0.58. For comparison, the maximum attainable confusion matrix under this evaluation setting is also reported in Table 1, indicating a substantially higher attainable accuracy under perfect alignment with the labels.

Comparing these results, we observe that structural regularity captures a measurable predictive signal on unseen domains but exhibits limited generalization. In particular, the model tends to overpredict the neutral class. This suggests that while structural regularity captures domain-level patterns correlated with macro-learning performance, it is not sufficient to predict outcomes across domains. Part of this limitation may stem from aggregating labels across different planners and configurations, which can obscure differences between macro-learning approaches. Appendix C shows that these observations are also stable across training–test splits.

6 Conclusions and Future Work

We address the problem of predicting when macro-learning is beneficial in automated planning, where performance varies widely across domains and applying macro-learning often requires costly trial and error. To this end, we introduce *structural regularity*, a metric that captures recurring structural patterns in plans by analyzing action sequences and parameter reuse without relying on object identities.

Our approach is based on three components: parameter flow, which tracks how parameters propagate across action tuples; a similarity measure that enables partial matching of these flows; and an aggregation scheme that captures how frequently such structures occur within and across plans. At the domain level, this yields a scalar score intended to reflect the degree of exploitable structure for macro-learning.

We evaluate the metric on a dataset compiled from 10 macro-learning studies, covering 47 domains and 84 data points labeled by macro-learning impact. This dataset constitutes a contribution, enabling systematic evaluation of predictive approaches in this setting.

Our results show that structural regularity captures signals correlated with macro-learning performance. While the metric fits the training data reasonably well, it generalizes only weakly to unseen domains, achieving performance above a uniform random baseline but below a majority-class baseline. This indicates that structural regularity reflects meaningful structural properties, but is not sufficient on its own for reliable cross-domain prediction. Overall, these findings suggest that structural regularity is a step toward predicting when macro-learning should be applied, while remaining computationally efficient as a practical preprocessing step.

A natural direction for future work is to develop new macro-learning approaches inspired by the parameter flow and similarity notions introduced in this paper. In particular, these ideas could be used to construct partially grounded macros that generalize across instances.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

Alarnaouti, D.; Baryannis, G.; and Vallati, M. 2023. Reformulation Techniques for Automated Planning: A Systematic Review. *Knowledge Engineering Review*, 38: e9.

Armano, G.; Cherchi, G.; and Vargiu, E. 2004. Automatic Generation of Macro-Operators from Static Domain Analysis. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 955–956.

Asai, M.; and Fukunaga, A. 2015. Solving Large-Scale Planning Problems by Decomposition and Macro Generation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 25, 16–24.

Bacchus, F. 2001. The AIPS'00 Planning Competition. *AI Magazine*, 22(3): 47–56.

Bäckström, C. 1998. Computational Aspects of Reordering Plans. *Journal of Artifical Intelligence*, 9: 99–137.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research*, 24: 581–621.

Castellanos-Paez, S.; Rombourg, R.; and Lalanda, P. 2021. ERA: Extracting Planning Macro-Operators from Adjacent and Non-Adjacent Sequences. In *Proceedings of the 17th Pacific Rim Knowledge Acquisition Workshop (PKAW 2020)*, 30–45. Springer.

Chrpa, L. 2010. Generation of Macro-Operators via Investigation of Action Dependencies in Plans. *The Knowledge Engineering Review*, 25(3): 281–297.

Chrpa, L.; and Siddiqui, F. H. 2015. Exploiting Block Deordering for Improving Planners Efficiency. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*.

Chrpa, L.; and Vallati, M. 2019. Improving Domain-Independent Planning via Critical Section Macro-Operators. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 7546–7553.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2014. MUM: A Technique for Maximising the Utility of Macro-Operators by Constrained Generation and Use. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A.; and Smith, A. 2007. Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *Journal of Artificial Intelligence Research*, 28: 119–156.

Dulac, A.; Pellier, D.; Fiorino, H.; and Janiszek, D. 2013. Learning Useful Macro-Actions for Planning with N-Grams. In *Proceedings of the IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, 803–810. IEEE.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning*, 84(1–2): 81–107.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Hoffmann, J.; and Edelkamp, S. 2005. The Deterministic Part of IPC-4: An Overview. *Journal of Artificial Intelligence Research*, 24: 519–579.

