Discrete Time-Series Clustering and Linear Temporal Logic Delineation

by

Brennan Cruse

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
May 2022

Abstract

The collection of information in this data-driven world has become paramount to the way businesses and individuals interact with society. From personal wearable technology to weather prediction, sales forecasting, and everything in between, a common characteristic among a significant proportion of this data is its relationship with respect to time. Acting as a key to unlock the power contained within time-series data, analytical techniques and logical representations provide a basis to translate data into learning outcomes. While time series data represents a significant opportunity for institutions and individuals to learn from the past to improve the future, the prevalence of unstructured data within real-world settings is an active challenge for existing analytical techniques. This impediment is especially relevant within research areas such as goal recognition, policy summarization, and system dynamic modelling, where the shared objective is to derive meaning from observed behavior. To establish meaningful insights from unstructured time-series data, partitions and patterns must be identified to effectively differentiate observations based on temporal attributes.

To address this, we propose two novel approaches, which both leverage linear temporal logic to provide structure to unstructured discrete time-series data by identifying and contrastively explaining the differences between an unspecified quantity of discrete time-series observations.

Our first proposed approach discovers a feature set of relevant temporal specifications to represent observations in vector-space, clusters data points via traditional clustering algorithms, and delineates clusters via conjunction of linear temporal logic features. Within reasonable search limits, we discover a near-perfect success rate for accurate and complete cluster definitions found by our algorithm for six simulated evaluation domains of three unique vocabulary sizes.

Our second proposed approach embraces a tree-based perspective to organize observations into clusters. By employing a Monte Carlo node-splitting approach, our algorithm seeks balance to contrastively divide any given set of discrete time-series observations into two sets with an accompanying temporal logic specification satisfying one of the sets. Recursively applying this procedure, we demonstrate the effectiveness of our approach to cluster and delineate discrete time-series observations, allowing temporal logic specifications to evoke insight at each level of the resulting tree.

Acknowledgments

I would like to first express my sincere gratitude to my supervisor, Dr. Christian Muise. Thank you for your continuous support, leadership, and guidance throughout my Masters studies and completion of this research. Your patience and mentorship has been instrumental in the pursuance of my goals.

I would also like to thank my fellow Mμ-lab members, as well as all faculty, staff, and students in the School of Computing. Despite the many challenges associated with remote learning and completing a graduate degree in the midst of a global pandemic, I have always felt a strong sense of community. I cherish the relationships that we have built.

Last but not least, I would like to thank my family and friends for being a constant source of motivation and encouragement. I would not have been able to do this without the incredible support network I am so fortunate to have.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

As dependence on time-series data grows within this technology-driven world, the demand for better techniques to analyze and understand temporal trends becomes increasingly prominent. Unlike the analysis of cross-sectional data which evaluates observations at a single point in time, time-series data evaluates observations at several points in time. This leads to an abundance of data points, where the interaction of binary variables, with respect to their order of occurrence, record complex phenomena. Especially in real-world settings, however, several events occur simultaneously. Consequently, the information contained within time-series data can be enormously powerful, but proportionately difficult to analyze, due to its lack of structure. This lack of structure is typically associated with abundant data, consisting of no order, categorization, or implied meaning. While time-series data represents a significant opportunity for institutions and individuals to learn from the past and present to improve the future, the prevalence of unstructured data within real-world settings

represents an active challenge for existing analytical techniques. To address this challenge, we propose a novel approach to provide structure to unstructured time-series data. By leveraging linear temporal logic, our proposed method successfully clusters an unspecified quantity of discrete time-series observations and contrastively explains chosen allocations.

1.2 Problem

Discrete time-series data, which can take the form of traces, exist naturally within a diverse range of domains, with examples including financial transactions [47], weather recordings [51], pandemic severity logs [8], and rock climbing movements [33]. While the sources and diversity of data collection are extensive, the structure of resulting observations are more difficult to pinpoint, impacting the ability of researchers to isolate and understand patterns.

Most prior works in this area have focused on effectively summarizing a single set of traces by determining temporal specifications that are satisfied by all traces in a given set. More recently, however, research has shifted focus towards contrastive explanations, where the task is to automatically identify specifications that differentiate two sets of traces. While contrastive explanations arguably offer greater insight than summarization-based approaches, because they allow trade-offs within plan rationale to be understood, the conditions required for these techniques to be applicable are quite niche. Most importantly, traditional contrastive explanation research condenses the assertion of contrast to exactly two predefined sets of traces using positive and negative labels. In practical applications however, predefined labels are often nonexistent, and natural decomposition likely leads to more than two categories of

observations. To accommodate this, we re-imagine contrastive explanations to account for multiple sets of traces. We also facilitate the natural discovery of contrastive sets, such that prior definitions of groups are not required as input. Since we are the first to both cluster and delineate multiple sets of traces, there is no baseline for us to compare our approach to. By automating the discovery of similar groups of traces and describing temporal differences amongst clusters, our approach expands the scope and adaptability of contrastive explanations.

The input to the problem is a set of plan traces, representing discrete time-series data to be evaluated. This collection of input traces takes the form of $\{Trace_1$, $Trace_2$, ..., $Trace_n\}$, and is unlabelled, containing no predefined order. Each trace within this input set represents an individual data point to be categorized and delineated. The output of the problem is a collection of cluster-defining LTL statements taking the form of $\{LTL_1, LTL_2, ..., LTL_k\}$, that some traces from the input set satisfy and some traces do not. Acting as a sorting mechanism for traces, these discovered formulas achieve both clustering and delineation objectives, in addition to allowing new traces to be seamlessly incorporated into the understanding of larger systems.

1.3 Objective

The primary objective of this work is to explore the topic of discrete time-series clustering and examine its practical application within simulated environments. Two unique methodologies are proposed and applied to the explanation of traces within a variety of diverse sample domains. This thesis aims to provide a mechanism for researchers to achieve stronger insights when analyzing discrete time-series data by establishing two unique frameworks to effectively cluster and delineate traces.

1.4 Contributions

The following contributions were made in pursuit of the research objectives.

- A vector-space method is proposed which uses the BayesLTL framework [39], as well as clustering algorithms to infer LTL specifications for contrastive explanations considering any quantity of candidate groups of traces. LTL features are learned by analyzing contrastive explanations between sample groups of traces, leading to intelligent groupings via traditional algorithmic clustering. Our novel procedure operates according to the following steps:

    1. Process input set of traces to establish feature set of relevant context-aware temporal properties. Features take the form of LTL formulas, which each trace either satisfies or does not satisfy. This results in matrix of binary measurements representing temporal characteristics of traces.
    2. Cluster input set of traces via traditional clustering algorithms based on established feature set to identify labels and allocate traces into subgroups.
    3. Analyze resulting clusters according to their collective entailment of initial LTL features. Embrace conjunctive LTL search space to establish cluster definitions and differentiate groups.

- A tree-based method is also introduced to infer LTL specifications for contrastive explanations, where the objective is to automatically identify k sets of traces guided by temporal logic specifications. By evaluating sets of traces on a single-split basis, a recursive tree structure is generated, which is demonstrated to effectively explain the characteristics of traces. Our tree-based method's novel procedure is as follows:

1. Initialize input set of traces within root node of a binary tree data structure.
2. Recursively execute novel node splitting approach that seeks balance to identify LTL capable of dividing nodes into two similarly sized subsets.
3. Analyze cluster definitions by reading LTL off of the resulting tree via the path of the root to terminal nodes.

1.5 Organization of Thesis

This thesis begins with an introduction to the background of linear temporal logic and plan explanations as applied to discrete time-series data in Chapter 2. The data used to evaluate the methodologies within this study is introduced in Chapter 3. The first vector space clustering methodology is discussed and evaluated in Chapter 4, followed by the second tree-based methodology in Chapter 5. Chapter 6 summarizes this thesis in the context of related work, and finally, Chapter 7 concludes this study by comparing our two methodologies and providing an overview of potential future extensions.

Chapter 2

Background

2.1 Discrete Time-Series Data

A **trace** is a form of time-series data that is designed to record changing states of the world over a period of time. Traces are composed of a series of **bit vectors** that individually describe instances of time, and are referred to as **steps** or **states**. Each bit within a given bit vector corresponds with a binary variable, which we refer to as a **fluent**, that is either true or false at each time step. While traces represent a discrete form of time-series data, continuous variables can also be discretized and represented within traces through the use of bins. Additionally, a **set of traces** can be constructed (which is the structure of our algorithm's input), where multiple distinct traces exist within an even larger structure, which is denoted simply as an array of traces. An example of the structure of a single trace is presented in Figure 2.1, where the trace of a traffic light transitioning from green to yellow to red is depicted and labelled.

As can be seen by this traffic light example, traces are a powerful way to represent temporal systems, since traces allow changing states over time to be accurately

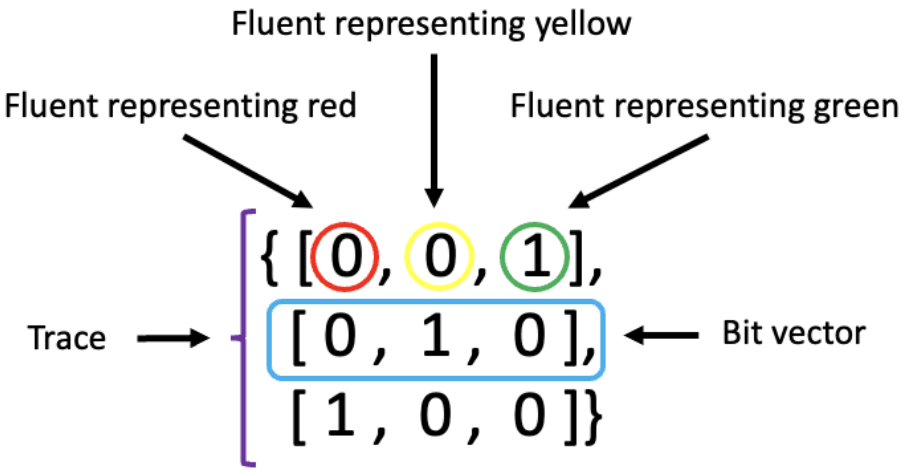

Figure 2.1: Example of a Trace and its Labelled Components

An example of a trace representing a traffic light that transitions from green to yellow to red. Each bit vector within this example represents a state of the world at a unique step in time, and the order of these bit vectors allows changes over time to be understood.

recorded via this notation. Additionally, these variables allow the interaction of various elements to be observed as they become active and inactive in different states. This also allows rules to be deduced that can provide greater insight into system dynamics. For instance, within the traffic light example, we can observe that the light is only ever one colour at a given time. In order to express these rules, state-based modal logics, such as *linear temporal logic*, can seamlessly be integrated into the analysis of trace data, which allows for powerful insights to be discovered and described.

2.2 Linear Temporal Logic (LTL)

Due to its powerful ability to encode relationships between events in time, LTL represents an effective medium for describing system behaviour in relation to the past, present, and future. LTL is a modal logic, and was first proposed by Pnueli in 1977

[52]. It has become widely adopted for applications such as automata-theoretical model checking [66, 54, 43], property expression in formal verification [30, 41], and as a specification language [39, 35, 44]. Due to its modalities referring to time, LTL is capable of expressing concepts such as possibility and necessity. For example, the characteristic of a traffic light always turning red after yellow can be represented by LTL as $G(\neg yellow|X(red))$. This example demonstrates necessity because the light is defined to always be yellow or not yellow. Possibility is also demonstrated because if the light is yellow in the current state, it will be red in the next state.

The syntax of LTL consists of two fundamental operators, "next"(X) and "until"(U), as well as additional operators built from those. The first fundamental operator, X, is designed to provide a constraint for the next period in time, where the proposition $X\phi$ is defined to be true if in the next time period ϕ is true. The second fundamental operator, U, is designed to connect various fluents with each other, where the proposition $\psi\ U\ \phi$ is defined to be true if ψ remains true in every state until ϕ becomes true. In addition to the two fundamental operators, higher-order operators can also be arbitrarily defined by practitioners using complicated formulas of X and U to create syntactically simple representations of advanced logical concepts for strengthened usability. For example, within BayesLTL [39], the template operator $\varphi_{stability}$ is defined as true if the proposition p_i eventually occurs and stays true forever. While in the lowest level of LTL, the most simple way to represent this concept is as $(1U!(1U!p_i))\wedge!(1U!(!p_i|!(1U!(p_i|!(1Up_i)))))$, BayesLTL's template definition allows the higher-level formula $stability(p_i)$ to represent the same thing. Other template operators used within BayesLTL can also be examined in Table 2.2. Traditionally within LTL applications, four common higher-order operators are defined, which are

"eventually"(F), "global"(G), "weak until"(W), and "release"(R). The proposition $F\phi$ is defined to be true if ϕ eventually becomes true at some future period. The proposition $G\phi$ is defined to be true if ϕ is true throughout the entire trace. The proposition $\psi\ W\ \phi$ is defined to be true if ψ holds until ϕ is true, and if ϕ never becomes true, then ψ must hold true globally. The proposition $\psi\ R\ \phi$ is defined to be true if ϕ is true up to and including the state where ψ first becomes true and if ψ never becomes true, ϕ must be true globally. Finally, the proposition $\psi\ M\ \phi$ is the same as release, but the release condition ψ must occur. An overview of these operators are represented within Table 2.1. By applying these unique operators, LTL possesses a vast ability to describe system behaviour within a wide variety of diverse domains.

Operator	Syntax	Meaning	Unabbreviated
Next	$X\phi$	ϕ has to hold at the next state	$X\phi$
Until	$\psi\ U\ \phi$	ψ has to hold at least until ϕ becomes true, which must hold at the current or a future position	$\psi\ U\ \phi$
Eventually	$F\phi$	ϕ eventually has to hold (may later become false)	$1\ U\ \phi$
Global	$G\phi$	ϕ has to hold on the entire subsequent path	$!(1\ U\ !\phi)$
Weak Until	$\psi\ W\phi$	ψ has to hold at least until ϕ; if ϕ never becomes true, ψ must remain true forever	$\psi\ U\ (\phi \mid !(1\ U\ !\psi))$
Release	$\psi\ R\ \phi$	ϕ has to be true until and including the point where ψ first becomes true; if ψ never becomes true, ϕ must remain true forever	$\phi\ U\ ((\phi\ \&\ \psi) \mid !(1\ U\ !\phi))$
Strong Release	$\psi\ M\ \phi$	ϕ has to be true until and including the point where ψ first becomes true, which must hold at the current or a future position	$\phi\ U\ (\phi\ \&\ \psi)$

Table 2.1: LTL Operators, Syntax, and Meaning

Standard LTL operators that are used to encode formulae about temporal paths, along with their associated syntax and meaning. LTL's fundamental operators, Next and Until, comprise the language, and additional operators are build from the fundamental operators for succinctness and readability. Unabbreviated equivalences of operators are also presented.

When evaluating the size $|\varphi|$ of an LTL formula, we embrace Gaglione et al.'s

definition, which counts the number of unique subformulas contained within an expression. For example, the size of $\varphi = (p\ UXq) \vee Xq$ is 5 because the unique subformulas in φ are p, q, Xq, $p\,UXq$, and $(p\,UXq) \vee Xq$ [28]. Similiar to BayesLTL [39], we embrace the same interpretable templates used by Kim et al. in our research; these templates can be viewed in Table 2.2.

2.3 Contrastive Explanations

As an alternative to traditional planning research which seeks to investigate the process of identifying optimal plans from problems, the area of plan explanation seeks to identify and describe characteristics of problems from plans. To make sense of observed actions, plan explanation research focuses on automatically learning temporal properties that allow system behaviour to be modelled, understood, and predicted. Contrastive explanations elevate the plan explanation problem to describe temporal differences between two sets of plan traces. Formulated in temporal logic, contrastive explanations use the entailment of logical statements to differentiate traces; a trace entails a formula when the behaviour within the trace is consistent with the formula's specified logic. The goal of contrastive explanations is to then identify formulas that are entailed by all traces in a predefined positive set of traces, but not entailed by all of the traces in a predefined negative set of traces. An example of contrastive explanations in action is presented in Figure 2.2. In this example, we observe two unique traffic light systems. In the first system, which is commonly found in Canada, the traffic light transitions from red, to green, to yellow, then repeats this cycle. The second system is common in Europe and the UK, where red/yellow also occurs after green and before red. To differentiate these two systems, we can use the following

LTL statement:

$$G(\neg red | X(green))$$

This contrastive explanation translates to "green comes next after red", which is true for system #1, but false for system #2. Since this example contains only three fluents and limited cyclical observations, it is easy to manually differentiate these systems; however, when additional complexity is introduced, automatic generation techniques must be relied on.

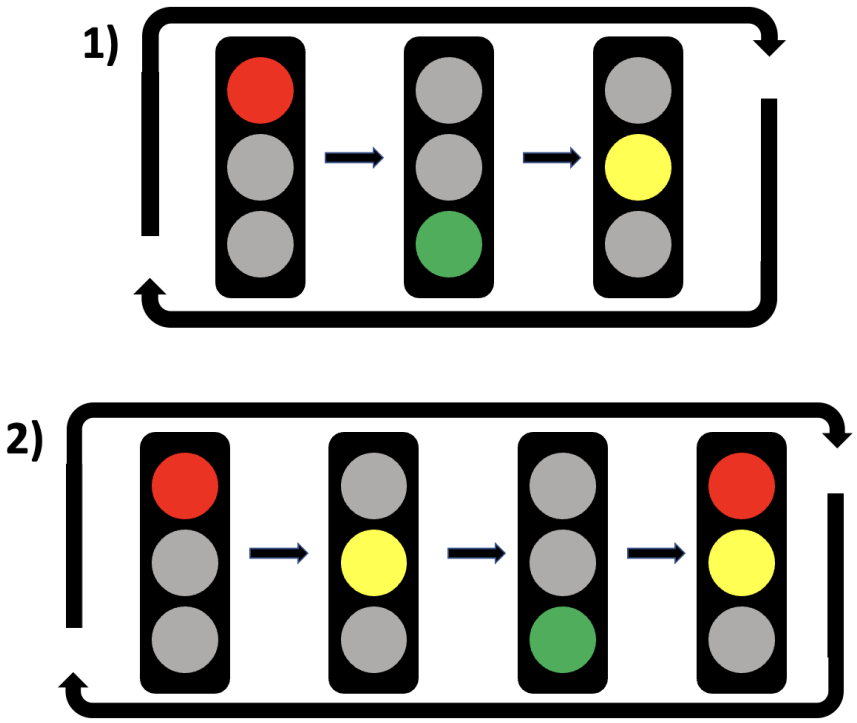

Figure 2.2: An Applied Example of Contrastive Explanations

An example of two unique traffic light systems that contrastive explanations can be applied. A contrastive explanation to effectively differentiate these two systems is $G(\neg red | X(green))$, which means "green comes next after red". Since this LTL statement is true for system #1, but not for system #2, this is an accurate contrastive explanation to describe their difference.

Approaches to automatic generation of contrastive explanations have adopted

SAT-based methods [49, 16, 28], in addition to Bayesian inference, as in the BayesLTL framework [39]. Our approach embraces the Bayesian inference strategy, using BayesLTL as a subprocess. Building on the strengths of LTL, BayesLTL [39] proposes a method to contrastively explain the differences between two sets of plan traces using LTL specifications. BayesLTL approaches specification learning as a Bayesian inference problem by building upon the fundamental Bayes theorem $P(\varphi|X) = \frac{P(\varphi)P(X|\varphi)}{\sum_{\varphi\in\Phi} P(\varphi)P(X|\varphi)}$. The goal of BayesLTL is to then infer $\varphi^* = argmax_\Phi P(\varphi|X)$, where $P(\varphi)$ represents the prior distribution over the hypothesis space, and $P(X|\varphi)$ is the probability of observing evidence (π_A, π_B), representing two unique sets of traces, given LTL specification φ. A probabilistic generative modelling approach is then used through the development and implementation of a prior function, a likelihood function, and a proposal function. BayesLTL's prior function is built to allow the system designer to incorporate their preferences. For example, the user might choose to specify a preference for "global" operators versus "until" operators. According to the system designer's parameter configuration, the prior function chooses a LTL template from a table of potential options (shown in Table 2.2) and decides upon the number of conjuncts and proposition instantiations for the various conjuncts. The likelihood function asserts contrast between the two sets of traces. By assuming that individual traces within sets are independent of each other, the likelihood of observing the input sets of traces within the satisfying (π_A) and non-satisfying (π_B) sets can be calculated via $P(X|\varphi) = \prod_{i=1}^{\pi_A} P(\pi_i|\varphi) \prod_{j=1}^{\pi_B} P(\pi_j|\varphi)$. Satisfaction checks are then conducted over all traces from both sets for the respective LTL specifications, which also provides robustness for outliers and noise. Finally, BayesLTL's proposal function approximates the true posterior and MAP estimates $\{\varphi^*\}$ by sampling from the true

posterior distribution and applies a Markov Chain Monte Carlo method to optimize template and LTL selection. In coordination with each other, these functions operate effectively together to generate relevant and interesting LTL specifications and differentiate a pair of trace sets.

Template	Meaning	Formula
global	p_i is true throughout the entire trace	$\mathbf{G}p_i$
eventual	p_i eventually occurs (may later become false)	$\mathbf{F}p_i$
stability	p_i eventually occurs and stays true forever	$\mathbf{FG}p_i \wedge \mathbf{G}(p_i \rightarrow (p_i \mathbf{WG} \neg p_i))$
response	If p_i occurs, p_j eventually follows	$\mathbf{G}(p_i \rightarrow \mathbf{XF}_{p_j})$
until	p_i has to be true until p_j eventually becomes true	$p_i \mathbf{U} p_j$
atmostonce	Only one contiguous interval exists where p_i is true	$\mathbf{G}(p_i \rightarrow (p_i \mathbf{WG} \neg p_i))$
sometime_before	If p_i occurs, p_j occurred in the past	$(p_j \wedge \neg p_i)\mathbf{R}(\neg p_i)$

Table 2.2: BayesLTL Templates [39]

The set of LTL templates embraced within BayesLTL [39], adopted within this thesis. When multiple propositions are used in a template, the condition is asserted for all propositions using conjunction. See Table 2.1 for an overview of LTL operators.

2.4 Trace Clustering in Business Process Mining

Research from the field of Business Process Mining has explored trace clustering as a means of reducing noise associated with data collected from realistic unstructured environments. Since multitudes of independent processes can exist within single event logs, the effectiveness of process mining techniques can benefit from separating out these unique elements and analyzing them individually. To cluster event logs and isolate unique processes, the field has adopted three broad classes of techniques, categorized as vector space clustering, context aware clustering, and model-based clustering.