Hofmann, T.; Niemueller, T.; and Lakemeyer, G. 2017. Initial Results on Generating Macro Actions from a Plan Database for Planning on Autonomous Mobile Robots. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS)*.

Linares López, C.; Celorrio, S. J.; and Olaya, A. G. 2015. The deterministic part of the seventh International Planning Competition. *Artificial Intelligence*, 223: 82–119.

Long, D.; and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

Long, D.; Kautz, H.; Selman, B.; Bonet, B.; Geffner, H.; Koehler, J.; Brenner, M.; Hoffmann, J.; Rittinger, F.; Anderson, C. R.; Weld, D. S.; Smith, D. E.; and Fox, M. 2000. The AIPS-98 Planning Competition. *AI Magazine*, 21(2): 13–33.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Ståhlberg, S. 2023. Lifted Successor Generation by Maximum Clique Enumeration. In Gal, K.; Nowé, A.; Nalepa, G. J.; Fairstein, R.; and Rădulescu, R., eds., *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, 2194–2201. IOS Press.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 International Planning Competition: Progress and Trends. *AI Magazine*, 36(3): 90–98.

A Dataset Datasheet

Table 2 lists the full dataset used in our experiments. Each row corresponds to a data point consisting of a domain, a specific IPC track, the macro-learning study from which the label was derived, the assigned label, and the number of solved problems. The reported number of solved problems corresponds to the number of instances we successfully solved per domain with LAMA. Of the 84 domain–track data points, 79 were used in our experiments. The remaining cases were excluded because no instances were successfully solved, and structural regularity cannot be computed in the absence of plan data. For IPC learning track domains, we only evaluated the testing instances. The domains are taken from Fawcett's collection of planning tasks.[2]

To prevent data leakage, we group all data points that share the same domain and track during splitting and cross-validation, ensuring that they are never separated between training and evaluation. Each label, however, still contributes equally to the evaluation. Labels are defined as -1 (negative impact), 0 (neutral impact), and 1 (positive impact).

Table 2: Dataset used in our experiments.

Domain	Track	Source	Label	#Solved
Blocksworld	ipc-2000-typed	Armano, Cherchi, and Vargiu (2004)	1	35
Elevator	hoffmann-miconic-simple-adl	Armano, Cherchi, and Vargiu (2004)	−1	150
Satellite	ipc-2004-deterministic-strips	Botea et al. (2005)	1	36
Airport	ipc-2004-deterministic-adl	Botea et al. (2005)	0	34
Pipesworld No-Tankage	ipc-2004-deterministic	Botea et al. (2005)	1	43
Pipesworld Tankage	ipc-2004-deterministic	Botea et al. (2005)	0	43
PSR	ipc-2004-deterministic-middle-compiled	Botea et al. (2005)	0	50
Philosophers	ipc-2004-deterministic-derived-adl	Botea et al. (2005)	1	48
Optical Telegraph	ipc-2004-deterministic-derived-adl	Botea et al. (2005)	1	4
Driverlog	ipc-2002-deterministic-untyped	Coles and Smith (2007)	−1	20
Freecell	ipc-2000	Coles and Smith (2007)	−1	79
Satellite	ipc-2004-deterministic-strips	Coles and Smith (2007)	0	36
Depots	ipc-2002-deterministic-untyped	Coles and Smith (2007)	1	20
Airport	ipc-2004-deterministic-adl	Coles and Smith (2007)	0	34
Philosophers	ipc-2004-deterministic-derived-adl	Coles and Smith (2007)	1	48
Pipesworld Tankage	ipc-2004-deterministic	Coles and Smith (2007)	1	43
Pipesworld No-Tankage	ipc-2004-deterministic	Coles and Smith (2007)	0	43
Blocksworld	ipc-2000-typed	Chrpa (2010)	1	35
Depots	ipc-2002-deterministic-untyped	Chrpa (2010)	1	20
Zenotravel	ipc-2002-deterministic-untyped	Chrpa (2010)	0	20
Rovers	ipc-2002-deterministic-typed	Chrpa (2010)	0	40
Gripper	ipc-1998	Chrpa (2010)	1	20
Satellite	ipc-2004-deterministic-strips	Chrpa (2010)	−1	36
Freecell	ipc-2000	Chrpa (2010)	−1	79
Sokoban	ipc-2008-deterministic-sat-strips	Chrpa (2010)	−1	29
Gold Miner	ipc-2008-learning	Chrpa (2010)	1	30
Pipesworld No-Tankage	ipc-2004-deterministic	Chrpa (2010)	−1	43
Pipesworld Tankage	ipc-2004-deterministic	Chrpa (2010)	−1	43
N-puzzle	ipc-2008-learning	Chrpa (2010)	−1	30
Blocksworld	ipc-2011-learning	Dulac et al. (2013)	1	35
Ferry	ipc-2011-learning	Dulac et al. (2013)	1	30
Satellite	ipc-2011-learning	Dulac et al. (2013)	1	28
Gripper	ipc-2011-learning	Dulac et al. (2013)	1	17
Grid	ipc-2011-learning	Dulac et al. (2013)	1	30
Depots	ipc-2011-learning	Dulac et al. (2013)	1	0
Mprime	ipc-2011-learning	Dulac et al. (2013)	1	30
Parking	ipc-2011-learning	Dulac et al. (2013)	1	30
Barman	ipc-2011-learning	Dulac et al. (2013)	1	15
Barman	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	0	15
Depots	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	1	0
Gripper	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	1	20
TPP	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	1	30
Parking	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	0	20

[2] https://github.com/fawcettc/planning-instances

Domain	Track	Source	Label	#Solved
Blocksworld	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	1	30
Rovers	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	1	40
Satellite	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	1	28
Spanner	ipc-2011-learning	Chrpa, Vallati, and McCluskey (2014)	1	0
Barman	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	1	20
Woodworking	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	1	20
Transport	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	1	18
TPP	ipc-2011-learning	Asai and Fukunaga (2015)	1	30
Elevator	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	1	20
Blocksworld	ipc-2011-learning	Asai and Fukunaga (2015)	0	30
Satellite	ipc-2011-learning	Asai and Fukunaga (2015)	0	36
Nomystery	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	−1	12
Floortile	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	0	7
Openstacks	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	−1	20
Parking	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	−1	20
Visitall	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	0	20
Parcprinter	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	0	20
Pegsol	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	0	20
Scanalyzer	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	0	20
Tidybot	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	0	17
Sokoban	ipc-2011-deterministic-sat-strips	Asai and Fukunaga (2015)	0	19
Rovers	ipc-2011-learning	Asai and Fukunaga (2015)	0	30
Barman	ipc-2011-learning	Chrpa and Siddiqui (2015)	1	15
Depots	ipc-2011-learning	Chrpa and Siddiqui (2015)	1	0
Gripper	ipc-2011-learning	Chrpa and Siddiqui (2015)	1	17
TPP	ipc-2011-learning	Chrpa and Siddiqui (2015)	1	30
Parking	ipc-2011-learning	Chrpa and Siddiqui (2015)	−1	30
Blocksworld	ipc-2011-learning	Chrpa and Siddiqui (2015)	1	30
Rovers	ipc-2011-learning	Chrpa and Siddiqui (2015)	1	30
Satellite	ipc-2011-learning	Chrpa and Siddiqui (2015)	1	28
Spanner	ipc-2011-learning	Chrpa and Siddiqui (2015)	0	0
Hiking	ipc-2014-satisficing	Hofmann, Niemueller, and Lakemeyer (2017)	1	20
Barman	ipc-2014-satisficing	Hofmann, Niemueller, and Lakemeyer (2017)	1	20
Blocksworld	ipc-2011-learning	Hofmann, Niemueller, and Lakemeyer (2017)	1	30
Barman	ipc-2011-learning	Chrpa and Vallati (2019)	1	15
Blocksworld	ipc-2011-learning	Chrpa and Vallati (2019)	1	30
Depots	ipc-2011-learning	Chrpa and Vallati (2019)	1	0
Gripper	ipc-2011-learning	Chrpa and Vallati (2019)	1	17
Matching-BW	ipc-2008-learning	Chrpa and Vallati (2019)	1	20
Rovers	ipc-2011-learning	Chrpa and Vallati (2019)	1	30
Sokoban	ipc-2008-learning	Chrpa and Vallati (2019)	0	29
Parking	ipc-2008-learning	Chrpa and Vallati (2019)	−1	30

B Parameter Heatmaps

Figure 1 shows the full parameter grid explored during hyperparameter tuning. Since the parameter space is three-dimensional, we visualize it using pairwise heatmaps, where each plot reports the minimum loss over the remaining parameter.