By using the contents of event logs and organizing them according to their attributes, vector space clustering embraces traditional clustering algorithms to distinguish groups of traces. As an alternative to vector-space clustering, context-aware trace clustering adapts the control-flow handling of the information with traces. As an extension of context-aware clustering, R.P. and Aalst show thats context-aware clustering can also be employed in conjunction with vector-space clustering by using context-aware attributes as features [12].

2.5 Off-the-Shelf Clustering Evaluation Without Ground Truth

Evaluating the strength of discovered clusters represents a challenge when the ground truth of the problem is unknown. Additionally, due to the flexible nature of delineation, there may be multiple underlying ground truths. This means that there is nothing to directly compare the discovered clusters to and measure their degree of correctness, as one would extrinsically be able to do with most unsupervised learning problems. In order to address this issue, researchers have introduced intrinsic evaluation metrics to measure clusters relative to themselves and to each other. Perhaps the most insightful intrinsic evaluation metrics available are the silhouette coefficient [38], the Calinski-Harabasz index [15], and the Davies-Bouldin index [23].

The **silhouette coefficient** [38] is a tool that was introduced in 1990 to measure clustering strength by evaluating distances between the respective clusters according to the attributes of the items within. With silhouette values ranging from -1 to 1, items are scored higher if they are closer to the other items within their cluster (cohesion) and farther from the other items outside of their cluster (separation). With a(x) representing the average distance from x to all vectors within the same cluster

and b(x) representing the average distance from x to all vectors outside of the cluster, the silhouette value for each item can be measured via $s(x) = \frac{b(x)-a(x)}{max\{a(x),b(x)\}}$, and the silhouette coefficient can then be calculated according to $SC = \frac{1}{N}\sum_{i=1}^{N} s(x)$, where N is the quantity of samples and i is used to iterate over each instance. Values of the silhouette coefficient close to 1 are most desirable because this means that clusters are far apart from each other and clearly distinct. When the silhouette coefficient is closer to 0, clusters are very similar to each other and the distance between them is insignificant.

The **Calinski-Harabasz index** [15] measures the ratio of variance between within-cluster dispersion and between-cluster dispersion. The Calinski-Harabasz index is similar to the silhouette coefficient [38] in that it is designed to measure the strength of derived clusters based on cohesion versus separation, however the Calinski-Harabasz index does so by evaluating distances from centroids. To calculate this for a set of data E of size n_E and number of clusters k, between-cluster dispersion is derived via $B_k = \sum_{q=1}^{k}(c_q - C_E)(c_q - C_E)^T$, where c_q represents the center of cluser q, and c_E represents the center of E, and within-cluster dispersion is derived via $W_k = \sum_{q=1}^{k}\sum_{x\in C_q}(x - c_q)(x - c_q)^T$. The Calinski-Harabasz index can then be calculated according to $\frac{tr(B_k)}{tr(W_k)} \times \frac{n_E-k}{k-1}$. Similar to the silhouette coefficient, higher values of the Calinski-Harabasz index are most desirable beacause they represent clusters that are more dense and well separated.

Finally, the **Davies-Bouldin Index** [23] is designed to measure the average similarity between clusters by aggregating an evaluation of similarity between each cluster and its most similar counterpart. To do this, given a cluster (i), cluster diameter (s_i), and distance between cluster centroids i and j (d_{ij}), the similarity (R_{ij}) for each

cluster can be calculated via $\frac{s_i+s_j}{d_{ij}}$. The Davies-Bouldin index can then be calculated according to $\frac{1}{k}\sum_{i=1}^{k}\max_{i\neq j} R_{ij}$. Since the Davies-Bouldin index measures the similarity of clusters, lower scores are indicative of more distinct clusters representing stronger allocations of data points.

Since these clustering metrics are designed to evaluate generic clustering strength, their applicability can differ across unique datasets. This means that, while their adaptability can be a powerful tool for some problems, they can fail within others. Within our vector-space approach, we will assess the ability of these three metrics to evaluate the strength of our discovered clusters.

Chapter 3

Data Collection and Overview

3.1 Derivation of Evaluation Dataset

The derivation of an evaluation dataset is critical to the analysis of the proposed methods within this thesis. As a baseline for trace generation, classical automated planning domains represent an excellent tool. While standalone planning domains are inappropriate for widespread testing, since they are often designed to probe specific details of performance, the use of a diverse selection of multiple domains can lead to strong evaluations. Available by the International Planning Competition (IPC), the problem domain definition language (PDDL) domain/problem files of a wide selection of these domains are made accessible to be used as benchmarks [26]. We selected six of these classical planning domains for our evaluation, which include blocksworld, gripper, satellite, zenotravel, TPP, and rovers. In addition to these six domains, we also embraced three vocabulary sizes in $\{10, 15, 20\}$ for each. Designed to represent diversity in testing, the selected domains and vocabulary sizes will allow unique details of performance to be probed within both methodologies.

The injection of LTL ground truth into these domain/problem files was then explored using the LtlFond2Fond framework [17], similar to BayesLTL's [39] approach to dataset generation. Designed to address the diversity of real-world planning problems, LtlFond2Fond enables problems otherwise unsolvable by planning techniques to be adapted and solved. By evaluating temporally extended goals specified in LTL and compiling them into problem instances, LtlFond2Fond effectively returns modified PDDL domain/problem files that can be handled by automated planners; this allows known ground truth to exist within generated traces. While the injection of known ground truth is helpful to evaluate the ability of traditional contrastive explanation frameworks to differentiate traces, since our approach is designed to differentiate several sets of traces, an abundance of viable options may exist. Essentially, it does not matter if a specific ground truth is found or not, as long as a high-quality ground truth is identified. For that reason, we chose not to use LTL injection, as compared to BayesLTL.

With our set of 18 domain/problem PDDL files, the task of computing plans was assigned to a planner. Since the objective of data set acquisition is focused on achieving a specific quantity of plans, rather than maximizing plan quality, tools from the area of diverse planning were used for this task. While the goal of traditional cost-optimal planning is to identify a single best plan corresponding to the lowest possible cost, a requirement of a valid evaluation dataset is such that multiple plan traces are identified. An effective type of diverse planning that considers our needs is called diversity-bounded diverse planning. Originally introduced by Nguyen et al. through their LPG-d planner [50], diversity-bounded diverse planning generates k-plans constrained by a diversity metric, which forces plans to have a minimum quantity of

measured difference from the other generated plans. As an evolution of diversity-bounded diverse planning, Katz and Sohrabi introduced an improved planner with a novel post-processing procedure [36]. Katz and Sohrabi's planner offers a wide variety of diversity metrics that can be provided as a bound by the user; an effective bounding metric that the planner offers is called stability similarity [27, 21]. Stability similarity measures the ratio of actions appearing in both plans compared to the total quantity of actions. In order to create a set of k unique plans with a minimum quantity of measured difference, Katz and Sohrabi's planner, in conjunction with the stability similarity, was used as an effective tool.

Chapter 4

Vector-space Clustering using Temporal Logic Features

Our vector-space approach is designed to cluster and delineate traces by identifying context-aware temporal characteristics of traces, and using these features, along with traditional clustering algorithms, to establish sub-groups. Clusters are then delineated via the identification of conjunctive LTL features to distinguish the traces in each cluster from those in other clusters. Through this robust process, discrete time-series data can effectively be clustered and delineated according to temporal attributes.

We evaluate the effectiveness of this approach by establishing a sample set of cluster sets derived by this method. By measuring the LTL size of these resulting cluster definitions and assessing their ability to accurately describe clusters, we determine a high level of effectiveness. Next, we test the ability of our approximated delineation approach and compare the resulting explanations with respect to LTL size and accuracy. We then assess off-the-shelf clustering metrics as a possible measure of cluster strength, and determine that due to the uniqueness of our input data, these metrics

are of little use. Finally, we test the consistency of our approach by analyzing the similarity of clusters discovered via running this approach multiple times on the same data. Overall, we identify that clustering traces in vector-space according to context-aware attributes is a highly effective approach to clustering and delineating discrete time-series data.

4.1 Methodology

Our vector-space methodology begins by sampling subsets of traces from the larger set and identifying representative contrastive explanations between subsets via BayesLTL [39]. The resulting LTL statements are then used to create binary features that allow traces to be represented in vector-space via their satisfaction of each respective formula. Next, by applying traditional clustering algorithms, trace-vectors are clustered into groups of similarly behaving elements. Finally, to delineate each cluster, we identify a set of LTL statements that the all traces in the cluster satisfy, but no traces outside of the cluster satisfy. Alternatively for delineation, we offer an approximation approach which uses BayesLTL once again to differentiate clusters for more compact formulas, however the tradeoff becomes the strength of cluster definitions.

4.1.1 Overview

The identification of contrastive temporal patterns between randomized subsets of traces represents the first step in our clustering process. To discover these patterns, we introduce a procedure where random subsets are constructed, and the BayesLTL framework [39] is used to reveal contrastive explanations. Since BayesLTL generates valuable insightful formulas within the analysis of each random set, these statements

have the potential to pinpoint key similarities and differences between observations in the global set of traces. Embracing the discovered contrastive explanations, we establish a feature set of LTL specifications, by which individual traces are evaluated for entailment. The resulting matrix of binary information sorted by traces can then be clustered according to traditional unsupervised algorithms. To complete the delineation process, we then use the conjunction of LTL features to differentiate clusters. Alternatively, we also offer a variant explanation step which uses pairwise BayesLTL for a more compact, but approximate result. The output of our process is then the identified clusters, as specified by the LTL in our final step, along with the temporal specifications to delineate their selection. These steps are also outlined as follows:

1. Initialize Set of Traces to be Clustered
2. Establish LTL Formulas via Feature Discovery Process (See Figure 4.1)
3. Create Dataframe by Evaluating Traces for Entailment of LTL Features (See Figure 4.2)
4. Cluster Dataframe via Traditional Clustering Algorithms
5. Explain Identified Clusters Using LTL Feature Conjunctions (see Section 4.1.5) or Pairwise BayesLTL [39] (See Section 4.1.6)
6. Return Clusters and Contrastive LTL Definitions

4.1.2 Context-Aware Temporal Feature Generation

In order to provide traditional clustering algorithms information to differentiate groups of traces, we must first identify a feature set of relevant temporal specifications.

This feature set is generated to be relevant to the input set of traces by identifying contrastive temporal patterns amongst subsets of traces. To discover temporal patterns within subsets of trace data, we embrace a sampling process in coordination with the BayesLTL framework [39]. We begin by randomly sampling a subset of traces from the full set and conduct a balanced split to establish two equally sized subsets. We then label one of these subsets positive and the other negative, which satisfies the input requirements of BayesLTL. By running BayesLTL on these subsets, we are then provided with contrastive explanations designed to differentiate the sets of traces. By default, BayesLTL returns the 10 best specifications. We store all of these discovered specifications before randomly sampling a new subset of traces and repeating this process; in future iterations however, we only keep specifications that are novel to our stored set of formulas. Once a desired quantity of specifications is fully generated, our procedure returns the set of unique formulas. An overview of this process can be viewed in Algorithm 1 and in Figure 4.1.

Since feature quality is typically subjective and cannot be quantitatively evaluated, we rely on two important considerations to remain confident in this process to generate relevant features. The first consideration is the proven ability of reasonably sized sample sets to represent the parent set, which we demonstrate in Section 5.2.4. We identify that samples as small as 2% are capable of representing the global set within some of our evaluation domains, and samples as small as 10% within all of our evaluation domains. The second element that we consider is the significant strength of BayesLTL to differentiate subsets of traces. Factoring in both of these considerations, this means that each LTL specification within the feature set should be capable of

precisely describing relevant variation within traces from the input set. As the quantity of features is increased, the ability of the feature set to extract meaningful insight into the temporal behaviour of the input set should increase drastically.

Algorithm 1: LTL Feature Discovery Process

Input: Set of traces π_n
Parameters: Statement limit $imit$, Sample proportion p
Output: Feature set $\{\varphi\}$ of relevant LTL formulas

1: $formulas \leftarrow [\,]$
2: **while** $(len(formulas) < limit)$ **do**
3: $\pi_{sample} \leftarrow$ random sample set of size $p|\pi_n|$ from π_n
4: $\pi_+, \pi_- \leftarrow$ random balanced split of π_{sample}
5: **for** values of φ resulting from BayesLTL(π_+, π_-) **do**
6: **if** $(|formulas| < limit)$ **and** (φ not in $formulas$) **then**
7: Append φ to $formulas$
8: **end if**
9: **end for**
10: **end while**
11: **return** $formulas$

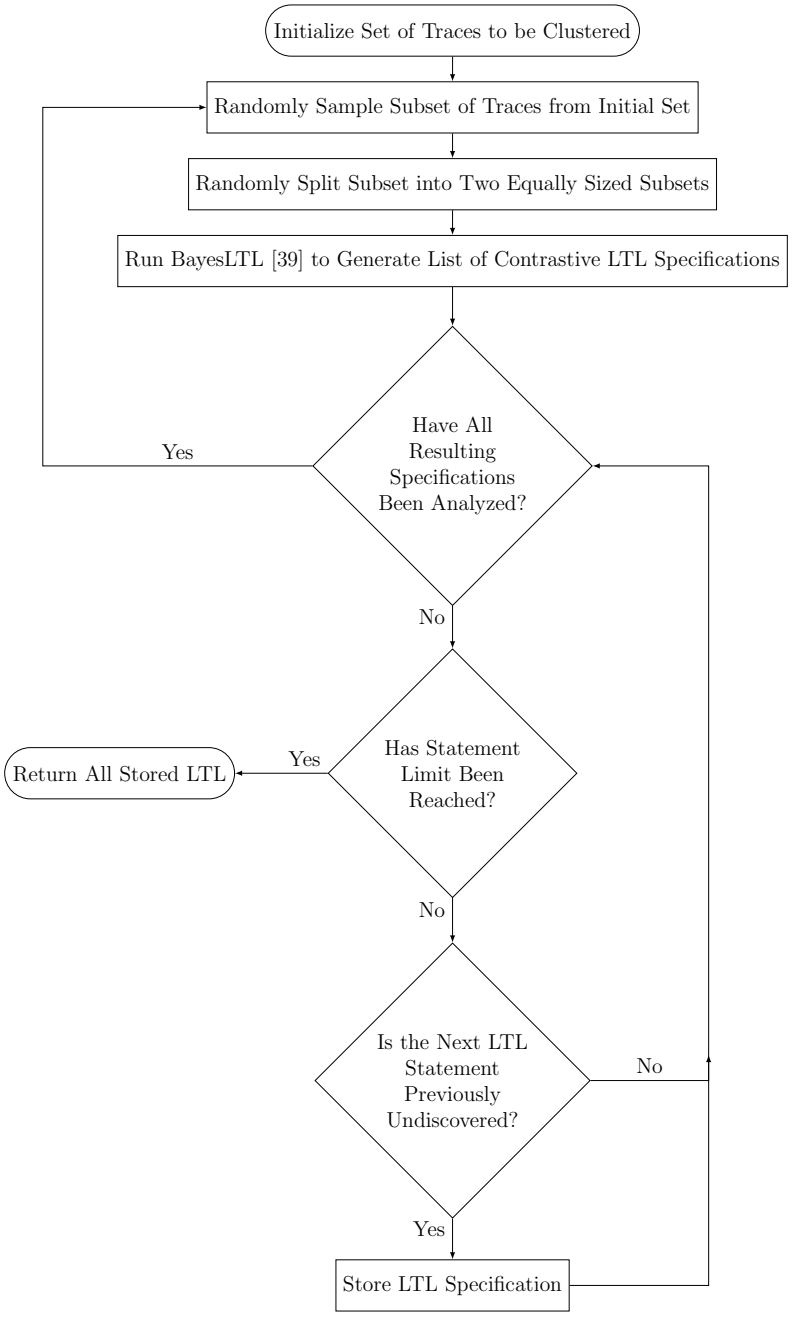

Figure 4.1: Flow Diagram of LTL Feature Set Discovery Process

Flow diagram of our vector-space feature discovery process. By conducting random samples from the parent set and identifying contrastive explanations from representative subsets, this process facilitates the discovery of relevant LTL characteristics of input traces.

4.1.3 Feature Mapping

Using our discovered set of LTL formulas, we embrace each unique specification as a feature to describe traces in the global set. Again using functionality from BayesLTL [39], we conduct entailment analysis between every trace and every formula; this is a very computationally simple calculation which is completed fast, even for very large datasets. This results in a matrix of size $|\pi| \times |\varphi|$ containing binary data, where rows represent individual traces, and columns represent LTL features; see Figure 4.2 for a visual representation of this data structure. While this transformed representation of the input set of traces may on the surface appear equally disorderly to raw trace data, underlying temporal patterns have now been exposed for similarity distance to be measured.

	φ_0	φ_1	φ_2	φ_3	$\varphi_{...}$
π_0	$\pi_0 \overset{?}{\models} \varphi_0$	$\pi_0 \overset{?}{\models} \varphi_1$	$\pi_0 \overset{?}{\models} \varphi_2$	$\pi_0 \overset{?}{\models} \varphi_3$	$\pi_0 \overset{?}{\models} \varphi_{...}$
π_1	$\pi_1 \overset{?}{\models} \varphi_0$	$\pi_1 \overset{?}{\models} \varphi_1$	$\pi_1 \overset{?}{\models} \varphi_2$	$\pi_1 \overset{?}{\models} \varphi_3$	$\pi_1 \overset{?}{\models} \varphi_{...}$
π_2	$\pi_2 \overset{?}{\models} \varphi_0$	$\pi_2 \overset{?}{\models} \varphi_1$	$\pi_2 \overset{?}{\models} \varphi_2$	$\pi_2 \overset{?}{\models} \varphi_3$	$\pi_2 \overset{?}{\models} \varphi_{...}$
π_3	$\pi_3 \overset{?}{\models} \varphi_0$	$\pi_3 \overset{?}{\models} \varphi_1$	$\pi_3 \overset{?}{\models} \varphi_2$	$\pi_3 \overset{?}{\models} \varphi_3$	$\pi_3 \overset{?}{\models} \varphi_{...}$
$\pi_{...}$	$\pi_{...} \overset{?}{\models} \varphi_0$	$\pi_{...} \overset{?}{\models} \varphi_1$	$\pi_{...} \overset{?}{\models} \varphi_2$	$\pi_{...} \overset{?}{\models} \varphi_3$	$\pi_{...} \overset{?}{\models} \varphi_{...}$

Figure 4.2: Vector-Space Matrix of Traces and Temporal Features

Matrix representing traces and their respective satisfaction of LTL features. Formulas are represented on the horizontal axis, and traces from the parent set are represented on the vertical axis. This matrix is later used as input for traditional clustering algorithms.

4.1.4 Cluster Discovery

Given our binary matrix representing traces and their respective entailment of LTL specifications, we now have the flexibility to identify clusters according to traditional

unsupervised algorithms. We allow the selection of algorithm to be chosen as a parameter, currently offering agglomerate hierarchical clustering and k-means, but any generic algorithm is feasible. These current algorithmic offerings allow the user to choose the quantity of clusters via k in k-means, or let the algorithm optimally determine k as is the case for agglomerate hierarchical clustering.

4.1.5 Delineation via Conjunct LTL Features

As the final step in our approach, the discovered clusters are compared with the LTL features to find minimal length sets of specifications that clusters uniquely entail. We begin this step by first identifying which LTL features are relevant to which clusters by collecting a set of features for each cluster that all traces in that cluster entail. Since all traces in multiple clusters may entail the same formula, we must then use the conjunction of relevant features to differentiate clusters. We formulate this as a search problem, where the objective for each cluster is to discover a minimal length set of features that is conjunctively entailed only by that cluster. Our full search space for each cluster becomes $2^n - 1$ in size, where n represents the quantity of features relevant to the given cluster. Fortunately, this search space can be generated as an ordered iterable with set sizes growing with additional iterations. This means that when iterating search in order, the minimal size goal is known to be achieved with the first identification of a distinct set of LTL. Additionally, we allow the search space to be limited via parameterization to a maximum acceptable set size, which drastically reduces the size of the search space to $\sum_{i=0}^{max\_conj} \binom{n}{i}$. Limiting the search space also makes practical sense due to the loss in specification value that occurs due to overfitted explanations with larger set sizes. When a set of LTL cannot be found

within the parameterized reasonable search limits, this is acknowledged within the output, and additional iterations of feature discovery are recommended to identify features with more robust fits. Finally, in completion of search for conjunct LTL sets for each cluster, the resulting explanations are returned as cluster definitions.

While the use of conjunct LTL features allows clusters to be perfectly explained within reasonable search limits in most of our investigated domains, associated complexity with this approach means that this may not always be the case. Additionally, conjunct LTL has the ability to lead to larger explanations, which could represent overfitting in some problem configurations. To address this, we also provide an approximation approach, where more compact explanations can be discovered at a more predictable level of complexity, however accuracy becomes the cost.

4.1.6 Delineation Approximation with BayesLTL

As an approximation of cluster definitions, an approach using BayesLTL [39] to compare discovered clusters can also be used. While this approximation approach tends to lead to more compact explanations, the trade-off becomes accuracy of trace allocations. Since BayesLTL is only designed to contrast two predefined input sets, using the framework to contrast multiple predefined sets reduces the soundness of resulting explanations. To expand the capacity of BayesLTL to account for more than two defined candidate groups of traces, we consider two unique methods, one-versus-all and pairwise analysis.

Our one-versus-all method approaches comparison by simply concatenating all sets of traces into a single set excluding a selected set-of-interest. BayesLTL's traditional functionality is then used to infer relevant contrastive explanations for those two sets.

The resulting specification is stored, and this process is repeated for each cluster. This results in k evaluations of BayesLTL being conducted to achieve k contrastive explanations, where each explanation describes one cluster. As an example of this one-versus-all approach, if three clusters π_A, π_B, and π_C are discovered within π, we derive BayesLTL explanations for all pairs in $\{(\pi_A, \pi_B \cup \pi_C), (\pi_B, \pi_A \cup \pi_C), (\pi_C, \pi_A \cup \pi_B)\}$. This results in the following cluster definitions: $\pi'_A = \{\pi : \pi \models \varphi_{\pi_A, \pi_{\neg A}}\}$, $\pi'_B = \{\pi : \pi \models \varphi_{\pi_B, \pi_{\neg B}}\}$, and $\pi'_C = \{\pi : \pi \models \varphi_{\pi_C, \pi_{\neg C}}\}$.

Alternatively, our pairwise method approaches comparison by inferring LTL specifications via BayesLTL from all possible unique pairs of clusters. This results in $\binom{k}{2}$ specifications, where each cluster is defined by its entailment (or lack of entailment) of $k-1$ specifications. As an example of this pairwise approach, we derive BayesLTL explanations for all pairs in $\{(\pi_A, \pi_B), (\pi_A, \pi_C), (\pi_B, \pi_C)\}$. This results in the following cluster definitions: $\pi'_A = \{\pi : \pi \models \varphi_{\pi_A, \pi_B} \wedge \varphi_{\pi_A, \pi_C}\}$, $\pi'_B = \{\pi : \pi \models \neg\varphi_{\pi_A, \pi_B} \wedge \varphi_{\pi_B, \pi_C}\}$, and $\pi'_C = \{\pi : \pi \models \neg\varphi_{\pi_A, \pi_C} \wedge \neg\varphi_{\pi_B, \pi_C}\}$. π'_A then represents the traces that satisfy both $\varphi_{\pi_A, \pi_B}$and φ_{π_A, π_C}, π'_B represents the traces that satisfy φ_{π_B, π_C}, but not φ_{π_A, π_B}, and π'_C represents the traces that dissatisfy both φ_{π_A, π_C} and φ_{π_B, π_C}.

4.1.7 Parameterization

k: Specified Number of Clusters

The k parameter allows the user to determine the quantity of clusters to be discovered. This parameter also currently determines the clustering algorithm to be used; if k is not specified, agglomerate hierarchical clustering is selected, however if k is specified, k-means is used.

limit: Quantity of LTL-Specification Features

The *limit* parameter determines the quantity of binary temporal features to be identified and used within the established vector-space. An example of a temporal feature from the Blocks domain could be *eventual* : (*clear object i*), which can be true for some traces, but false for others. Lower values of *limit* may lead to underfitting due to insufficient information for cluster discovery. Larger values of *limit* provide the clustering algorithm with greater insights, however search time for features will be proportionately extended. The default value of the *limit* parameter is set at 200 because we identify values exceeding this threshold to be commonly associated with successful explanations of all clusters using conjunct features. This will be discussed further in Section 4.2.1.

sample_prop: Sample Proportion to be Drawn from Parent Set

The *sample_prop* parameter determines the proportion of traces to be randomly sampled when subsets are created. For example, if a set containing 500 traces is clustered and the *sample_prop* parameter is set to 20%, a random subset of 100 traces will be sampled at each iteration, resulting in contrastive explanations being generated from two randomized sets of 50 traces. The default value of *sample_prop* is set at 10%, which is supported by our evaluation in Section 5.2.4, where we evaluate the representative strength of various sample sizes. While larger sample sizes tended to provide greater representative ability, the magnitude of this effect was found to be very small-scale, so a small sample size of 10% represents an opportunity for runtime savings; when higher quality features are desired, runtime can be used as a tradeoff via higher values of *sample_prop*.

max_conj: Maximum Set Size of Conjunctive LTL in Delineations

The maximum quantity of LTL conjunctions accepted within cluster explanations. More importantly, this parameter limits the search space of LTL combinations. When this parameter is inactive, the ordered explanation search space for each cluster is $2^n - 1$ in size, where n represents the quantity of LTL features that all traces in the given cluster entail. Since this search space can quickly grow enormously large, limiting its size is typically very important. The max $\_conj$ parameter therefore allows the search space to be limited to $\sum_{i=0}^{max\_conj} \binom{n}{i}$ in size. The default value for this parameter is set at 3 to accommodate reasonable search limits in robust domains.

4.2 Evaluations and Results

Our evaluation begins by establishing a sample set of cluster sets identifying when applying this procedure to our sample domains. We then use this sample set to measure the general effectiveness of this approach using both conjunctive LTL discovery, as well as approximated cluster definitions. Next, we test the ability of off-the-shelf clustering algorithms to describe our discovered clusters, followed by an analysis into the discovery rate of LTL features. Finally, we explore the consistency of our proposed approach by measuring the similarity of clusters discovered from our sample sets.

4.2.1 General Effectiveness of Clustering/Delineation Process

To conduct a general analysis into the strength of our clustering and delineation approach, we use our six sample domains and three vocabulary sizes across multiple test iterations. For each of the 18 domain/vocabulary configurations in our evaluation

dataset, we cluster and delineate their respective traces 10 separate times to gain a representative overview. Within this analysis, we use k-means clustering with a value of 8 for k to allow for a uniform evaluation across all dataset configurations. After identifying conjunct sets of LTL to represent cluster definitions, we measure clustering performance according to LTL size and search success.

We measure LTL size according to Gaglione et al.'s definition using the quantity of unique subformulas, discussed further in Section 2.2. Formulas that are very small may be indicative of underfitting behaviour, while formulas that are very large may represent overfitting. Seeking robust fit between mappings of formulas and sets of traces, desirable formulas will be of a subjectively reasonable size, relative to the domain, to adequately describe relevant temporal characteristics of problems. Next, we define search success as the proportion of clusters accompanied by distinct LTL definitions. For example, if 8 clusters are identified (as specified by k), but LTL explanations are found for only 6 clusters, search success would be 75%. Higher values of search success indicate more complete definitions of cluster-sets. By aggregating these performance metrics for each of our investigated configurations, we assess our algorithm's general ability to accurately and precisely differentiate clusters of traces.

In analyzing LTL size among our sample sets, we evaluate the size of both the temporal features used to cluster traces and the formulas used to explain the resulting clusters. Table 4.1 describes the distribution of LTL size within cluster explanations and feature sets from our sample domains and vocabulary sizes.

Domain	$\|V\|$	LTL Size Cluster Explanations (10 Combined Sets of 8 LTL)					LTL Size Vector Space Features (10 Combined Sets of 1000 LTL)				
		Min	Med	Max	μ	σ	Min	Med	Max	μ	σ
Blocks	10	2	6	19	8.588	4.572	2	6	31	8.644	4.638
	15	5	6	19	9.038	4.468	2	5	31	7.432	3.965
	20	2	2	17	3.186	2.688	2	5	21	5.345	2.016
Gripper	10	2	5	16	5.287	3.082	2	6	25	7.334	3.556
	15	2	5	19	5.588	3.578	2	6	31	6.962	3.420
	20	2	5	19	5.138	2.841	2	6	25	6.780	3.390
Rovers	10	2	6	13	7.000	3.917	2	6	31	8.521	4.069
	15	2	6	25	7.237	4.540	2	7	31	9.353	4.203
	20	2	6	19	6.500	3.814	2	6	25	8.240	4.320
Satellite	10	5	6	19	8.250	3.814	2	13	31	11.659	3.914
	15	2	6	19	6.987	3.123	2	11	37	10.497	4.291
	20	2	6	13	6.388	3.124	2	6	25	7.894	4.055
TPP	10	2	6	13	5.450	3.511	2	6	31	8.802	4.495
	15	2	5	19	5.688	4.145	2	6	31	7.957	4.294
	20	2	5	19	5.175	3.438	2	6	31	7.828	4.396
Zeno-Travel	10	2	6	19	7.850	3.911	2	7	25	9.181	4.161
	15	2	6	13	6.612	3.366	2	6	25	8.309	4.096
	20	2	6	17	5.775	3.268	2	6	25	7.834	4.074

Table 4.1: Size Analysis of Discovered LTL Describing Vector Space Clusters

Analysis of LTL specification size $|\varphi|$ distribution using our LTL conjunction approach. LTL size is measured according to the quantity of unique subformulas, as defined by Gaglione et al. (Clusters lacking definitions are excluded from this calculation). Each row reports the minimum, median, maximum, mean, and standard deviation of $|\varphi|$ within the 10 combined sample sets of cluster explanations and combined feature sets of each respective domain/vocabulary configuration.

By presenting the minimum, median, maximum, mean, and standard deviation of specification size from each configuration, an understanding of size tendency can be inferred. Addtionally, for a better understanding of LTL size using BayesLTL's [39] templates, example formulas (from our tree-based approach) and their respective sizes can be viewed in Appendix B. An interesting observation within Table 4.1 is the larger size of the features used to establish clusters, as compared to the LTL used to explain clusters; in almost all domain/vocabulary size configurations, the mean and median LTL size are larger for vector space features versus explanation LTL. This is likely explained by the idea that vector space features probe higher level temporal characteristics of smaller sample sets, where stronger fits can be discovered. Alternatively, when evaluating features that are relevant to all traces within a cluster, the more generalized formulas will be used.

Since search success will be highly depend on the quantity of features derived from the feature discovery process and used in clustering, we evaluate search success with respect to a variety of feature-set sizes. For each of our six experimental domains and three vocabulary sizes, we begin by establishing 10 feature sets of 1000 unique specifications through our feature discovery process. For each feature set, we then take the first specifications for values in $\{10n : n \in Z^+, i < 101\}$ and cluster and delineate traces according to those features, repeating 10 times. We then measure search success amongst these clusters, average the results amongst feature sizes, and plot search success with respect to the quantity of features used.

Examples of the resulting cluster-set-defining formulas from each domain can be viewed in Appendix A. An interesting observation within these example formulas is the fact that conjunction is rarely required. Within examples from our simulated

evaluation domains, single elements of feature sets were capable of distinguishing clusters most of the time, which is a strong indication of the effectiveness of our feature discovery process. This also represents a mitigating argument against complexity limitations, since single features appear first in the search order.

Figure 4.3 presents search success versus quantity of features for the 18 configurations in our evaluation dataset. For all configurations, with the exception of Blocks20, search success appears to approach 100% as the quantity of features grows. This result provides a very strong positive indication of success for our vector-space clustering method because it effectively demonstrates the ability of our approach to consistently discover and accurately describe clusters. To address Blocks20, it appears that search success tends to stagnate around 75% for most feature sizes. While this may appear problematic, 75% still indicates successful explanations for 6/8 of the discovered clusters. Additionally, our selection of 8 for k was an arbitrary decision for experimental consistency; perhaps there are only 6 natural clusters within the Blocks20 dataset.

These plots are also an important tool to assist in the selection of how many feature to use when selecting the *limit* parameter. It appears that in most domains, search success is consistently near 100% when the quantity of features exceed around 200. For this reason, we have also set the default value of the *limit* parameter to 200 to minimize feature discovery runtime, but maximize expected results. Overall, since search success appears to approach 100% for 17/18 configurations, we can conclude that our vector-space clustering method is capable of performing very strongly to cluster and delineate traces as intended.

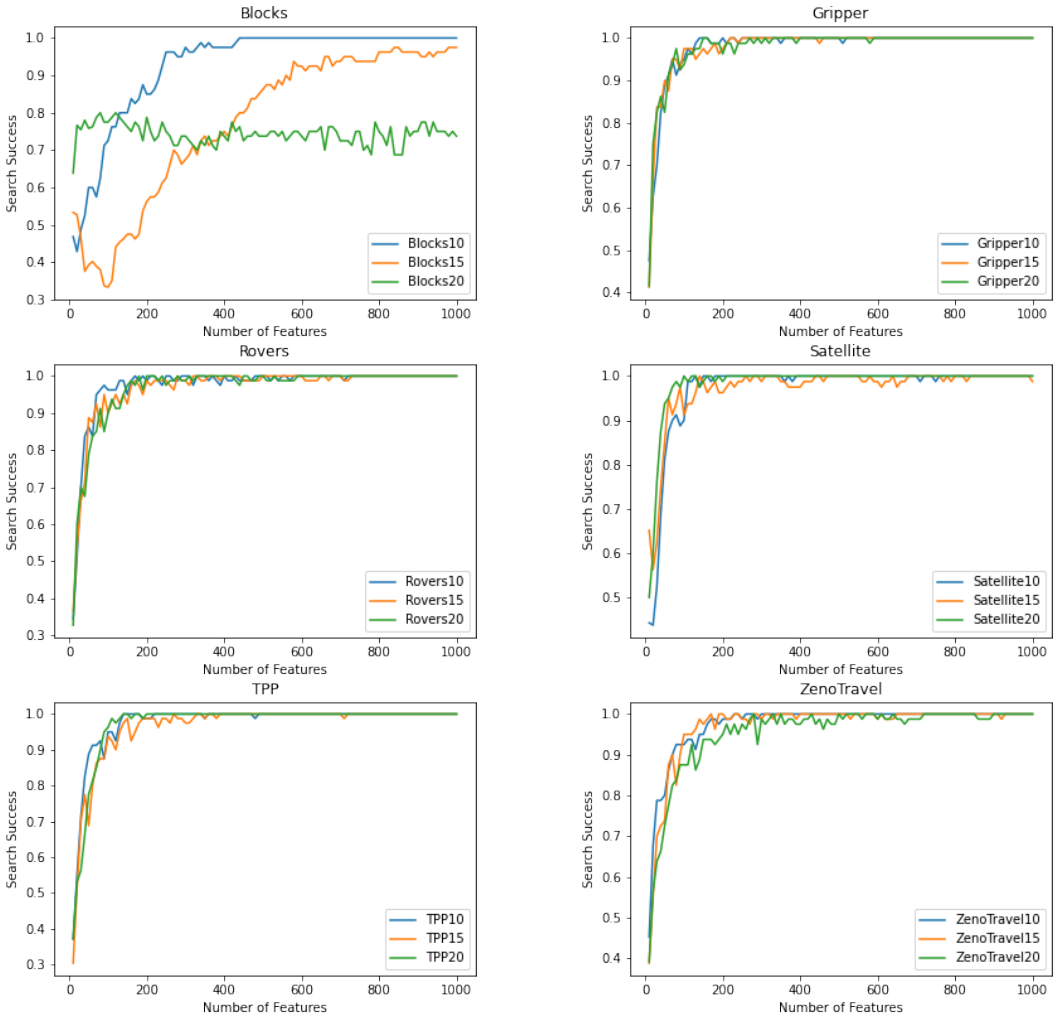

Figure 4.3: Conjuctive LTL Search Success Versus Quantity of Features Plots

Proportion of traces accurately and completely described by identified LTL features with respect to the quantity of features used in clustering.

4.2.2 General Evaluation of Delineation Approximation Approach

Using the same clusters discovered previously in Section 4.2.1, we measure the ability of our delineation approximation approach to explain clusters. Additionally, since we propose two slightly different approaches to approximate cluster definitions, we conduct this evaluation on both approaches to gather an understanding of each method's

strengths and weaknesses.

In order to assess the strength of these approaches, we use LTL size, in addition to the introduction of a new metric, delineation accuracy. Since our clustering procedure is independent from cluster approximation, there exists a threat of incompatibility, where clusters could be discovered, but not described with full coverage. Additionally, the possibility exists for traces to co-exist within multiple clusters, which we also view as an error. To measure the size of this potential error, we establish a simple accuracy metric, which quantifies the strength of the discovered differentiation by counting the number of traces that do not perfectly fit into single formulae-defined clusters. This error metric is formally defined as:

$$\frac{|\{\pi : (\sum_{i=0}^{k} \pi \models \varphi_i) \neq 1\}|}{|\pi|} \tag{4.1}$$

By evaluating and aggregating delineation accuracy and LTL size for the resulting explanations of both the one-vs-all and pairwise approximation methods, the ability of these approaches to define clusters was investigated.

In Table 4.2 LTL size measurements can be viewed for both the one-vs-all and pairwise approximation approaches. From these measurements, it is clear that pairwise explanations tend to be smaller in most domains, measured by median and mean. Standard deviation LTL size is also demonstrated to be smaller for the pairwise approach, leading to more consistent sizes of formulas. Interpreting these results, the pairwise approximation approach should be preferred over the one-vs-all approach when it comes to compact and consistent formula sizes. Another relevant comparison is the LTL size tendency of approximated explanations versus LTL conjunction

Domain	$\|V\|$	LTL Size Pairwise Approximation (10 Combined Sets of 28 LTL)					LTL Size One-vs-All Approximation (10 Combined Sets of 8 LTL)				
		Min	Med	Max	μ	σ	Min	Med	Max	μ	σ
Blocks	10	2	2.0	19	3.525	2.084	2	5.5	13	4.938	2.931
	15	2	6.0	16	5.700	2.869	2	3.5	13	4.188	2.682
	20	2	2.0	19	3.043	2.155	2	5.0	37	5.237	5.323
Gripper	10	2	2.0	13	3.725	2.110	5	6.0	19	7.688	3.794
	15	2	2.0	17	3.464	2.152	2	6.0	31	8.00	5.243
	20	2	2.0	25	3.468	2.609	2	6.0	13	7.112	2.873
Rovers	10	2	5.0	13	4.218	2.412	2	7.0	25	10.225	7.165
	15	2	5.5	19	5.075	3.499	2	6.5	25	9.775	5.528
	20	2	5.0	19	4.714	3.401	2	6.5	25	8.225	5.253
Satellite	10	5	6.0	25	6.429	2.584	5	6.0	13	6.075	0.792
	15	2	5.0	13	4.218	2.019	5	6.0	16	6.713	2.301
	20	2	2.0	6	3.200	1.836	2	6.0	19	6.938	4.107
TPP	10	2	2.0	13	3.432	2.562	2	6.0	37	6.338	4.757
	15	2	5.0	19	4.682	3.491	2	6.0	25	6.763	3.698
	20	2	2.0	13	2.725	1.798	2	6.0	13	5.700	1.687
Zeno-Travel	10	2	6	19	5.279	3.620	2	8.0	25	10.312	5.636
	15	2	6.0	19	5.029	2.762	2	7.0	37	8.988	5.806
	20	2	5.0	19	4.571	3.271	2	7.0	19	7.525	3.697

Table 4.2: LTL Size Analysis of Approximation Approaches

Analysis of LTL specification size $|\varphi|$ distribution using our one-vs-all and pairwise approximation approached. LTL size is measured according to the quantity of unique subformulas, as defined by Gaglione et al.. Each row reports the minimum, median, maximum, mean, and standard deviation of $|\varphi|$ within the 10 combined sample sets of cluster explanations and combined feature sets of each respective domain/vocabulary configuration.

explanations. Comparing approximation LTL measurements in Table 4.2 to LTL conjunction measurements in Table 4.1, we see a consistent size reduction in approximated explanations. This result makes sense since BayesLTL [39] is designed to identify one compact formula, whereas our conjunction-based approach combines

multiple formulas together, leading to larger sizes.

Delineation accuracy can be viewed for both our pairwise analysis and one-versus-all analysis in Table 4.3, where the average and standard deviation error rates are reported.

Domain	$\|V\|$	Delineation Error Rate			
		One-vs-All		Pairwise	
		μ	σ	μ	σ
Blocks	10	0.753	0.093	0.655	0.211
	15	0.390	0.372	0.132	0.261
	20	0.637	0.390	0.724	0.417
Gripper	10	0.564	0.206	0.168	0.259
	15	0.379	0.334	0.123	0.147
	20	0.441	0.424	0.190	0.203
Rovers	10	0.641	0.326	0.118	0.196
	15	0.735	0.169	0.324	0.200
	20	0.814	0.140	0.525	0.203
Satellite	10	0.888	0.262	0.336	0.215
	15	0.725	0.242	0.061	0.130
	20	0.900	0.096	0.000	0.000
TPP	10	0.723	0.240	0.371	0.197
	15	0.730	0.178	0.659	0.103
	20	0.555	0.147	0.294	0.206
Zeno-Travel	10	0.772	0.197	0.373	0.152
	15	0.836	0.126	0.338	0.222
	20	0.797	0.191	0.798	0.098

Table 4.3: Delineation Error of Approximated Explanation Approach

Average and standard deviation delineation error using one-versus-all and pairwise evaluation methods to differentiate clusters. Error is measured by the quantity of traces that do not fit into exactly one cluster divided by the total quantity of traces. Delineation error is formally defined in Equation 4.1.

As is clear by the measurements presented in this table, approximated explanations can often fail to provide a complete representation of clusters. That being said, the ability of these approximation approaches to accurately define clusters appears

to be differ significantly by domain; in some domain configurations, approximated explanations are demonstrated as highly effective, while in others, unproductive. For instance, using the pairwise approach, the Satellite20 domain measures 0% average delineation error, but in ZenoTravel20, almost 80% average error is recorded. This indicates that the success of our approximation process is largely dependent on the uniqueness of input problems themselves. While this lack of success is clearly problematic in some domains, since delineation error is a reliable metric that is easy to measure and comprehend, it can always be embraced to ensure clustering relevance after all evaluations. Additionally, when analyzing the source of this error further, we find that in many cases, error is derived mostly from one or two faulty contrastive explanations, whereas the explanations of other clusters are more sound. For example, within one of our samples from the rovers10 domain, the discovered cluster sizes were {20, 32, 7, 10, 12, 8, 6, 5} prior to redefining clusters via LTL, however, they became {19, 32, 1, 10, 12, 95, 6, 93} post-explanation. In this example, the measured delineation error is 90%, however, it is clear that most of the error is derived from two of the eight clusters. This means that, although delineation error is effective in measuring absolute error, in many cases, the LTL discovered to redefine clusters may be more effective than the delineation error metric presents them to be.

When evaluating error sources within our pairwise analysis, an interesting observation is that across all evaluation configurations, no traces appear in multiple clusters, meaning 100% of the measured error is derived from traces that fit into no clusters. Alternatively, within our one-versus-all analysis, error is observed consistently from both sources, with the dominant source being traces existing in multiple clusters for most configurations. This means that although an assortment of traces are left out

within most configurations using the pairwise approach, the identified clusters are very strongly defined, as opposed to the one-vs-all approach where cluster definitions are much weaker. Similar to our one-vs-all approach, we also identify a potential misrepresentation that sometimes occurs, where delineation error is derived from one or two clusters, and all other clusters are accurately defined.

Comparing our two approximation approaches to cluster explanations, it appears that the pairwise method should be preferred over the one-vs-all method. This is because in most domain/vocabulary configurations, resulting formulas tend to be smaller in size, and lesser delineation error is reported when using the pairwise approach. Additionally, the increased strength of cluster definitions derived from the pairwise method demonstrates it as the superior approximation option. When comparing our approximation-based approaches to our conjunction-based approach, the tradeoff appears to be accuracy versus formula compactness. Overall, delineating clusters via our approximation-based approaches can at times be strong and capable, but can also be unreliable in terms of accuracy. For accuracy purposes, our LTL conjunction method should be preferred, however for conciseness or predictable complexity, the approximation method may be the better option.