The heatmaps reveal consistent trends across parameter combinations. In particular, performance is sensitive to both β and γ. For β, the best performance is consistently obtained at $\beta = 0.0$, indicating that emphasizing longer tuples is detrimental. The parameter γ has a strong influence, with best results concentrated around $\gamma \approx 1.75$. In contrast, α shows lower sensitivity, with a relatively broad region of near-optimal values around 0.6.

Overall, these results indicate that the parameter choices reported in the main text are sound, with each parameter exhibiting a relatively smooth performance profile around the selected values rather than a sharply tuned optimum.

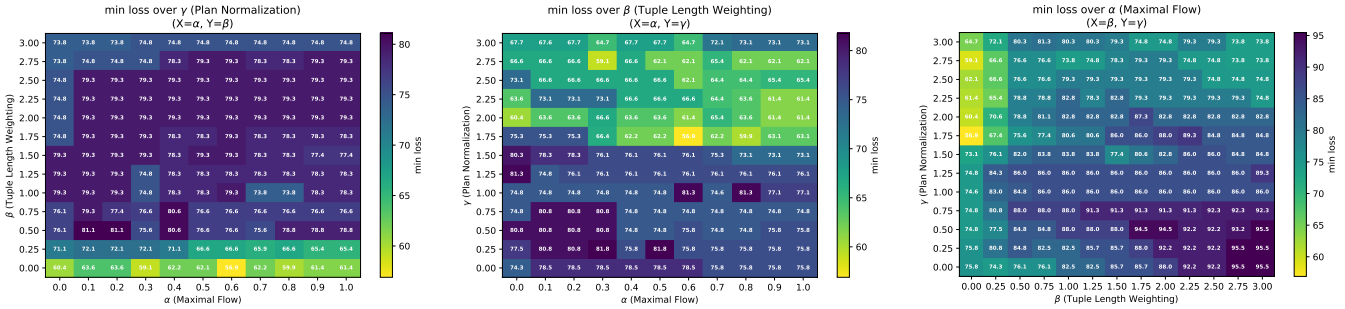

Figure 1: Pairwise heatmaps of the structural regularity parameters α, β, and γ, showing minimum loss over the third parameter.

C Robustness Across Random Seeds

To assess the stability of our results with respect to the random data split, we repeat the full training and evaluation procedure across 10 different random seeds. The split corresponding to seed 0 is the one used in the main text. For each split, we report the training and test accuracy, along with the selected parameter values.

Table 3 shows that performance varies across seeds, with some splits yielding higher or lower accuracy than the one used in the main text. However, the overall results remain comparable, and the mean performance is close to the reported results. This indicates that the observed behavior is not an artifact of a particular random split.

The selected parameter β is frequently set to 0.0, supporting the conclusion that emphasizing longer tuples does not consistently improve predictive performance. The parameters α and γ show moderate variation, but remain within a similar range as the values reported in the main text.

Table 3: Results across 10 random seeds of data splitting.

Seed	Train Acc	Test Acc	α	β	γ
0	0.52	0.42	0.6	0.00	1.75
1	0.55	0.32	0.6	0.00	1.75
2	0.28	0.16	0.5	3.00	1.75
3	0.32	0.11	0.9	3.00	1.00
4	0.52	0.37	0.6	0.00	2.00
5	0.55	0.32	0.6	0.00	1.75
6	0.58	0.47	0.7	0.00	2.75
7	0.57	0.26	0.6	0.00	1.75
8	0.55	0.58	0.8	0.75	2.75
9	0.27	0.26	0.7	3.00	2.75
Mean	0.47	0.33	0.66	0.68	2.03
Std	0.12	0.14	0.12	1.28	0.63

Due to conflicting labels across studies, the theoretical optimum accuracy is strictly below 1 for all splits reported in Table 3, with exact optimums varying between seeds.