4.2.3 Can Off-the-Shelf Clustering Metrics Accurately Describe Cluster Strength?

While off-the-shelf clustering evaluation metrics (requiring no ground truth) are effective in many contexts, they are designed to generalize the evaluation of clustering quality. This means that their effectiveness can sometimes be limited within specialized applications. As such, we are interested in studying their effectiveness as a

measurement tool for the discovered clusters within our described methods. As the most common metrics, we apply our analysis to the silhouette coefficient [38], the Calinski-Harabasz index [15], and the Davies-Bouldin index [23]. These metrics are discussed in greater depth Section 2.5.

To evaluate the applicability of these metrics, we propose the assumption that clustering quality should improve with the amount of information provided as input to the clustering process, at least initially. This amount of information is reflected by the number of LTL features discovered prior to clustering, by which traces are evaluated for entailment. When more information is provided to characterize the behaviour of traces, clustering algorithms should be more equipped to discover like-groups. It is important to note, however, that there may also exist a tradeoff that occurs between improved insight and introduced noise. While every new feature may on-the-surface be expected to improve clustering quality, it is possible that introduced noise could also impair performance. That being said, if effective, we should expect evaluation metrics to initially improve with the quantity of LTL features used, though they may later tamper off.

To test this, we used the clusters derived in Section 4.2.1 to measure clusters according to off-the-shelf methods using a variety of progressively larger feature sizes. For 10 sets of clusters derived from feature set sizes in $\{10n : n \in Z^+, i < 101\}$, quality was measured according the silhouette coefficient, the Calinski-Harabasz index, and the Davies-Bouldin index, as discussed in Section 2.5. These measurements are plotted in Figure 4.4.

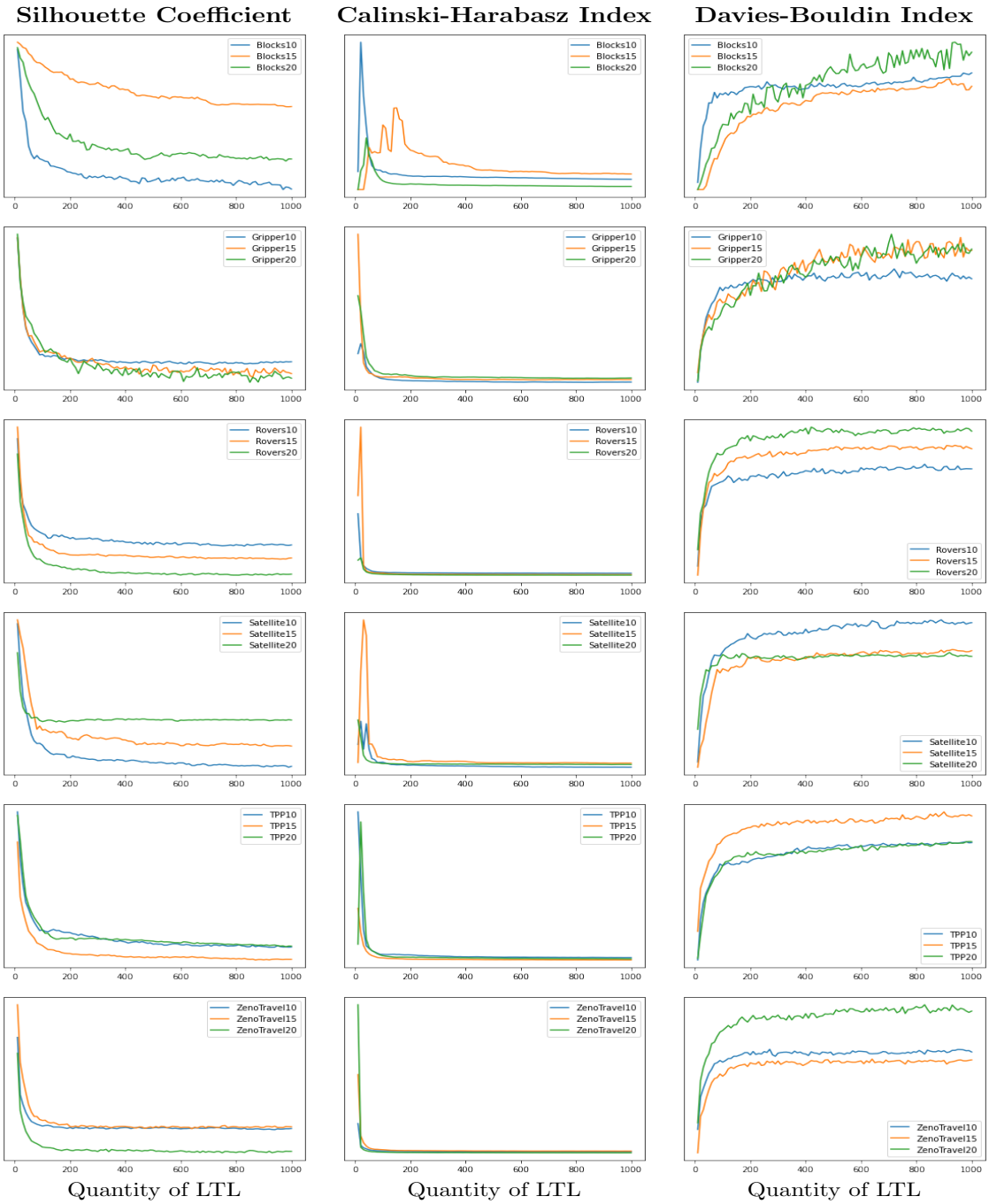

Figure 4.4: Evaluation Plots of No-Ground-Truth Clustering Metrics

Off-the-shelf metrics vs quantity of LTL features used in clustering. For each feature quantity in $\{10n : n \in Z^+, i < 101\}$, 10 unique feature sets are established, and metrics across feature sets are averaged for each domain/vocabulary size configuration.

In Figure 4.4, we consistently observe declining measurements of the silhouette coefficient and Calinski Harabasz index with increased feature quantities. We also observe increasing measurements of the Davies Bouldin index. Each of these observed trends appear to diminish marginally and occur for every domain/vocabulary size configuration within our evaluation dataset. While higher values of the silhouette coefficient and Calinski Harabasz index indicate higher quality clustering, and the same is true for lower values of the Davies Bouldin index, our results represent a trend towards the opposite for all three metrics.

Due to the consistent downward trend in measurements of clustering quality presented by these metrics with advances in feature information, it is unlikely that they depict an appropriate assessment to inform feature selection. These results are likely explained by the idea that there are probably several similarly correct allocations of clusters, and the proposed selection within metric calculation is simply not meaningful in the larger context. Additionally, when we increase the quantity of features, the quantity of reasonable cluster allocations likely increase significantly in parallel, which further harms the scores of these metrics. This does not mean that the identified clusters are weak in any way, simply that there are an abundance of other correct solutions. From these results, we infer that off-the-shelf clustering evaluation methods are likely unsuitable for measuring the strength of clusters, considering the uniqueness of the data we are using.

4.2.4 Approximating Discovery Rate of Temporal Features

When generating feature sets within Section 4.2.3, we also recorded the quantity of new LTL specifications discovered at each iteration; we define new specifications at

a given iteration as the set of specifications discovered at the current iteration that have not been previously discovered in past iterations. The quantity of new specifications discovered per iteration is of interest because it allows us to better understand characteristics of the sample space. For example, a smaller discovery rate would likely be indicative of lesser remaining unseen features. By observing the rate of feature discovery for each of our sample domains and vocabulary sizes, and analyzing how this rate varies with additional iterations, our objective is to gain insight into a fundamental component of our process. To evaluate the discovery rate of LTL statements, we develop 10 feature sets of 1000 LTL statements for each domain/vocabulary size configuration in our evaluation dataset, recording the quantity of new features discovered at each iteration of BayesLTL[39]. For each iteration, we average the quantity of new features discovered amongst the 10 sets and plot these values in scatter plots along with linear trend lines in Figure 4.5.

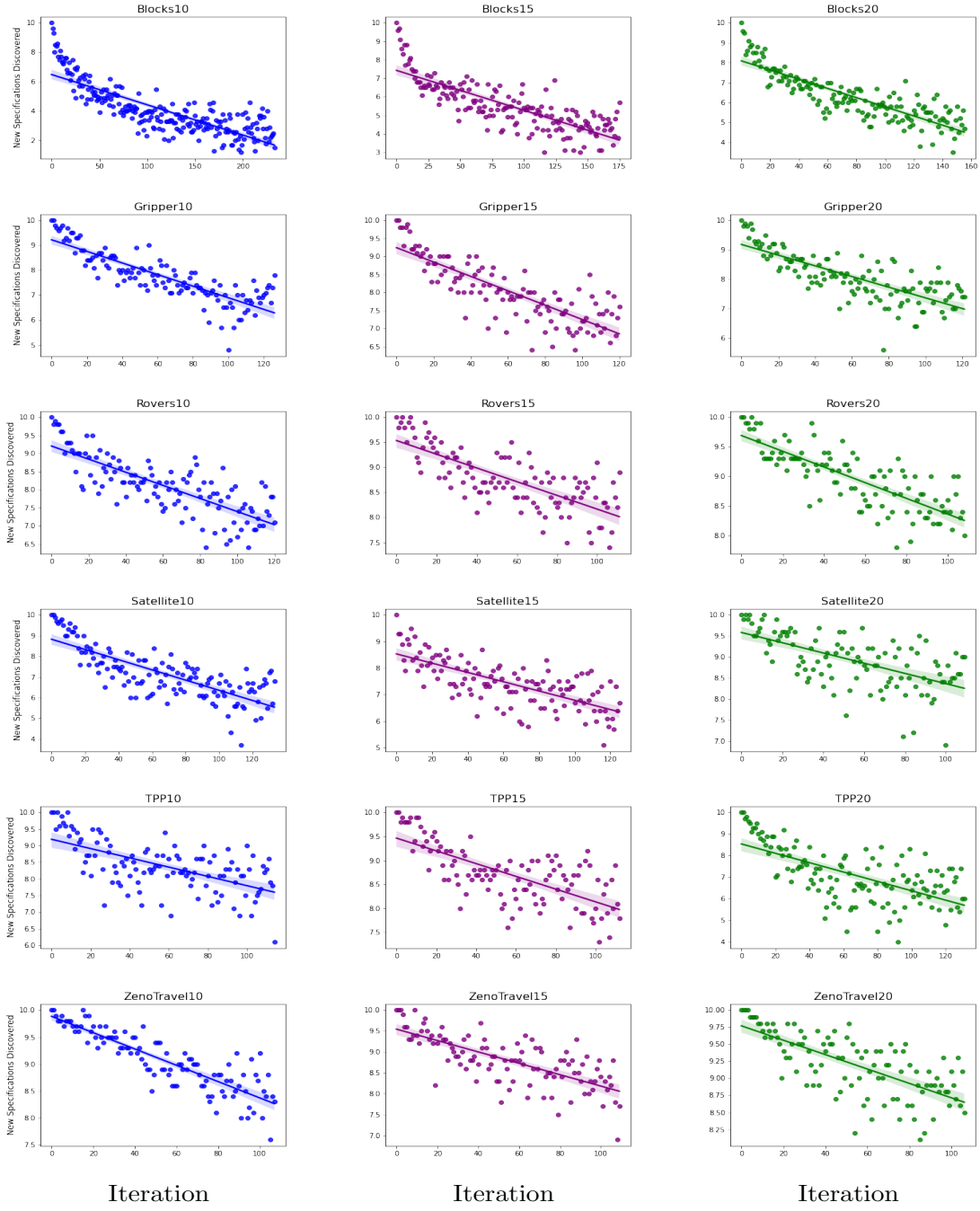

Figure 4.5: Evaluation Plots of LTL Rate of Discovery

Quantity of new specifications found per iteration of BayeLTL [39] search. Over the development of 10 unique feature sets of 1000 features, we count new LTL discovered per iteration and average this value across feature sets.

The rate of LTL discovery appears to decrease over increased iterations for all domain/vocabulary size configurations. This effect is especially exaggerated in configurations such as Block10, where a diminishing marginal effect is also clearly observed. For all configurations, an average of 9 or 10 features tend to be discovered within the first several iterations and this number consistently trends downward with additional iterations.

4.2.5 Similarity of Discovered Clusters Versus Quantity of Features

Since several instances of ground truth may exist within sets of traces, unique clusters may be discovered for each occasion of our clustering process. Variance in cluster discovery is not problematic, however the factors that influence this effect are of interest to better expose the mechanics of our procedure. While there are several factors that may influence variance of cluster discovery, we propose that the quantity of LTL statements used within the process likely has the strongest impact. Since feature sets of larger quantities are more likely to contain similar or identical features than sets of lesser quantities, resulting clusters should also be more similar. To test this hypothesis, we evaluated feature set sizes in $\{10n : n \in Z^+, i < 101\}$, establishing new features 10 times and clustering traces according to those features for each domain/vocabulary configuration within our evaluation dataset. Since we are most interested in the resulting groupings of our primary clustering step, for the purpose of this experiment, we evaluate clusters based on their definitions prior to our explanation step. For each trace, we then grouped the 10 discovered clusters for similarity analysis. We begin by evaluating similarity of two clusters of traces π_a and π_b according to $\frac{|\pi_a \cap \pi_b|}{|\pi_a \cup \pi_b|}$. Next, to measure similarity amongst multiple clusters, we

evaluate the similarity of all non-self mapping pairs of clusters and take the mean of these values. After measuring the similarity all 10 clusters for each trace, we then take the average of this similarity measurement for all traces. This similarity analysis is conducted for each experimental feature set size and plotted for visual understanding. Formally, the similarity of clusters can be represented as follows, where π_a and π_b represent clusters of traces, m represents any given set of clusters, and M represents the set of cluster sets corresponding with the clusters from each feature set that each trace belongs to.

- Similarity of two clusters:

$$\beta(\pi_a, \pi_b) = \frac{|\pi_a \cap \pi_b|}{|\pi_a \cup \pi_b|}$$

- Similarity of a set of clusters:

$$\sigma(m) = \frac{\sum_{i=1}^{|m|} \sum_{j=1}^{|m|-1} \beta(m_i, \{m : m_i \notin m\}_j)}{|m|(|m|-1)}$$

- Similarity of all clusters per trace:

$$\mu(M) = \frac{\sum_{i=1}^{|M|} \sigma(M_i)}{|M|}$$

Analyzing the similarity of our discovered clusters in Figure 4.6, it is evident that clustering similarity does in fact increase with the quantity of features used; this positive relationship between quantity of features and clustering similarity appears

consistent across all domains/vocabulary size configurations. As the quantity of features becomes very large however, this curve tends to flatten out, meaning similarity still increases with additional features, but the effect becomes less strong. These results are also indicative of consistency within the clustering step of our methodology, as long as enough features are used. As more features are captured, the knowledge provided to inform clustering becomes more similar, resulting in more uniform instances of identified clusters. The only exception to this discovered trend appears to be TPP10, where the similarity is seen to peak, tamper off slightly, then flatten out. While it is unclear why this peaking behaviour is observed, we still observe a reduction in variance, which is consistent with our previously proposed explanation of more uniform clustering instances, as a result of additional knowledge. Using these results to inform parameterization of feature quantity, lesser variation should be expected with larger values of the statement_limit parameter. Therefore, if more unique results are sought, perhaps in the event that multiple instances of clustering are conducted with the same traces, a lesser quantity of features should be used.

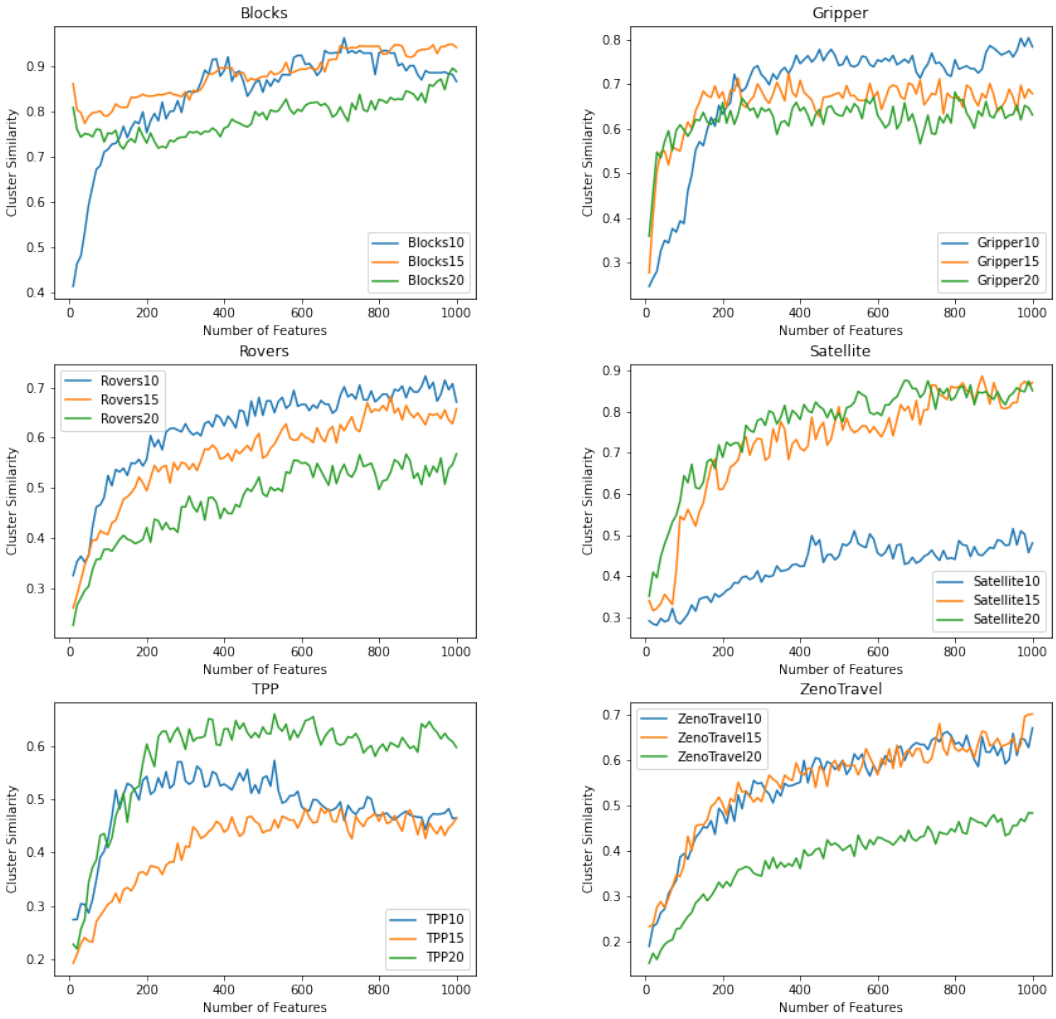

Figure 4.6: Cluster Similarity Versus Quantity of Features Plots

Evaluations of cluster similarly amongst 10 features sets with respect to the quantity of features used for each domain/vocabulary size configuration.

4.2.6 Conclusions

In this chapter, we introduced a novel trace clustering and delineation method that is demonstrated to be both effective and accurate. Given a set of traces, this methodology discovers a feature set of relevant LTL, clusters traces based on their entailment of features, and finally delineates traces based on a search procedure that identifies sets of features that all traces in a given cluster entail. Testing this approach on our evaluation dataset of 18 domain/vocabulary size configurations, we see that success rates approach 100% when sufficient feature quantities are used.

Two approximation methods were also introduced, which allow for more compact formulas to be discovered for cluster delineation. While approximation tends to comes at the cost of accuracy, our proposed delineation accuracy metric allows this tradeoff to be fully understood when approximation is used. Next, we investigated the effectiveness of off-the-shelf clustering evaluation metrics and determined that, due to the likely abundance of similarly correct cluster allocations, these metrics do not provide meaningful assessments.

Discovery rates of unseen temporal features were then evaluated within our feature discovery process, where we found that the quantity of features found per iteration tends to decrease near linearly. However, since multiple features are still discovered per iteration after several hundred iterations, this indicates that there are likely several additional formulas available to be found.

Finally, by clustering traces multiple times using varying sizes of feature sets, and measuring the similarity of generated clusters, we determine that as feature quantity increases, clusters become more similar at a marginally diminishing rate. Overall, through experimentation within our diverse evaluation domains, our novel

trace clustering and delineation method is proven to be robust, accurate, and effective.

Chapter 5

Tree Discovery using Temporal Logic

Our tree-based clustering and delineation approach embraces a binary tree data structure to represent groups of traces and their respective explanations. In this chapter we will begin by introducing the mechanics and procedures inside our methodology that enable tree discovery. Next, we will explore our proposed approach in greater depth within our evaluation and results by generating sample trees and measuring the results of performance-probing tests. Overall, through extensive analysis, we demonstrate tree generation as a robust and effective means to cluster and delineate traces.

5.1 Methodology

Our tree-based methodology begins by identifying and proposing a solution to a vital sub-problem, which is the identification of a procedure to accept a single set of traces and produce contrastive LTL to split the set into two similar sized subsets. To achieve this result, we propose a Monte Carlo approach in coordination with BayesLTL [39], which seeks to maximize set-size balance. Our proposed solution of this sub-problem then enables trees to be generated that automatically identify

clusters and their respective explanations. By initializing a root node to contain our problem's input traces and recursing this sub-algorithm until the desired quantity of clustered are found, this methodology is successful in discovering relevant clusters from discrete time-series data.

5.1.1 Overview

While BayesLTL [39] has proven effective in generating contrastive explanations between sets of traces, it is limited to exactly two sets of traces with known identity labels, referred to as positive and negative. If the quantity of sets is unknown and identity labels are nonexistent, the generation of constrastive explanations between sets of traces becomes much more challenging. In order to solve this problem, the goal of our approach is to leverage the strengths of BayesLTL [39] through the novel *creation of a contrastive explanation tree*. By initializing a tree with a root node containing the set of all input traces, a subproblem can be isolated as automatically identifying an LTL specification that optimizes a split of the traces within this node. If a process is discovered to effectively split an arbitrary single set of traces, this process can recursively be implemented to provide significant insights to explain the differences between all nodes. The following approach will introduce a Monte Carlo strategy to use BayesLTL [39] and effectively discover LTL specifications that split single sets of traces. Additionally, once the recursive splitting process begins, optimal stopping criteria must be determined to minimize complexity and overfitting; alternatively, the process would continue until only unique traces exist in the tree's terminal clusters. While default parameters have been defined to fit most use cases, those parameters have also been exposed to the user to be customized for unique

environments. An overview of this tree generation procedure can be seen in Figure 5.1, where tree growth can be understood as an iterative node-splitting process that continues until user-defined parameters are reached.

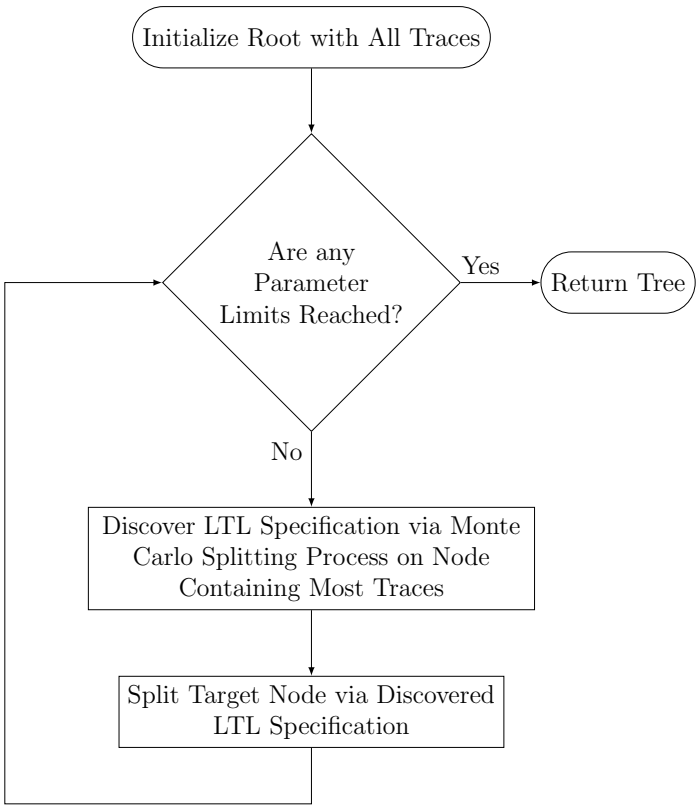

Figure 5.1: Flow Diagram of Tree Generation Process

Flow diagram of our tree generation process. Through targeting the node with the largest quantity of traces and splitting it via our Monte Carlo approach, our algorithm seeks balance to contrastively divide any given set of plan traces into two sets with an accompanying temporal logic specification. By recursing this procedure until predefined parameters are met, a tree of nodes representing clusters is defined and returned.

5.1.2 Tree and Node Structure

Our proposed delineation tree consists of nodes and edges that resemble a binary tree data structure. The primary attribute that defines a given node is a set of

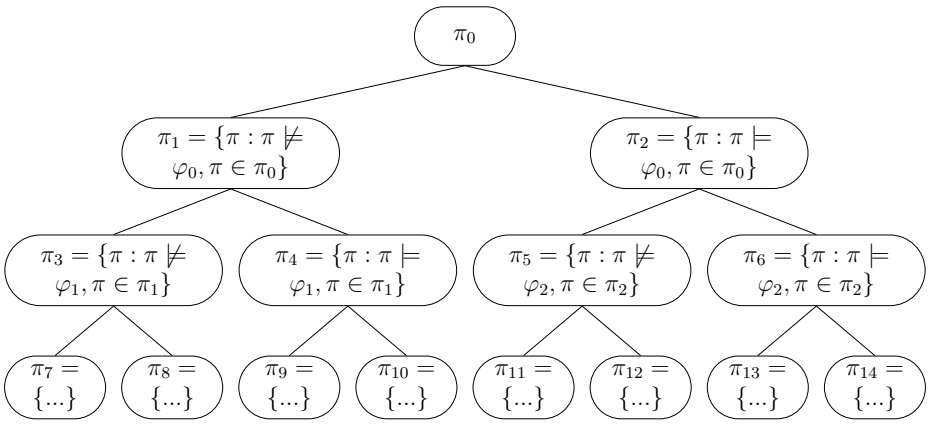

Figure 5.2: Structural Representation of Delineation Tree Format

A visual representation of the binary tree structure used within our delineation process. Each node π represents a collection of traces sorted by entailment for each formula φ down the tree.

internally stored traces. Non-terminal nodes also possess an LTL formula that is catered to their respective traces and designed for delineation. By using a given node's LTL specification, traces within that node can be constrastively evaluated based on entailment. This evaluation allows two new nodes to be created, whereby traces are allocated into two subsets based on whether the formula is satisfied by a given trace. These subsets of traces are initialized as new nodes, and this novel process is recursively implemented and repeated until a tree is fully created. Once the tree generation process is complete, the delineation of nodes can be analyzed as a collective or in subsets of any size. By evaluating the shortest path between any pair of nodes, the conjunction of LTL along the tree's branches allows nodes to be accurately differentiated and contrastively explained. See Figure 5.2 for a visual representation of this tree structure, where the root node is denoted as π_0. Expanding from π_0, the LTL specification φ_0 is discovered, which allows nodes π_1 and π_2 to be created. This node-splitting procedure continues to enable tree growth.

5.1.3 Node Splitting Criteria

As the primary engine of tree generation, our approach relies on a node splitting technique that is designed to efficiently discover contrastively differentiating LTL specifications. Representing the primary sub-problem of cluster identification and delineation, the challenge of this step is to automatically discover LTL specifications that maximize information gain. Formally, we define information gain of a specification φ, given node n, containing the set of traces π as:

$$1 - \frac{\Big|\big|\{\pi : \pi \models \varphi, \pi \in \pi_n\}\big| - \big|\{\pi : \pi \not\models \varphi, \pi \in \pi_n\}\big|\Big|}{|\pi_n|} \tag{5.1}$$

Specifications that demonstrate information gain of 1 are considered perfect, while specifications with information gain of 0 are ineffectual. By maximizing information gain, we maximize the balance of the resulting tree, leading to an optimally efficient and concise data structure.

To discover specifications that maximize information gain, our technique embraces the explanatory power of the BayesLTL framework [39]. Given the effectiveness of BayesLTL to identify LTL specifications differentiating two groups of traces, the tool can be used as an instrument to assist in the discovery of relevant splits. BayesLTL, however, requires positive and negative labels for input traces. Therefore, given a single set of unlabelled traces, the problem becomes identifying the optimal allocation of traces into positive and negative subsets (π_+ and π_-), such that an LTL specification can be discovered that maintains balance post-evaluation of entailment. Assuming a balance of $|\pi_{Positive}| = |\pi_{Negative}| = |\pi|/2$ for even-sized sets and $|\pi_{Positive}| = \lfloor|\pi|/2\rfloor$, $|\pi_{Negative}| = \lfloor|\pi|/2\rfloor + 1$ for odd-sized sets, the number of ways in which a cluster of

traces can be arranged into two balanced groups is represented by

$$\frac{1}{2}\begin{pmatrix} |\pi| + (|\pi| \equiv 2) \\ \frac{1}{2}(|\pi| + (|\pi| \equiv 2)) \end{pmatrix} \tag{5.2}$$

This defined search space very quickly leads to combinatorial explosion. To combat the complexity associated with this immense search space, we employ both sampling and Monte Carlo search.

We first use random sampling without replacement to mitigate the complexity of each search step. This means that instead of evaluating BayesLTL [39] on positive and negative sets of size $\frac{1}{2}|\pi_n|$, we reduce the contrastive evaluation to input sets of size $\frac{p}{2}|\pi_n|$, where $0 < p < 1$ represents the size of the sample proportion from the parent set. We demonstrate the representative abilities of varying sample proportions in Section 5.2.4, and allow this value to be adjusted as a parameter. By reducing search step complexity through our use of sampling, search capacity is enhanced.

We iterate search according to a Monte Carlo approach, which is procedurally shown in Algorithm 2, and represented as a flow chart in Figure 5.3.

Algorithm 2: LTL Specification Search

Input: Parent set of traces π_n

Parameters: Iteration limit max_iter, Information gain threshold τ, Sample proportion p

Output: LTL specification φ_{best}

$\varphi_{best} \leftarrow None$
for $i = 0...max\_iter$ **do**
  $\pi_{sample} \leftarrow$ random sample set of size $p|\pi_n|$ from π_n
  $\pi_+, \pi_- \leftarrow$ random balanced split of π_{sample}
  for values of φ resulting from BayesLTL(π_+, π_-) **do**
    if $InfoGain(\varphi, \pi_n) \geq \tau$ **then**
      return φ
    else if $!\varphi_{best}$ **or** $InfoGain(\varphi, \pi_n) > InfoGain(\varphi_{best}, \pi_n)$ **then**
      $\varphi_{best} \leftarrow \varphi$
    end if
  end for
end for
return φ_{best}

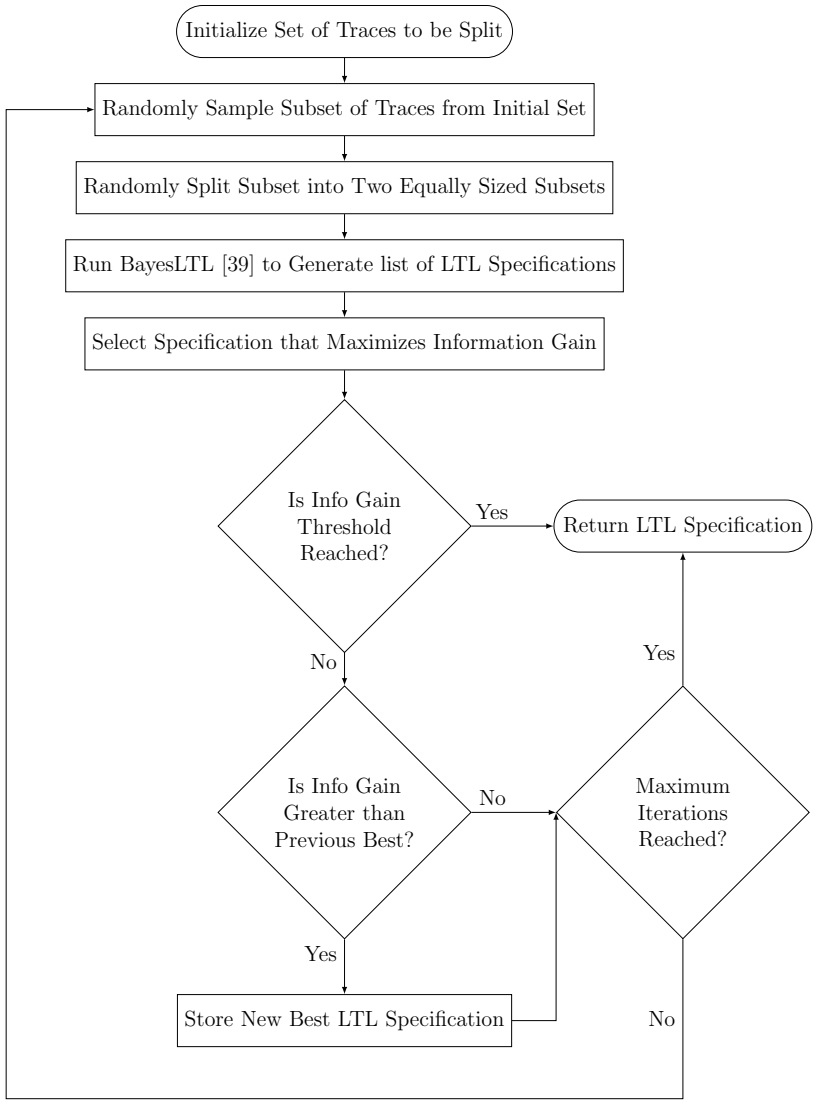

Figure 5.3: Flow Diagram of Monte Carlo Splitting Process

A flow diagram of our Monte Carlo splitting process, designed to split candidate nodes of traces into two new balanced nodes via the discovery of contrastive LTL. By conducting iterations of BayesLTL [39] until a minimal information gain threshold is reach or a maximum iterations parameter is reached, balance is sought amongst the two resulting subsets established by a given split.

Our proposed approach begins by sampling a subset of traces from the parent set, as previously described. We then randomly split this subset into two equally sized new subsets, labelling one positive and the other negative. Applying BayesLTL [39] to these positive and negatively labelled sets of traces, we arrive at a list of contrastive explanations, which each individually attempt to best describe variation between the two groups. Using this list of discovered LTL specifications, we score each formula on the information gain it offers to the parent set of traces. If a given specification is found to provide information gain that is either perfect or is beyond a parameterized threshold, the specification is accepted and the splitting process is ceased for that node. Alternatively, the specification offering the highest information gain is compared with the previous best, and the better specification is retained. This entire process is repeated until a parameter representing the maximum permitted quantity of iterations is reached, and the best found LTL formula is accepted. This information gain-maximizing formula is then stored in the parent node and used as a splitting mechanism to create child nodes.

5.1.4 Stopping Criteria

Without stopping criteria, the explanation tree would continuously grow until all possible sets of unique traces are individually isolated as terminal nodes. While this may be the desired outcome of some specialized problems, in most scenarios this would represent overfitting. Not only would the results become less useful, but the complexity of discovering so many explanations would also be quite significant. In order to maximize the derived value of the explanation tree, default stopping criteria has been defined as well as parameterized for user customization. The first

important stopping criterion to define is if all traces in a given node are identical. Continuing to branch from a node of entirely identical traces would lead to an infinite loop of unsuccessfully attempting to find an LTL specification that splits the group, then branching all traces in a random leftward or rightward direction. Alternative stopping criteria that can be defined by the user, including a maximum number of clusters, a maximum desired tree depth, a minimum quantity of elements and minimum proportion of elements that must remain in each node. Details regarding these criteria are discussed next.

5.1.5 Parameterization

The discovery of an explanation tree to contrast traces via LTL specifications is effective in several simulated environments, however, the way in which the tree is implemented has a strong influence on the power of its insights. For example, in environments with fewer traces, it may be of interest to understand the differences between each individual trace, but in environments with greater quantities of traces, a similar approach would clearly lead to overfitting. While "one-size-fit-all" parameters have been provided as default for baseline analysis, the ability to customize these parameters has also been provided to the user. Although these default parameters values are set to values that generally work well, to maximize the strength of a tree's insights in specialized domains, it is recommended that these parameters be carefully chosen and optimized to best fit the characteristics of a given problem setting.

k: Specified Number of Clusters

The k parameter allows the user to specify the quantity of terminal nodes for the generated tree to contain. This is an optional parameter; if k is not specified, the quantity of leaves will not be restricted.

max_depth: Maximum Depth of Tree

The max_depth parameter limits the depth in which the tree can grow. By default, max_depth is set to None, which allows the maximum depth of the tree to be unconstrained.

min_prop: Minimum Proportion in Bottom Nodes

The min_prop parameter establishes a minimum size, measured by the quantity of traces held within, that terminal nodes are permitted to be relative to the size of the tree's root node. The default value of min_prop is set at 10% to mitigate unnecessary computation associated with calculating LTL specifications down to the single trace level.

sample_prop: Sample Proportion to be Drawn from Parent Set

The *sample_prop* parameter determines the proportion of traces to be randomly sampled in each Monte Carlo iteration. For example, if a target node contains 500 traces and the *sample_prop* parameter is set to 20%, a random subset of 100 traces will be sampled at each iteration, resulting in candidate explanations being generated from two randomized sets of 50 traces. The default value of *sample_prop* is set at 10%, which is supported by our evaluation in Section 5.2.4, where we evaluate the

representative strength of various sample sizes. While larger sample sizes tended to provide greater representative ability, the magnitude of this effect was found to be very small-scale, so a small sample size of 10% represents an opportunity for runtime savings; when higher quality features are desired, runtime can be used as a tradeoff via higher values of *sample_prop*.

max_iter: Monte Carlo Maximum Iterations

The max_iter parameter controls the maximum quantity of iterations that will be undertaken when searching for LTL specifications that balance a given split. Greater values of max_iter will correspond with better balanced splits, however this will come at the cost of increased runtime. The default value of max_iter is 10, which allows balanced splits to be discovered for a wide range of problems.

verbose: Scope of Logging Output

This boolean parameter determines whether progress is logged to the user throughout the generation of the tree. If verbose is set to True, LTL specifications and information regarding discovered splits will be presented to the user live as they are discovered, however if verbose is set to False, no information will be displayed. The default value of the verbose parameter is set to True.

random_state: Random Number Generation

This parameter is used for the generation of random numbers within the cluster-splitting process, and allows tests to be reproducible. The default value of random_state is None, which randomizes the initialization of the random number generation.

5.1.6 Evaluation Metrics

To analyze the quality and characteristics of a generated tree, a variety of customized metrics have been considered. Since balanced trees are considered desirable, these metrics are largely designed to quantify balance. While these metrics can be used to evaluate independent trees by themselves, the metrics can also be used to contrast different trees when manually examining best fit.

Average Depth

The average depth metric is calculated as the sum of depth of all nodes in a tree divided by the total quantity of nodes. While average depth is a valuable measurement to understand the structure of a single tree, it is limited when it comes to comparing different trees to each other. Since trees can be initialized with root nodes containing varying quantities of traces, the number of splits to obtain similar clusters can also vary. However, when comparing trees with identical root nodes, the average depth metric can be quite insightful.

Worst Depth

The worst depth metric is evaluated by identifying the depth of the tree's deepest leaf node. The measurement of a tree's worst depth represents the maximum quantity of LTL specifications required to explain an identified cluster of traces. Similar to the average depth metric, the worst depth metric is less valuable for comparing different trees to each other, and offers greater value for understanding the structure of an individual tree.

Average Depth to $\log_2$n Ratio

To better compare different trees to each other, the average depth of a given tree can be represented in relation to its quantity of nodes (n). In order to do this, the average depth can be divided by $log_2 n$ to represent the balance of the tree. The denominator of $log_2 n$ is used because in a perfectly balanced binary search tree, most nodes are at the bottom of the tree, leading to an average depth of approximately $log_2 n$. This means that smaller values of the ratio are indicative of more balanced trees, and perfectly balanced trees should demonstrate ratios converging to 1.

Worst Depth to $\log_2$n Ratio

The calculation of worst depth with respect to $log_2 n$ is also an effective measurement to account for tree size and compare the structures of various trees. For similar reasons to evaluating $log_2 n$ as the denominator under average depth, calculating the ratio of worst depth to $log_2 n$ provides insight into the least balanced component of a given tree. Also similar to the average depth to $log_2 n$ ratio, smaller values of this ratio are indicative of more balanced trees.

Depth Variance

The depth variance metric is calculated by evaluating the variance of terminal node depth within a given tree. Depth variance attempts to quantify the dispersal of node-depths within a tree to provide insight into the tree's structure.

5.2 Evaluations and Results

Our evaluation begins by generating a sample set of trees, and measuring the samples according to relevant metrics, such as average node depth and aggregated information gain. We then analyze the deeper mechanics of this method by first testing the effect of node size on execution time. Next, the representative ability of LTL derived from sample sets to balance the larger set is assessed via estimates of probability distributions. The value of additional exploration iterations is then evaluated, and finally, an intelligent splitting approach is investigated to reduce iteration requirements.

5.2.1 General Effectiveness of Tree Generation

To evaluate the effectiveness and applicability of our delineation process, we analyze our approach's adaptability and scalability. To assess these attributes, we generated 20 trees for each domain/vocabulary size configuration in our evaluation dataset. An example tree is presented in Figure 5.4 for a better understanding of what these trees look like. Within Figure 5.4, however, we set the k parameter to 8 for compactness, whereas we allow k to represent the quantity of traces for the purpose of this experiment.

Post-generation, we then judged the strength of the discovered trees according to balance and information gain metrics. Balance, which represents the efficiency of our

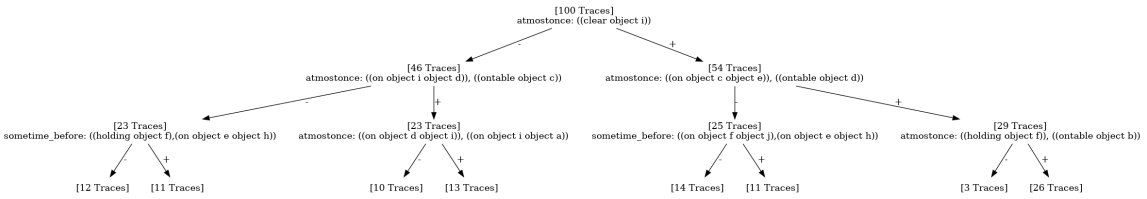

Figure 5.4: Example of a Generated Tree from Blocksworld Domain

Example tree generated from the blocksworld evaluation domain with a vocabulary size of 10. This tree was derived using a value of 8 for the k parameter. The to_png() function within our code's Tree class allows visuals like this to be automatically created for any discovered tree, providing a more intuitive understanding of trace allocation.

process to organize and differentiate traces, was calculated using average node depth. Since high quality specifications minimize the quantity of splits required to isolate traces, smaller values of average depth are desirable, as they are indicative of higher quality specifications. When analyzing average depth of trees, it is important to note they can both only be evaluated in contrast to trees with identical quantities of root traces, as is the case within our presented experiment. Alternatively, the balance can also be evaluated as a ratio of $\log_2(n)$, where n represents the total quantity of nodes in a tree, releasing the measurement from its context dependency, but the resulting value is less intuitive to comprehend. We calculated information gain according to Formula 5.1 for all nodes containing at least 10 traces in each tree, then averaged these values for each individual tree, presenting the median tree. The minimum node size of 10 traces was selected as a filter for our analysis of information gain as averages should not be overshadowed and overstated by including easier splits of less relevant nodes. By generating and evaluating several trees of diverse configurations, the resilience of our algorithm to identify high quality specifications is demonstrated.

Table 5.1 summarizes the characteristics of the resulting trees generated to evaluate the efficacy of our approach. For comparison when analyzing these values, an

optimal tree initialized with 100 traces would have an average depth of 5.759. From the recorded measurements, it is clear that our algorithm is capable of discovering and differentiating clusters existing within a variety of unique domains associated with diverse vocabulary sizes. In analyzing the median information gain of our evaluation trees, we observe little impediment associated with higher-complexity problems. This is also shown to translate successfully to median average depth, where measures appear similarly small across all domains and vocabulary sizes. These observations effectively demonstrate our algorithm's ability to cluster and delineate traces in an efficient manner.

Domain (100 Traces)	$\|V\|$	Average Depth (20 Trees)		Average Info Gain (20 Trees), ($\|\pi\| \geq 10$)	
		Med	Mean	Med	Mean
Blocks	10	6.299	6.308	0.505	0.501
	15	6.372	6.404	0.388	0.380
	20	6.219	6.202	0.128	0.135
Gripper	10	5.789	5.792	0.860	0.867
	15	5.995	6.009	0.729	0.726
	20	6.382	6.381	0.622	0.618
Rovers	10	5.836	5.821	0.788	0.783
	15	5.942	5.944	0.873	0.876
	20	5.783	5.776	0.879	0.880
Satellite	10	5.789	5.788	0.911	0.921
	15	5.810	5.811	0.844	0.847
	20	5.783	5.797	0.879	0.873
TPP	10	5.779	5.784	0.897	0.898
	15	5.789	5.787	0.891	0.893
	20	5.799	5.796	0.888	0.893
Zeno-Travel	10	5.779	5.781	0.898	0.899
	15	5.784	5.784	0.897	0.895
	20	5.779	5.779	0.899	0.895

Table 5.1: General Evaluation of Sample Trees

Performance results of 20 test cases for each of the listed domains and vocabulary sizes $|V|$. Trees were generated using a sample proportion of 40%, an information gain threshold of 80%, and a maximum of 10 iterations, isolating all unique traces as leaves. Each row reports the median and mean of average node depth and average information gain from the 20 samples of each respective domain/vocabulary configuration. For reference, a perfect tree initialized with 100 traces and possessing maximal information gain at every node, would have an average node depth of 5.759.

5.2.2 Analysis of Discovered Specifications

Our algorithm is capable of identifying high-information gain LTL specifications to establish minimal-depth trees; however a more extensive analysis of those specifications is necessary to rule out unintended behaviour. For instance, it would likely be problematic if only large specifications were identified because this would be indicative of overfitting, mitigating the usefulness of discovered formulae. To investigate the quality tendency of discovered specifications, we evaluate the size of the specifications within our sample trees; we measure size according to the number of unique subformulas within a given expression, as defined by Gaglione et al. and discussed in Section 2.2.

In Table 5.2 we present the resulting size measurements of specifications within our 20 evaluation trees for each domain/vocabulary configuration. These values demonstrate our algorithm's ability to identify low-complexity formulas corresponding with a robust fit for each configuration within our test data. Histograms of these distributions can also be viewed in Figure 5.5.

Domain (100 Traces)	$\|V\|$	Average LTL Size (20 Trees), ($\|\pi\| \geq 10$) Min	Med	Max	σ
Blocks	10	6.905	7.841	8.857	0.626
	15	4.381	5.390	6.947	0.676
	20	3.600	4.667	5.733	0.562
Gripper	10	5.357	6.942	8.357	0.765
	15	6.933	8.036	9.063	0.742
	20	7.125	8.092	10.222	0.744
Rovers	10	5.824	7.129	10.467	1.081
	15	7.071	8.171	10.467	1.083
	20	5.400	8.157	10.800	1.231
Satellite	10	6.266	7.633	10.231	1.104
	15	4.643	6.829	8.692	0.712
	20	5.214	6.379	8.067	1.009
TPP	10	6.423	8.100	10.307	1.104
	15	4.643	7.031	11.214	1.601
	20	5.214	6.893	9.286	0.935
Zeno-Travel	10	5.857	9.033	11.400	1.407
	15	6.308	8.136	10.357	1.180
	20	4.923	7.833	9.786	1.085

Table 5.2: Size Analysis of LTL Within Evaluation Trees

Analysis of LTL specification size $|\varphi|$ distribution within the test cases established in Table 5.1. LTL size is measured according to the quantity of unique subformulas, as defined by Gaglione et al.. Each row reports the minimum, median, maximum, and standard deviation of average $|\varphi|$ within the 20 sample trees of each respective domain/vocabulary configuration, where robust fit is apparent.

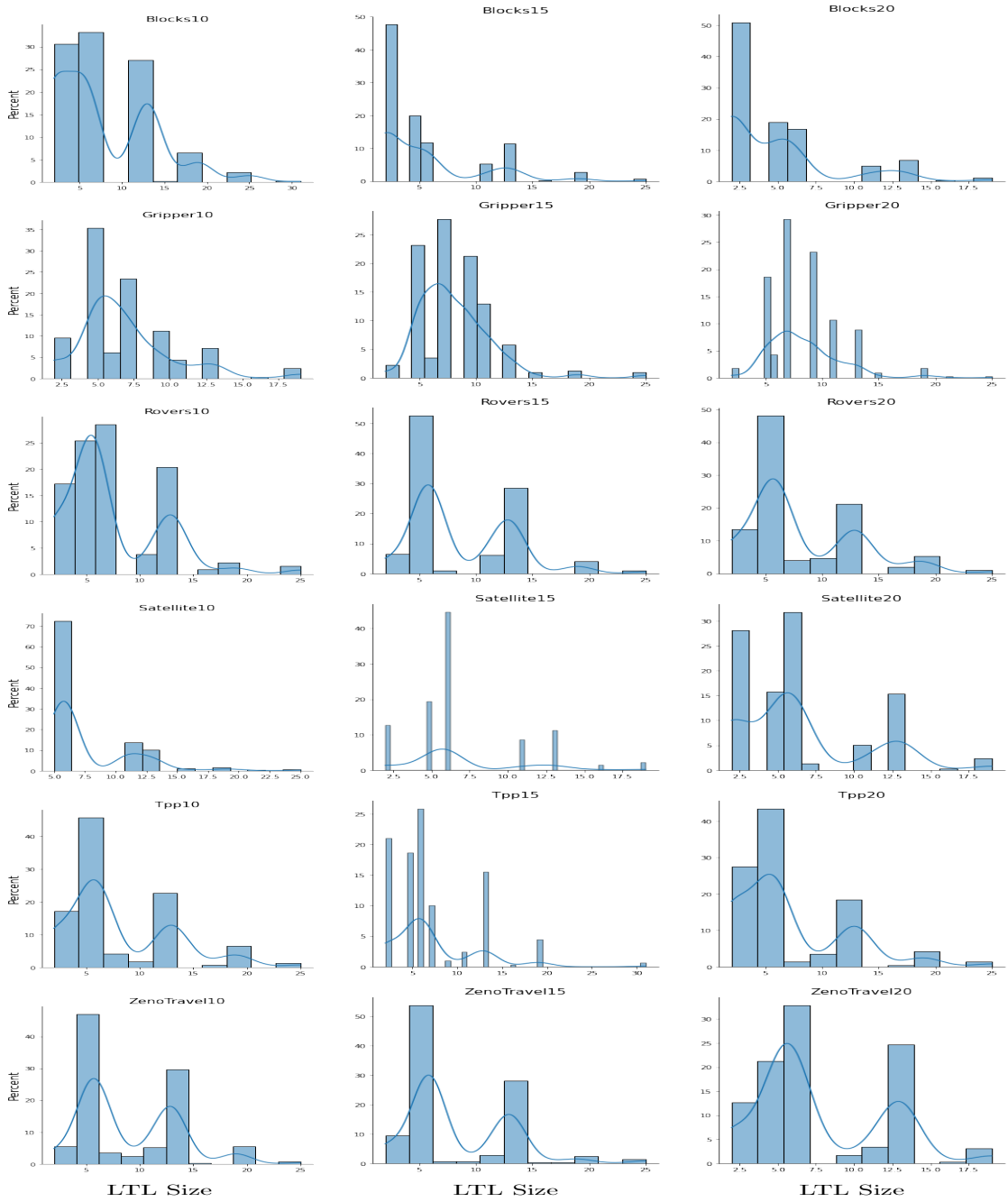

Figure 5.5: Histograms Representing Distributions of LTL Size in Evaluation Trees

Histograms of LTL size distributions amongst evaluation trees from all domain/vocabulary size configurations. Each histogram represents the distribution of LTL size of combined formula sets of 20 trees. Only nodes containing 10 or more traces are included within this analysis to ensure only relevant splits are evaluated.

An interesting observation within these histograms is a bi-modal tendency that is apparent within some domains. This is likely due to the lower bound of LTL size that is found whenever strong generalization must occur, in addition to a normal distribution of LTL size when non-generalized splits can be identified. To provide a clear understanding of the LTL specifications our process discovers, we present example formulas in Appendix B. Another insightful finding in these measurements is the tendency of LTL size to negatively correlate with vocabulary size in most domains. This trend is likely due to the idea that higher complexity domains possess greater quantities of natural ground truth to be found, which eases discovery.

5.2.3 Node Size Versus Splitting Time

Since our Monte Carlo node-splitting technique with randomized balanced subsets of traces represents the primary sub-process behind our delineation method, the execution time performance of this step, with respect to the quantity of analyzed traces, was of interest. We expected the execution time required to split a single node of traces to increase with the quantity of input traces; however, the rate of growth defining this relationship was unknown. To approximate this rate of growth, we conducted 10 splits for each size $|\pi|$ in $\{10n : n \in Z^+, n < 11\}$ of randomly sampled subsets of traces, measuring execution time of each split. By plotting these data points for each domain, the relationship between node size and splitting time can be visualized and understood.

In Figure 5.6, we observe a near linear relationship between node size and splitting time within all six of the explored domains. This relationship emphasizes the importance of our sampling step, since the time complexity of a single split appears

to grow continuously large.

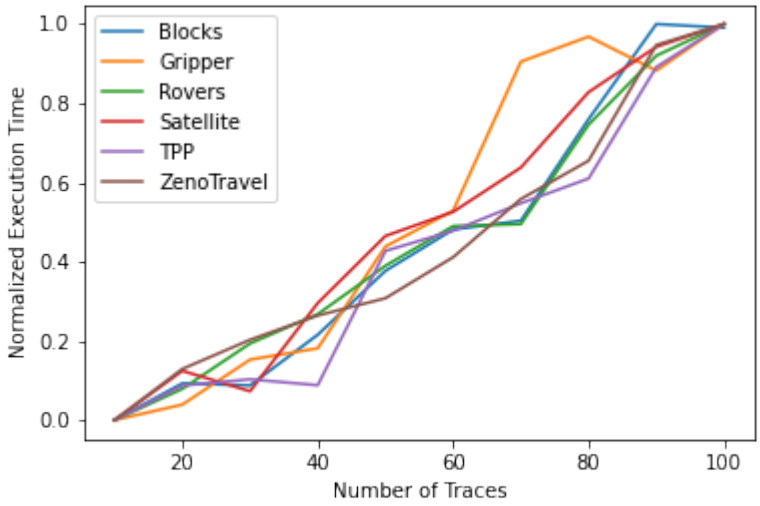

Figure 5.6: Evaluation Plots of Execution Time Versus Node Size

The average execution time of splitting a node π of randomly sampled traces with respect to the quantity of traces $|\pi|$ over 10 splits for each value of $|\pi|$ in $\{10n : n \in Z^+, n < 11\}$. Since scales of execution time differ relative to each domain, normalization is conducted via $\frac{x_i - min(x)}{max(x) - min(x)}$ for plotting purposes.

5.2.4 Probability Distribution of Information Gain with Respect to Sample Size

Given a randomly sampled subset of traces of a larger set, it is of interest to derive the probability distribution of information gain as a function of sample size. Intuitively, larger sample sizes should have stronger representative abilities; however since runtime is dependent upon the size of the input set, accepting fewer sampled traces may be of better utility. We estimated this distribution by running 100 splits of sample proportions in $\{0.02, 0.05, 0.1, 0.4, 0.7, 1\}$ for each domain and analyzing the resulting information gain.

As seen in Figure 5.7, sampling was shown to be highly effective and proved

capable of representing the population distributions.

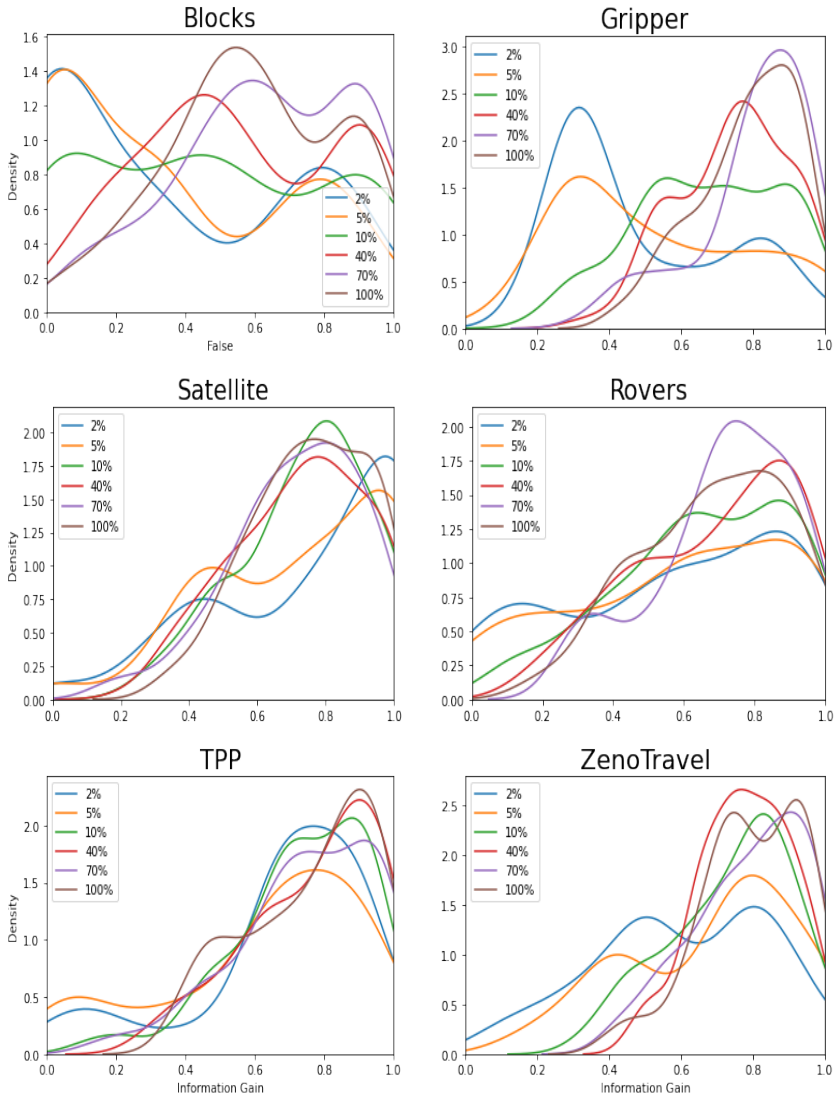

Figure 5.7: Kernel Density Estimations of Information Gain Versus Sample Proportion

Kernel density estimation (KDE) plots of information gain with respect to sample proportion used when approximating split. Each domain subplot shows the KDE's of 100 single-split iterations for each sample proportion in {0.02, 0.05, 0.1, 0.4, 0.7, 1}.

Within all six of the evaluated domains and across each of the tested sample proportions, significant density was observed in the upper range of the information gain spectrum. As would be expected, higher sample proportions tended to provide greater representative ability; however, the size of this effect was interestingly small-scale. Within some domains, such as Satellite and TPP, we observe sampling effectiveness with sampling sizes as small as 2%; within all domains, however, we observe effectiveness with samples as small as 10%. This tiny performance cost of using small sample sizes represents a strong opportunity for runtime savings within complex domains.

5.2.5 Specification Exploration Time Utility

When searching for an LTL specification to split a given set of traces, the quantity of exploration iterations permitted will impact the information gain of the resulting split. Since iterations represent opportunities for better specifications to be found, the quantity of iterations should positively correlate with the resulting information gain. The rate at which information gain is improved per iteration is of interest because iterations come at a cost of execution time. To investigate this tradeoff, we conducted 20 splits for each iteration limit value in $\{5i : i \in Z^+, i < 11\}$ for each evaluation domain, recording execution times and discovered LTL. We also used a sample size of 100% to consistently ensure maximal information was provided to the algorithm. By analyzing the average information gain and average execution time of these splits with respect to the iteration limit used for each domain, the curve of this relationship can be approximated.

When increasing specification search iterations, we observe performance improvements through measurements of information gain across all six evaluation domains. However, with marginal increases, the size of observed information gain improvement tends to decrease. This trend of diminishing marginal benefit of search iterations persists across all experimented values of permitted iterations, while search time appears to increase near-linearly. This means that although the cost of search time increases at a constant rate, the marginal value received by incurring this cost decreases with higher values of the parameter. This curve of diminishing marginal value can be seen in Figure 5.8. This trend can likely be attributed to the idea that specifications satisfying the information gain threshold do not always exist, or, for a variety of reasons, may be less conducive to being found. Since our algorithm's precondition for early stopping is achieving a minimum score of information gain, if this milestone is impossible to reach, all iterations will still be conducted, even if a globally optimal specification has already been found.

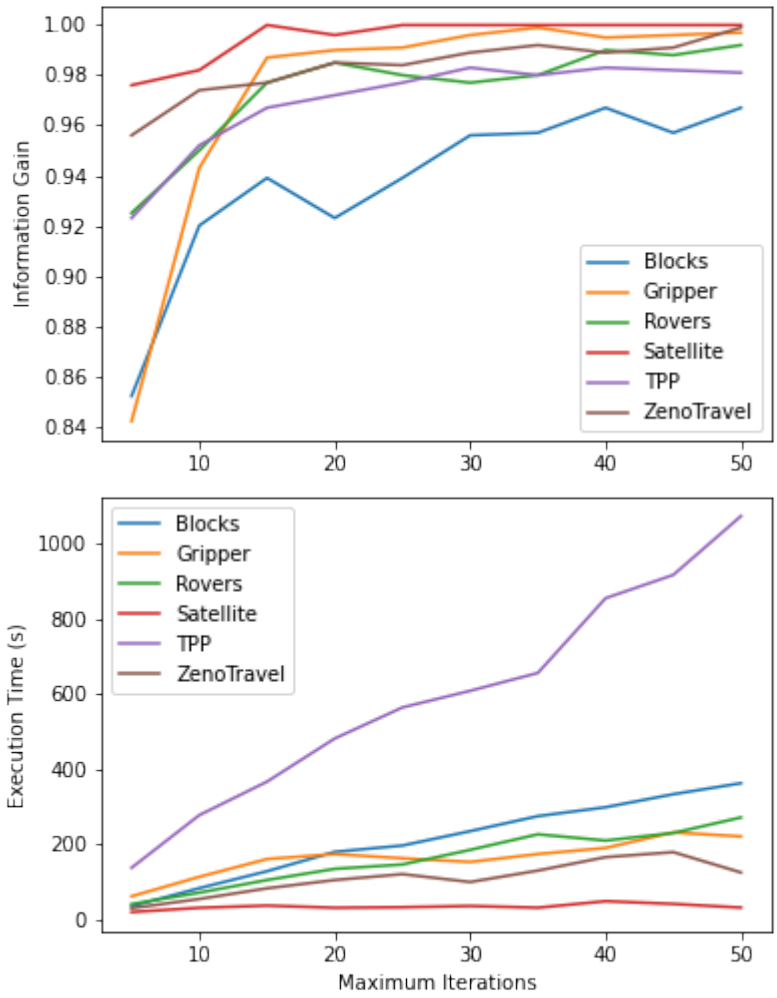

Figure 5.8: Information Gain and Execution Time Versus Maximum Iterations Plots

The average information gain and average execution time with respect to permitted maximum quantity of iterations over 20 sample splits for each test value and evaluation domain. Splits were conducted using a sample proportion of 10% with an information gain threshold of 100% to ensure maximal performance potential.

5.2.6 Intelligent Splitting Process

With complexity mitigation in mind, we investigated a process whereby splitting criteria could be intelligently identified via the analysis of previous splitting attempts. In this revised procedure, when a discovered specification fails to meet the information

gain threshold, we do not resample π_+ and π_- from π_{parent}. Instead, we reestablish π_+ and π_- as the subsets of traces that satisfy and dissatisfy φ, respectfully. We then rebalance π_+ and π_- by randomly drawing traces from the majority set and inserting them into the minority set. Our hope was that the sizes of these subsets would converge over multiple iterations, leading to the desired information gain specified by the threshold. An overview of this revised method can be viewed in Figure 5.9 and procedurally in Algorithm 3.

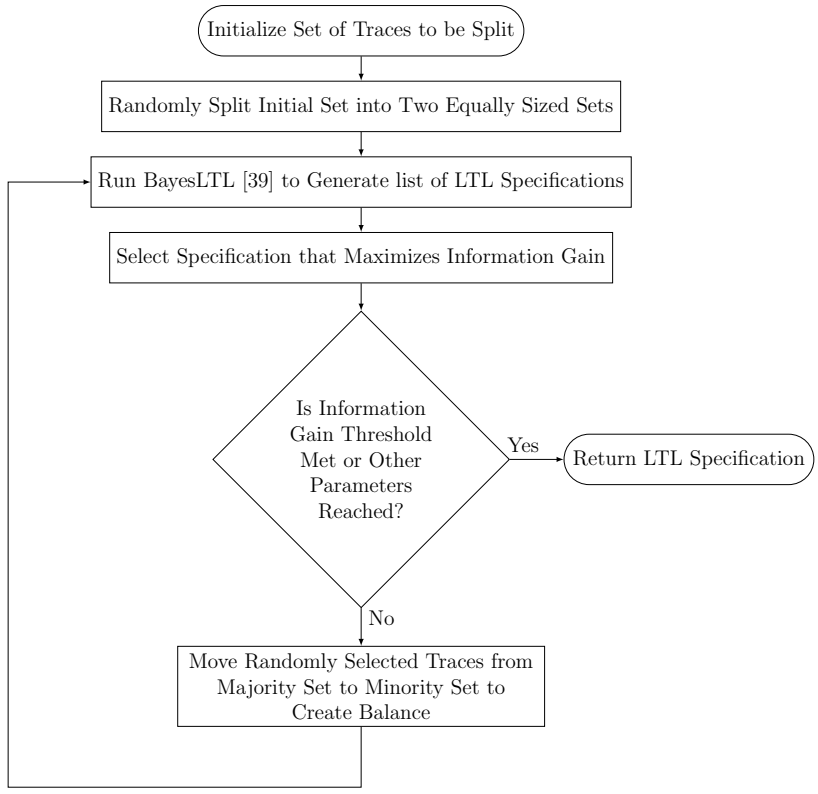

Figure 5.9: Flow Diagram of Experimental Intelligent Splitting Process

Process flow diagram of our experimented intelligent splitting procedure. By substituting our Monte Carlo approach for a subset re-balancing step, the objective of this revised procedure is to mitigate complexity through a reduction in the quantity BayesLTL [39] iterations.

When testing this revised process to split nodes of traces, we observed identical

Algorithm 3: Experimental Intelligent Splitting Algorithm

Input: Parent set of traces π_n
Parameters: Iteration limit max_iter, Information gain threshold τ, Sample proportion p
Output: LTL specification φ_{best}

$\varphi_{best} \leftarrow None$
$\pi_{sample} \leftarrow$ random sample set of size $p|\pi_n|$ from π_n
$\pi_+, \pi_- \leftarrow$ random balanced split of π_{sample}
for $(i = 0...max\_iter)$ **do**
  $\varphi \leftarrow$ LTL that maximizes $InfoGain(\varphi, \pi_n)$ from BayesLTL(π_+, π_-)
  if $InfoGain_{\varphi,\pi_n} \geq \tau$ **then**
    return φ
  else if $!\varphi_{best}$ **or** $InfoGain(\varphi, \pi_n) > InfoGain(\varphi_{best}, \pi_n)$ **then**
    $\varphi_{best} \leftarrow \varphi$
  end if
  while $(||\pi_+| - |\pi_-|| > 1)$ **do**
    if $|\pi_+| > |\pi_-|$ **then**
      Randomly select trace from π_+ and append to π_-
    else
      Randomly select trace from π_- and append to π_+
    end if
  end while
end for
return φ_{best}

formulas being discovered repeatedly until the maximum iterations were reached. We observed this same pattern across all six of our evaluation domains, indicating a clear lack of success. With no variety in the formulas being discovered, our revised algorithm failed to learn, meaning the resulting information gain was equal to that of the first identified formula. While this observed failure to learn rejects the proposed method as a viable alternative, it is possible that the approach could be adapted and improved to make learning feasible. This potential avenue for future work will be discussed in greater depth within Section 7.3.

5.3 Conclusions

In this methodology, we proposed a new approach to discrete time-series clustering and delineation, which embraces a tree structure to organize and differentiate traces. By introducing a Monte Carlo approach in coordination with BayesLTL [39], we proposed a strategy to split any set of traces into two similarly sized subsets via contrastive LTL. Recursing this procedure, our methodology allows relevant LTL to be discovered in the form of tree generation, where groups of traces are differentiated via the conjunction of LTL along the tree's branches.

Exploring the effectiveness of this approach within sample trees, we discover a strong tendency towards balance, as measured by average depth, in addition to average information gain within the tree's splits. We also measure the size of resulting LTL to assess underfitting/overfitting, and conclude that robust fit is common within all of our experiment domains.

Next, we discover a relationship between node size and splitting time, which provides strong rationale for our sampling step. Analyzing this sampling step further, we estimate probability distributions of information gain with respect to sample sizes, and discover sampling effectiveness with samples as small as 2% in some evaluation domains, and 10% in all domains.

We then explore the value of marginal iterations when searching for LTL to split a given set of traces, and discover and diminishing marginal relationship. Next, we investigate an intelligent splitting approach designed to facilitate learning from prior splitting attempts. Testing this revised approach, we establish key takeaways to incorporate into future research of intelligent splitting processes.

Finally, we contrast the effectiveness of our two unique methodologies and discuss

the strengths and weaknesses of both approaches. Overall, we conclude that our tree-based methodology is both effective and applicable in the context of discrete time-series clustering and delineation.

Chapter 6

Related Work

To provide a better understanding of the contributions we have made, we will discuss this thesis in the context of related work. We will begin by discussing time series predictability, followed by plan explanations, and LTL mining. Next, we discuss works from the areas of contrastive explanations, and LTL inference via decision tree learning. Finally, policy summarization, and trace clustering in business process mining will be explored. By discussing the similarities and differences between this thesis and related works, our contributions to these unique areas will be better understood.

6.1 Time Series Data Predictability

A component of time series analysis that has garnered significant research interest is the area of forecasting [22]. This is largely due to the business application of decision making under uncertainty. While the motivation of this paper is not necessarily rooted in forecasting, it is however focused on the identification of temporal features within time-series data, which forecasting also attempts to capture from its prediction-oriented perspective. One of the first statistical models to approach this problem was introduced by Box et al. in 1970 through an method called autoregressive

integrated moving average (ARIMA). By examining past observations and evaluating lagged moving averages, ARIMA allows for future values to be predicted based on the assumption that temporal structures will repeat themselves. While ARIMA is effective in modelling trends within time-series data, it is very limited when it comes to identifying seasonality. In order to better account for seasonality, an extension of ARIMA called Seasonal ARIMA (SARIMA) became common practice [6]. By using seasonal differencing, SARIMA transforms data into a stationary format to be analyzed. In addition to ARIMA and SARIMA, there are a variety of other widely-accepted classical forecasting methods available such as vector autoregressive [34] and exponential smoothing methods [20]. More recently, however, forecasting research has shifted towards the application and adaptation of modern AI tools such as neural networks for the time-series prediction problem. For instance, the AR-Net [64] model by Triebe et al., uses a feed-forward neural network approach to model autoregressive process dynamics. Other machine-learning approaches embrace recurrent neural network designs to take advantage of their strong ability to model sequential data [5, 59]. While the research objectives of this thesis and the area of forecasting are different in nature, there are overlapping elements of interest, such as the identifying features, rules, and system characteristics that are important to predictability. LTL specifications learned from discrete time-series data offers potential to the area of forecasting by discovering patterns that may repeat themselves in future scenarios.

6.2 Plan Explanations

As an alternative to traditional planning research which seeks to investigate the process of identifying optimal plans from problems, the area of plan explanation seeks to

identify and describe characteristics of problems from plans. Through the analysis of observed behaviour, the goal of plan explanation is to use abductive-reasoning to infer the rationale behind observed actions to better understand why certain events occur within plan traces. To make sense of observed actions, plan explanation research focuses on automatically learning temporal properties that allow system behaviour to be modelled, understood, and predicted.

The majority of prior works in the area of plan explanations are focused on describing a single set of plan traces. The analysis of one set of plan traces is similar to this thesis in that it allows system behaviour to be better understood, however is limited in scope because multiple sets of traces are not accounted for, as is the case in this thesis. An area where plan explanations have been particularly relevant is the setting of human-computer interaction [39]. This is because when plans are generated by computers, it is of significant benefit if they can be understood by humans in order to encourage adoption and enable participation by humans. Seegebarth et al. investigated this in 2012 with the intention of facilitating human-understanding of the rationale behind plans via automatically generated explanations [56]. To do this, they propose a formal approach where plan information is represented as first-order logic formulae and explanations are represented as proofs within the resulting system. Using a prototype interface, the real-time effectiveness of this system is demonstrated. Similarly, in 2017, Magnaguagno et al. investigated plan explanations by developing a cloud-based planning tool to assist new users in understanding and visualizing the plan generation process when writing code [45]. By integrating code editing and state-space visualization, Magnaguagno et al.'s tool called, "WEB PLANNER," emphasised relationship visualizations between the domain, problem, and resulting plan.

In coordination with the user, the tool generates two visualizations. The first visualization focuses on the explored state-space through a heuristic visual where cartesian and radial tree depictions are used; the second visualization employs a "Dovetail Metaphor" [46], which allows the user to view changing predicates throughout plan execution. While both of these prior works successfully demonstrate their effectiveness in describing the behaviour of plans through user-friendly interfaces, they lack the provision of rationale regarding plan selection in the first place, and they require expert interpretation for their value to fully be recognized. Our approach, in contrast, is designed to provide insightful rationale regarding relationships between all plans and can be interpreted fully with the basic understanding of LTL.

Alternative works have focused on explicable and predictable planning, where generated plans are designed to be understood by humans naturally with less dependence on additional tools. By operating on the idea that humans understand plans by associating tasks with actions, Zhang et al. developed a learned model for the labeling scheme of humans [68]. This model is then used by agents to synthesize or choose plans that are computationally explicable and predictable for humans to interact with. Similarly, in 2021, Seimetz et al. empirically study a method, where users are asked to annotate examples as good or bad, and LTL formulas are learning to extract plan preferences. Mixed-initiative planning is another method that has been developed to improve human understanding of planning. In mixed-initiative planning, plans are automatically revised based on user input. In 2018, Borgo et al. introduced an approach that is implemented in the XAI-PLAN framework, where humans are given the ability to suggest alternatives to plan actions and observe the resulting outcomes [11]. By comparing the results of different actions, humans are

provided stronger intuitions regarding plan rationale.

Plan explanations are also relevant in the setting of goal recognition. By observing an agent's behaviour, the objective of goal recognition is to infer the agent's plans and goals. Expanding upon goal recognition, probabilistic plan recognition seeks to infer a probability distribution over a set of possible goals. Ramírez and Geffner explored the topic of probabilistic plan recognition in 2010 using off-the-shelf classical planners [53]. Ramírez and Geffner proposed an approach where this problem can be efficiently addressed by calculating the cost of achieving a goal when complying to the observations versus not complying. Sohrabi et al. also investigated this problem in 2016 and discovered an extended approach to account for missing and noisy observations [60]. In settings where observations are unreliable, goal recognition becomes a much more difficult problem, since particular observed actions of an agent can be misleading in relation to the agent's goal. To account for this, Sohrabi et al. define two additional objectives in their approach to supplement the original plan costs, then linearly optimizes all of the objectives. Using this approach, the researchers successfully demonstrate the method's increased effectiveness of plan generation in most domains.

While each of the previously discussed works contribute significant value to the domain of plan explanations, they all focus on establishing explanations based on the perspective of a single model. Chakraborti et al. describes this as a "soliloquy" approach that is "wholly inadequate in most realistic scenarios" [19]. Since the domain and task models of humans tend to differ significantly from that of AI systems, Chakraborti et al. propose that plan explanations should instead be focused on discovering the differences between these models, and they introduce a method where

this can be approached as a model reconciliation problem [19]. By minimizing changes made to the human's model and specifying explanations in the form of model updates, a model reconciliation approach can be effective in improving the completion of an incomplete model. Our approach embraces a similar ideology, where we evaluate discrete time-series data from the fundamental perspective that unique systems tend to exist within subsets of a larger system. Chakraborti et al. propose that explanations should be studied in light of differing models, and we seek to identify temporal characteristics defining differing models within discrete time-series data.

6.3 Mining Linear Temporal Logic Specifications

Due to its modalities referring to time, LTL is capable of expressively and effectively describing temporal patterns within time-series data. These patterns represent considerable value because they can often provide insight into hidden dynamics within systems that cannot be represented by traditional metrics such as plan costs. While the value of describing time-series data with linear temporal logic is clear, the optimal process in which specifications are identified is less defined. In 2017, Kasenberg and Scheutz investigated the discovery of LTL specifications for agents planning in Markov Decision Processes. By approaching this as a multiobjective optimization problem, Kasenberg and Scheutz developed a process for mining globally persistent specifications using state-based and action-based objective functions, and a notion of "violation cost" [35]. Using these components, evolution-based algorithms can then be employed to optimize LTL formulas represented as parse trees. Lemieux et al. also explored the inference of LTL specifications in 2015 when they introduced a tool called Texada, designed to extract specifications of arbitrary length and complexity

[44]. Given a user-defined LTL property type template as well as a log of traces, Texada evaluates and outputs a set of LTL formulae representing all valid instances of the property type valid on all of the traces. A template based approach was also employed by Shah et al. to infer task specifications using a probabilistic model based on three prior distributions [58]. Shah et al.'s method introduces a domain-independent likelihood function which only depends on the number of conjunctive clauses in a candidate formula.

While each of these approaches are uniquely effective, they all focus only on the identification of specifications that are entailed by all of the plan traces. Similarly, the approaches outlined in this thesis are designed to identify LTL specifications that entail plan traces, however an important differentiating aspect is that they focus on specifications entailed by some, but not all sets of traces.

The task of mining accurate LTL specifications, however, only represents part of the challenge because the interestingness of discovered LTL is equally important. To evaluate interestingness, several unique metrics exist, however the most common are called support and confidence, which were both introduced by Agrawal and Srikant [7]. While support measures the frequency of co-occurence of items appearing in a dataset, confidence measures amount of times if-then statements are found to be true. Cecconi et al. presents an excellent overview of many of these metrics in their 2021 paper, developed to facilitate the adaptation of these metrics to declarative process mining [18].

Since we embrace the BayesLTL framework [39] as a subprocess of both of our methodologies, we also indirectly use the same method to identify interestingness.

BayesLTL selects suitable formulae and assigns interestingness scores based on simplicity of LTL, designer template/proposition preferences, and syntactic similarity to incumbent samples. We also use generic input preferences within our approach to accept a wide variety of LTL templates and propositions.

6.4 Contrastive Explanations

The research area of constrastive explanations concerns the delineation of traces using relative specifications, and its objectives are most similar to those within this thesis. Previous research on deriving contrastive explanations however, has been limited in scope and tends to focus on contrasting only two sets of traces; the methodologies within this thesis allow for any quantity of sets to be contrasted, and these sets are automatically identified. A novel approach within this domain was introduced by Neider and Gavran in 2018, and proposed a SAT-based method to produce LTL formulas that delineate two sets of traces [49]. By reducing the problem to a set of satisfiability problems in propositional boolean logic, Neider and Gavran embrace the power of optimized SAT solvers for this task. Another relevant approach that adopts SAT-based methods was introduced in 2019 by Camacho and McIlraith [16]. By using SAT solving to search through a state-space of labelled skeleton formulae, Camacho and McIlraith's approach constructs an alternating automaton that can be used to identify accurate LTL specifications [16]. While both of these discussed approaches [49, 16] demonstrate success in identifying valuable contrastive specifications for problems with perfect traces, since they are designed to output only one minimal-length LTL specification, they leave potential for failure when sets contain imperfect traces. For example, in a situation where no single specification exists to

delineate all traces from both sets, these approaches would be unsuccessful. To address this problem and account for noise such as sensor issues and unintended human behaviour in realistic environments, Kim et al. introduced a probablistic approach in 2019 called BayesLTL [39]. Discussed more thoroughly in Section 2.3, BayesLTL adopts a Bayesian inference approach to derive contrastive explanations between two sets of input traces. Our approach embraces this Bayesian inference strategy, using BayesLTL as a subprocess. However, instead of limiting the assertion of contrast to two sets of plan traces, we evaluate contrast amongst k-sets. Our approach also clusters input traces to establish suitable contrastive sets automatically, as opposed to prior works, which require contrastive sets to be predefined by the user.

6.5 Temporal Logic Inference via Decision Tree Learning

Leveraging decision tree learning algorithms to infer temporal logic formulas is an area that has also been previously explored [10, 14, 28]. As the most relevant approach in this space, Gaglione et al. use decision tree learning in their Algorithm 2 to discover contrastive LTL specifications between two sets of plan traces. Similarly, our tree-based approach adopts a decision tree learning method for contrastive explanations; however, our tree's structure is also designed to hold sets of traces in nodes, rather than only representing formulas. Our use of decision tree learning is focused on the task of cluster discovery, conducted simultaneously with formula construction.

6.6 Policy Summarization

The topic of policy summarization focuses on globally characterizing the actions of an agent in order to describe its policy to a user. A common method in which this

has been done is through the selection of a subset of state-action pairs, where the challenge is determining the set of pairs that leads to the most important insights. Amir and Amir explored policy summarization in 2018 through the lens of identifying state-action pairs when they developed an algorithm called HIGHLIGHTS designed to describe agent behaviour in different situations [9]. By using agent simulations, HIGHLIGHTS can effectively extract interesting trajectories and assist humans in understanding the capabilities of an agent. State-action pair identification was also studied by Lage et al. in 2019, when they introduced an imitation-learning based approach to extract and reconstruct an agent's policy [42]. Additionally Lage et al. discovered that in order to produce high-quality reconstructions, it is important for the model used during summarization to be the same as the one used during reconstruction. Our approach to discrete time-series clustering within this thesis is also designed to analyze system behaviour according to variations defined by unique situations. However, instead of summarizing this behavior in the form of policy, the objective of our model is to identify these unique situations from raw trace data and provide differentiating criteria.

Another tool that can be used for the task of policy summarization is the notion of abstraction. In abstraction-based approaches, the goal is to present the agent's policy at a high-level, such that all details that are not directly important are left out. Topin and Veloso investigated an abstraction-based approach in 2019 in the area of explainable deep learning by introducing Abstracted Policy Graphs [63]. By explaining the decisions of an agent in the context of expected future transitions, Topin and Veloso's Abstracted Policy Graphs provide significant insight into agent policy using only a learned value function as well as observed transitions. Another

novel approach focused on abstraction was formalized in 2020 by Sreedharan et al.. Through the derivation of analytically computed landmarks, Sreedharan et al. use the ordering of these landmarks to summarize policies for Stochastic Shortest Path Problems [62]. These temporal abstractions are then demonstrated by Sreedharan et al. to provide high-level overviews of agent policy.

Although similar to abstraction-based methods, other works approach policy explanation by establishing mappings to interpretable terms. Hayes and Shah explored this strategy in 2017 with the goal of synthesizing the control logic of robots for humans to understand [32]. By using a composition of functions, Hayes and Shah's approach first identifies a question being asked and maps it to a template, then the template is resolved to a relevant set of states, summarized concisely according to relevant attributes, and finally mapped to natural language for the user. Another insightful contribution in the realm of mapping to interpretable terms was developed by Koul et al. in 2018 when they explored policy summarization for recurrent neural networks (RNNs) [40]. Koul et al.'s novel technique called Quantized Bottleneck Insertion is designed to learn finite policy representations of the continuous-valued memory vectors and observation features within RNNs. Using this approach, Koul et al. effectively demonstrate improvements in model interpretability via learned policy representations. Similarly, our approach to discrete time-series clustering emphasizes interpretablity by using policy-like rules to differentiate vectors, however, we focus on identifying multiple sets of ruled based on recorded behaviour, rather than the analysis of a single model.

6.7 Trace Clustering in Business Process Mining

Prior to the age of computers, workflow models were used by organizations to visually explain workflow processes. This traditional approach to workflow management involved manual generalizations of process models representing idealistic views of operational systems and attempted to continuously improve them over time. A focus on the reengineering of processes was sparked in 1993 by Hammer and Champy through their book titled "Reengineering the Corporation: A manifesto for Business Revolution" [31], which emphasized the notion of radically changing processes by manually analyzing time-series data to identify operational inefficiencies. The integration of technology with process management, called process mining, however, represents the most significant advancement in the field. According to Aalst, a pioneer in the field, "process mining bridges the gap between traditional model-based process analysis and data-centric analysis techniques such as machine learning and data mining" [2].

Process mining has since demonstrated significant success in many domains using time-series data extracted from real events to map the actions of agents to business processes [2]. By using process mining techniques, the potential is created within organizations to improve operational efficiencies by identifying redundancies and seeking routes of less resistance. A primary challenge within the field, however, is the lack of structure characteristic of flexible real-world environments. Since business processes in flexible environments tend to lack perfect structure, the analysis of behaviour within these environments often leads to similarly unstructured results. Examples of common problems associated with process mining in flexible environments include excessive task nodes and relations, as well as "spaghetti" process models, such as in

Figure 6.1. In order to tackle this concern, researchers have approached trace clustering from the broad categories of vector space clustering, context-aware clustering, and model-based sequence clustering.

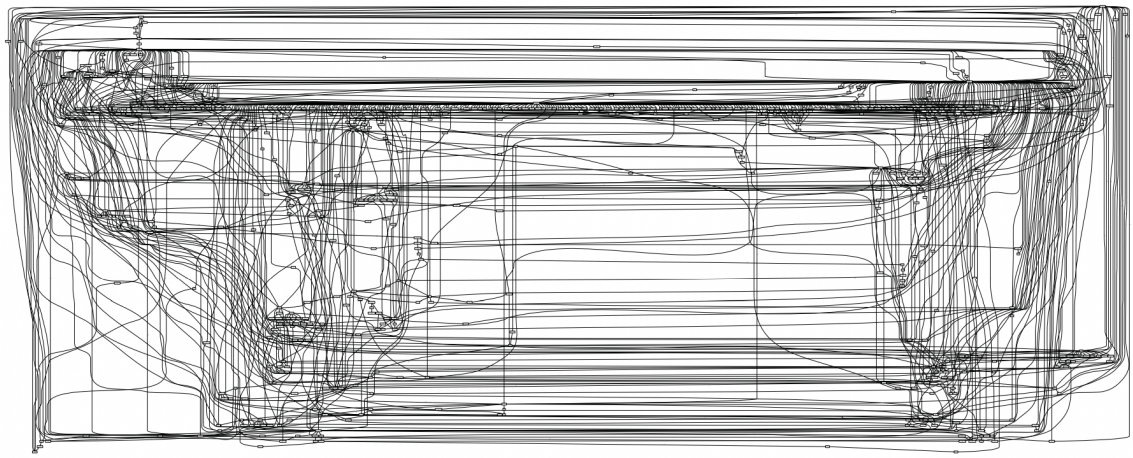

Figure 6.1: Spaghetti Process Model Example [1]

"Spaghetti process describing the diagnosis and treatment of 2765 patients in a Dutch hospital. The process model was constructed based on an event log containing 114,592 events. There are 619 different activities (taking event types into account) executed by 266 different individuals (doctors, nurses, etc.)" [1]

Vector space trace-clustering methods focus on using attributes of event logs as features for traditional clustering algorithms with the goal of discovering simpler sub-models. One of the first papers to investigate vector space trace-clustering was published by Greco et al. in 2006 and is titled "Discovering Expressive Process Models by Clustering Log Traces" [29]. Within the paper, Greco et al. propose a novel process mining algorithm that uses k-means clustering to identify process-variants and model them via distinct workflow schema. Provided to the k-means algorithm as features were activities, as well as transitions within event logs. Building on this approach in 2009, Song et al. proposed a new method based on log profiles that incorporated additional features as well as different clustering algorithms to improve

performance [61]. By organizing traces into profile vectors representing transitions, case attributes, event attributes, and performance, the researchers demonstrated increased effectiveness of clustering approaches to partition event logs. Similarly within our first methodology, we embrace a vector-space approach to cluster traces. However, instead of using low level attributes such as case and event characteristics, we focus our vector construction on binary temporal attributes that have proven to differentiate some traces within a larger set.

Context-aware clustering methods adapt the way in which the control-flow of context-aware information is handled. To do this, R.P. and Aalst proposed a method in 2009 where traces are clustered based on generic edit distance [55]. In their approach, R.P. and Aalst define an automated way to derive costs of edit operations substitution, insertion, and deletion. These costs are then used to ensure that the context and ordering of traces are preserved when clustering based on edit distance. Improving upon this approach, Bose and van der Aalst also propose a new approach where a vector-space model is used with conserved patterns as its feature-set [12]. By defining patterns of maximal repeats, super maximal repeats, and near super maximal repeats, context-aware feature sets are developed that allow traces to be converted into vectors representing the number of occurrences of features. Within our clustering methodology, we also embrace the identification of temporal patterns to distinguish elements. Additionally, like Bose and van der Aalst, we embrace a vector-space model using conserved patterns as a feature set for clustering. However, different from Bose and van der Aalst where patterns are constrained to a finite set of predetermined templates, the patterns our approach can use are limited only by the expressive power of LTL.

Model-based sequence clustering methods approach trace-clustering from a unique angle that borrows techniques generally used within the field of bioinformatics. In bioinformatics, the concept of sequence clustering is often used to make sense of seemingly meaningless sequences, such as automatically establishing families of proteins from large protein datasets [24]. By using these established sequence clustering techniques, model-based sequence clustering attempts to leverage their ability to operate with less dependence on information about business logic that other clustering methods may inaccurately assume exist within event logs. In accordance with model-based sequence clustering, Ferreira et al. introduced a trace-clustering approach in 2007, where the researchers used first-order Markov models as well as the Expectation-Maximization algorithm [25]. By applying this technique to human workflow and database system trace settings, the researchers successfully demonstrate the ability of their technique to recover original behaviour via an automated modelling approach. As an extension of model-based sequence clustering, Weerdt et al. propose a new method with a premise observation that existing trace-clustering methods "suffer from a large divergence between the clustering bias and the evaluation bias" [67]. Weerdt et al.'s 2013 approach tackles this issue by attempting to find an optimal distribution of traces amongst clusters that maximizes combined accuracy of the resulting process models. By employing a top-down greedy algorithm, an active learning inspired approach is applied to outperform existing trace-clustering methods.

While each of these approaches demonstrate various degrees of success establishing clusters and separating traces into defined groups, they all lack the ability to explain the differences between clusters. Since these methodologies exist within the context of business process mining, the objective of their research is to provide better inputs

for process-model generation and process-analysis. Although the output clusters of this thesis methodology could be used in a similar respect, the primary objective is focused on the analysis and source of differentiation. In order to translate the findings of these related works into contrastive explanations, several additional steps would need to be undertaken, none of which would produce sound results. A simple option within the field of process mining would likely be to translate the discovered clusters into process-maps, then compare them manually to search for differences. However this approach would be intuitively time-consuming and potentially lack accuracy. Alternatively, there are existing frameworks designed to automatically compare process models [3, 65, 48]; however their outputs are designed to provide a binary evaluation of equivalence, which would offer little value in this context. As an improvement to binary equivalence, Aalst et al. propose a methodology to evaluate the degree of similarity on an ordinal scale from 0 (completely different) to 1 (identical) based on causal dependencies and ordering [4]. Evaluating the degree of similarity between clusters may represent an effective tool to measure the strength of delineation, but it still fails to provide concrete specifications to define the differences.

Chapter 7

Conclusion and Future Work

7.1 Comparison of Vector-Space and Tree-Based Approaches

While our two unique methodologies are designed to tackle the same problem of clustering and delineating discrete time-series data in the form of traces, they embrace very different strategies. Both approaches accept identical inputs and produce similar clear and concise definitions of identified clusters via LTL, but they achieve these objectives through very different procedures. Comparing our two methods, these diverse approaches each possess relative strengths and weaknesses, leading to varying effectiveness based on their application.

Since our vector-space method embraces traditional clustering algorithms which use ultra-efficient distance metrics to calculate dissimilarity, this method is much easier to scale, as compared to our tree-based approach. This integration with traditional clustering algorithms via the creation of a standard input dataframe also allows a variety of different algorithms to be efficiently tested for fit, promoting the adaptability of our vector-space approach. On the contrary, our tree-based method relies entirely on LTL splits via Monte Carlo iterations of BayesLTL to evaluate similarity, which

means complexity will typically be greater with less room for reductions. The quantity of features used can also be adjusted within our vector-space approach, which can result in less calculations of BayesLTL, as desired. These characteristics are especially important for scalability, considering the fact that iterations of BayesLTL represents the bulk of complexity for both approaches. Overall, the scalability characteristics of our vector-space approach demonstrate favorability towards this method when π contains a very large quantity of traces to delineate.

Another important topic of comparison for our two approaches is their ability to accurately account for and represent all traces within the input set π. Since our tree-based approach clusters and delineates traces simultaneously via recursive splits from a root node, all traces are always fully accounted for by nature. This also tends to lead to more equally sized clusters, since at each node, the objective of maximizing information gain is sought. On the other hand, our vector-space method approaches clustering and delineation as a two-step process, where potential for misalignment exists when transitioning steps. We identify this misalignment as rare within our simulated evaluation domains, however, the possibility remains within our vector-space method for clusters to be undefined via conjuntive LTL explanations, or if approximation is used, traces may exist within multiple clusters or no clusters; these negative outcomes are impossible in our tree-based approach. Our vector-space method is also capable of continuing search past the first discovered definition, which allows multiple accurate definitions to be discovered for each cluster, and vetted by the user for interestingness; this cannot occur within our tree-based approach because cluster definitions are interdependent. Finally, with our tree-based approach, the tree structure allows for greater interpretability, since formula derivation can be

understood component-by-component via the path to the root.

Although both of our approaches achieve similar objectives, their differing strategies lead to unique outcomes. If scalability is a primary consideration, our vector-space method may be preferred due to its adaptability with feature set size, and its integration with ultra-efficient traditional clustering algorithms. A desire for multiple cluster explanations may also lead to choosing our vector-space method. However, if consistency and completeness of cluster definitions are valued more, our tree-based approach may be the better option. Ultimately however, since both approaches accept the same input and target the same solution, there may be benefit in establishing variety by using both and accepting the most interesting solution for a given problem.

7.2 Summary

The objective of this research is to explore the topic of discrete time-series clustering and examine its practical application within simulated environments. By establishing two unique methodologies to cluster and delineate raw trace data, this thesis aims to advance the ability of researchers to achieve powerful insights when analyzing discrete time-series data.

In pursuit of these research intentions, our first methodology in Chapter 4, proposes a vector-space clustering approach, which constructs context-aware temporal features to identify like-groups of traces. Through the random sampling of smaller subsets from the parent set and applying the BayesLTL framework [39], we identify representative contrastive LTL statements, by which traces can be evaluated for entailment. Using traditional clustering algorithms, followed by the revisitation of features via the identification of conjunctive sets of LTL to differentiate clusters, we

demonstrate the effectiveness of this approach, as applied to six benchmark domains and three unique vocabulary sizes from the International Planning Competition [26]. We furthermore introduce an approximation-based approach which uses BayesLTL to explain clusters via pairwise or one-vs-all analysis, and is proven effective to identify even more compact LTL statements at the slight cost of definition completeness.

In our second methodology in Chapter 5, we reimagine the trace clustering and delineation challenge by proposing a novel tree generation technique which allows k clusters to be discovered and described. By embracing a Monte Carlo node-splitting approach, our algorithm seeks balance to contrastively divide any given set of plan traces into two sets with an accompanying temporal logic specification satisfying one of the sets. Recursing this procedure, we demonstrate the effectiveness of our approach to cluster and delineate plan traces from our evaluation dataset of benchmark domains, allowing temporal logic specifications to evoke insight at each level of the resulting tree. We also compare the effectiveness of our two methodologies and discuss the criteria that may lead to choosing one over the other. Finally, we highlight the contributions of our two methodologies in the context of related work in Chapter 6.

By proposing two unique methodologies, this thesis addresses the difficult task of discrete time-series clustering and delineation from two very different angles. In exploration of these distinct strategies, we discover high levels of performance associated with both approaches in six simulated domains with three vocabulary sizes. Given this thorough investigation, we conclude that both of our proposed methodologies are capable of evoking unique and applicable insights, empowering researchers to observe discrete time-series data from a novel lens.

7.3 Limitations and Future Work

Within our tree-based approach, we identify scalability as a potential limitation based on our findings in Section 5.2.3; investigating the relationship between node size and splitting time, we discovered a positive near-linear connection between these two variables. This relationship is present for our sample domains in Figure 5.6. From this finding, we can infer that as the quantity of traces becomes very large, the execution time required to generate trees may also grow similarly. To address this concern, in Section 5.2.6, we investigated the potential for an intelligent splitting process to reduce complexity associated with tree generation; the process that we tested is also depicted in Figure 5.9. Since iterations of BayesLTL [39] represent the bulk of complexity within our approach, reducing the quantity of these iterations offers powerful opportunity for efficiency improvements. While the intelligent-splitting method that we tested failed to converge, it allowed us to identify some key characteristics to facilitate success in future work. From our exploration, we can deduce that in order for an intelligent-splitting algorithm to be successful, it is likely that greater freedom from the initial random allocation of traces is necessary. This could perhaps take the form of randomized resets or other statistical methods to introduce noise. By strategically introducing variability to learn from previous splitting iterations, we believe that complexity can be reduced in future work.

Within our vector-space approach, limitation is represented via potential misalignments between our clustering step and interpretability step. Since explanations of clusters are independent from cluster identification, if conjunctive LTL cannot be found within reasonable search limits, clusters will remain undefined. Additionally, when using our approximation-based explanation method, traces may be allocated

into multiple cluster or no clusters. To mitigate this source of error, future work should investigate the integration of these two steps, such that misalignment is not possible. Within our approximation approach, a potential line of future work to combat this challenge could be the development of an iterable process that reduces or eliminates error via more than one explanation attempt. Within our conjunctive LTL approach, this error could be mitigated by employing strategic search to replace our current size-ordered exhaustive method. For example, we identify that LTL specifications that are selected by our conjunctive search strategy tend to be smaller than the central tendency of all LTL features; heuristics like this could be investigated and applied to improve search efficiency. Additionally, existing interpretability frameworks that work simultaneously with model training and prediction could be applied to promote a better understanding of resulting clusters. By innovating our vector-space approach to minimize misalignment between the method's two fundamental steps, the potential for better cluster definitions can be created within future research.

Next, to account for situations where labels may exist within the input to the algorithm, future work includes the adaptation of our methodology to enable this. Within our vector-space approach, this adaptation could begin with feature generation via iterations of BayesLTL amongst samples from the labelled groups. The clustering step could then be bypassed since labels already exist, then an identical conjunctive search strategy could be employed to describe the clusters via LTL.

There also exists potential to combine the strengths of both approaches to establish even stronger insights. A possibility of this hybrid approach could be identifying clusters via our tree-based method, then embracing feature generation and conjunctive search from our vector-space approach to enhance interestingness by generating

additional LTL explanations of clusters. Another possibility could be clustering traces via our vector-space methodology, then embracing a tree structure to explain clusters.

Finally, while both approaches are shown to be highly effective in clustering and delineating traces from the input dataset, it may be interesting to next use those discovered explanations to evaluate addition traces conforming with the identified systems. By applying a framework such as LtlFond2Fond [17], this could be formulated as a planning problem, where the discovered LTL is incorporated as an extended goal. Addition plans for each cluster could then be generated by using a top-k [37] or diverse planner [36]. By evaluating additional plans that are consistent with cluster definitions, additional insights could be discovered.

Bibliography

[1] W. Aalst. Process mining: Discovering and improving spaghetti and lasagna processes. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 1–7, 04 2011. doi: 10.1109/CIDM.2011.6129461.

[2] W. Aalst. *Process Mining: Data Science in Action*. Springer Berlin, 01 2016. ISBN 9783662498507. doi: 10.1007/978-3-662-49851-4.

[3] W. Aalst and T. Basten. Inheritance of workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, 270:125–203, 01 2002. doi: 10.1016/S0304-3975(00)00321-2.

[4] W. Aalst, A. Medeiros, and A. Weijters. Process equivalence: Comparing two process models based on observed behavior. In *Lecture Notes in Computer Science*, pages 129–144, 09 2006. ISBN 978-3-540-38901-9. doi: 10.1007/11841760_10.

[5] H. Abbasimehr, M. Shabani, and M. Yousefi. An optimized model using lstm network for demand forecasting. *Computers & Industrial Engineering*, 143:106435, 2020. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2020.106435.

[6] Q. Acton. *Advances in Machine Learning Research and Application: 2013 Edition*. ScholarlyEditions, 2013. ISBN 9781481670982.

[7] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *Proc. 20th Int. Conf. Very Large Data Bases VLDB*, 1215, 08 2000.

[8] M. O. Alassafi, M. Jarrah, and R. Alotaibi. Time series predicting of covid-19 based on deep learning. *Neurocomputing*, 468:335–344, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.10.035.

[9] D. Amir and O. Amir. Highlights: Summarizing agent behaviors to people. In *the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*, Stockholm, Sweden, July 2018 2018.

[10] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta. A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 04 2016. doi: 10.1145/2883817.2883843.

[11] R. Borgo, M. Cashmore, and D. Magazzeni. Towards providing explanations for ai planner decisions. *ArXiv*, abs/1810.06338, 2018.

[12] R. P. J. C. Bose and W. M. P. van der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In S. Rinderle-Ma, S. Sadiq, and F. Leymann, editors, *Business Process Management Workshops*, pages 170–181, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-12186-9.

[13] G. Box, G. Jenkins, and W. U. M. D. of STATISTICS. *Time Series Analysis: Forecasting and Control*. Holden-Day series in time series analysis and digital processing. Holden-Day, 1970. ISBN 9780816210947.

[14] A. Brunello, G. Sciavicco, and E. Stan. *Interval Temporal Logic Decision Tree Learning*, pages 778–793. Logics in Artificial Intelligence, 05 2019. ISBN 978-3-030-19569-4. doi: 10.1007/978-3-030-19570-0_50.

[15] T. Caliński and H. JA. A dendrite method for cluster analysis. *Communications in Statistics - Theory and Methods*, 3:1–27, 01 1974. doi: 10.1080/03610927408827101.

[16] A. Camacho and S. A. McIlraith. Learning interpretable models expressed in linear temporal logic. In *ICAPS*, 2019.

[17] A. Camacho, E. Triantafillou, C. Muise, J. A. Baier, and S. A. McIlraith. Non-deterministic planning with temporally extended goals: Ltl over finite and infinite traces. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3716–3724. AAAI Press, 2017.

[18] A. Cecconi, G. De Giacomo, C. Di Ciccio, F. Maggi, and J. Mendling. Measuring the interestingness of temporal logic behavioral specifications in process mining. *Information Systems*, page 101920, 11 2021. doi: 10.1016/j.is.2021.101920.

[19] T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 156–163. AAAI Press, 2017. ISBN 9780999241103.

[20] C. Chatfield. The holt-winters forecasting procedure. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 27(3):264–279, 1978. ISSN 00359254, 14679876.

[21] A. Coman and H. Muñoz Avila. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, page 946–951. AAAI Press, 2011.

[22] F. Dama and C. Sinoquet. Analysis and modeling to forecast in time series: a systematic review. *CoRR*, abs/2104.00164, 2021.

[23] D. Davies and D. Bouldin. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-1:224 – 227, 05 1979. doi: 10.1109/TPAMI.1979.4766909.

[24] A. Enright, S. Dongen, and C. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research*, 30:1575–84, 05 2002. doi: 10.1093/nar/30.7.1575.

[25] D. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira. Approaching process mining with sequence clustering: Experiments and findings. In *Proceedings of the 5th International Conference on Business Process Management*, BPM'07, page 360–374, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3540751823.

[26] M. Fox and D. Long. The 3rd international planning competition: Results and analysis. *ArXiv*, abs/1106.5998, 2003.

[27] M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus

plan repair. In *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'06, page 212–221. AAAI Press, 2006. ISBN 9781577352709.

[28] J. Gaglione, D. Neider, R. Roy, U. Topcu, and Z. Xu. Learning linear temporal properties from noisy data: A maxsat-based approach. In *ATVA*, volume 12971 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2021.

[29] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *Knowledge and Data Engineering, IEEE Transactions on*, 18:1010– 1027, 09 2006. doi: 10.1109/TKDE.2006.123.

[30] D. Grosse and R. Drechsler. Formal verification of ltl formulas for systemc designs. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, volume 5, pages V–V, 2003. doi: 10.1109/ISCAS.2003.1206243.

[31] M. Hammer and J. Champy. Reengineering the corporation: A manifesto for business revolution. *Business Horizons*, 36(5):90–91, 1993.

[32] B. Hayes and J. A. Shah. Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, pages 303–312, 2017.

[33] Y. Jiang and D. Lan. Probability model of rock climbing recognition based on information fusion sensor time series. *EURASIP Journal on Advances in Signal Processing*, 2021, 11 2021. doi: 10.1186/s13634-021-00816-5.

[34] S. Johansen. Statistical analysis of cointegration vectors. *Journal of Economic Dynamics and Control*, 12(2):231–254, 1988. ISSN 0165-1889. doi: https://doi.org/10.1016/0165-1889(88)90041-3.

[35] D. Kasenberg and M. Scheutz. Interpretable apprenticeship learning with temporal logic specifications. *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4914–4921, 2017.

[36] M. Katz and S. Sohrabi. Reshaping diverse planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):9892–9899, Apr. 2020. doi: 10.1609/aaai.v34i06.6543.

[37] M. Katz, S. Sohrabi, O. Udrea, and D. Winterer. A novel iterative approach to top-k planning. In *ICAPS*, 2018.

[38] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley, 1990. ISBN 978-0-47031680-1.

[39] J. Kim, C. Muise, A. Shah, S. Agarwal, and J. Shah. Bayesian inference of linear temporal logic specifications for contrastive explanations. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5591–5598. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/776.

[40] A. Koul, S. Greydanus, and A. Fern. Learning finite state representations of recurrent policy networks. *CoRR*, abs/1811.12530, 2018.

[41] O. Kupferman. Sanity checks in formal verification. In C. Baier and H. Hermanns, editors, *CONCUR 2006 – Concurrency Theory*, pages 37–51, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37377-3.

[42] I. Lage, D. Lifschitz, F. Doshi velez, and O. Amir. Exploring computational user models for agent policy summarization. *IJCAI : proceedings of the conference*, 28:1401–1407, 08 2019.

[43] T. Latvala. *Automata-theoretic and bounded model checking for linear temporal logic*. PhD thesis, Väitöskirja :, Espoo, 2005. Tiivistelmä ja 5 erip.

[44] C. Lemieux, D. Park, and I. Beschastnikh. General ltl specification mining (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 81–92, 2015. doi: 10.1109/ASE.2015.71.

[45] M. Magnaguagno, R. Pereira, M. D. Móre, and F. Meneguzzi. A tool to develop classical planning domains and visualize heuristic state-space search. In *2017 Workshop on User Interfaces and Scheduling and Planning (UISP@ICAPS)*, 2017.

[46] M. C. Magnaguagno, R. Pereira, and F. Meneguzzi. Dovetail - an abstraction for classical planning using a visual metaphor. In *The 29th Florida Artificial Intelligence Research Society Conference*, 2016.

[47] R. Malladi and P. Dheeriya. Time series analysis of cryptocurrency returns and volatilities. *Journal of Economics and Finance*, 45:75–94, 09 2020. doi: 10.1007/s12197-020-09526-4.

[48] R. Milner and R. Milner. *A Calculus of Communicating Systems*. Springer Berlin Heidelberg, 1980.

[49] D. Neider and I. Gavran. Learning linear temporal properties, 2018.

[50] T. Nguyen, M. Do, A. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190:1–31, 10 2012. doi: 10.1016/j.artint.2012.05.005.

[51] A. H. Nury, M. S. Koch, and M. J. Alam. Time series analysis and forecasting of temperatures in the sylhet division of bangladesh. In *Proceedings of 4th International Conference on Environmental Aspects of Bangladesh*, 2013.

[52] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977. doi: 10.1109/SFCS.1977.32.

[53] M. Ramírez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 1121–1126. AAAI Press, 2010.

[54] K. Y. Rozier. Survey: Linear temporal logic symbolic model checking. *Comput. Sci. Rev.*, 5(2):163–203, May 2011. ISSN 1574-0137. doi: 10.1016/j.cosrev.2010.06.002.

[55] J. C. B. R.P. and W. Aalst. Context aware trace clustering: Towards improving process mining results. In *Proceedings of the Ninth SIAM International Conference on Data Mining*, 04 2009. doi: 10.1137/1.9781611972795.35.

[56] B. Seegebarth, F. Müller, B. Schattenberg, and S. Biundo-Stephan. Making hybrid plans more clear to human users - a formal approach for generating sound explanations. In *ICAPS*, 2012.

[57] V. Seimetz, R. Eifler, and J. Hoffmann. Learning temporal plan preferences from examples: An empirical study. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4160–4166. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/572. Main Track.

[58] A. Shah, P. Kamath, J. A. Shah, and S. Li. Bayesian inference of temporal task specifications from demonstrations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[59] S. Siami-Namini, N. Tavakoli, and A. S. Namin. The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3285–3292, 2019. doi: 10.1109/BigData47090.2019.9005997.

[60] S. Sohrabi, A. Riabov, and O. Udrea. Plan recognition as planning revisited. In *IJCAI*, 2016.

[61] M. Song, C. W. Günther, and W. M. P. van der Aalst. Trace clustering in process mining. In D. Ardagna, M. Mecella, and J. Yang, editors, *Business Process Management Workshops*, pages 109–120, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[62] S. Sreedharan, S. Srivastava, and S. Kambhampati. Tldr: Policy summarization for factored ssp problems using temporal abstractions. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1):272–280, Jun. 2020.

[63] N. Topin and M. Veloso. Generation of policy-level explanations for reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 2514–2521, Jul. 2019. doi: 10.1609/aaai.v33i01.33012514.

[64] O. Triebe, N. P. Laptev, and R. Rajagopal. Ar-net: A simple auto-regressive neural network for time-series. *ArXiv*, abs/1911.12436, 2019.

[65] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43:555–600, 1996.

[66] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of the VIII Banff Higher Order Workshop Conference on Logics for Concurrency: Structure versus Automata: Structure versus Automata*, page 238–266, Berlin, Heidelberg, 1996. Springer-Verlag. ISBN 3540609156.

[67] J. Weerdt, S. vanden Broucke, J. Vanthienen, and B. Baesens. Active trace clustering for improved process discovery. *IEEE Trans. Knowl. Data Eng.*, 25: 2708–2720, 12 2013. doi: 10.1109/TKDE.2013.64.

[68] Y. Zhang, S. Sreedharan, A. Kulkarni, T. Chakraborti, H. Zhuo, and S. Kambhampati. Plan explicability and predictability for robot task planning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1313–1320, 2017.

Appendix A

Vector Space Methodology

Cluster	LTL Definition
0	atmostonce: ((on object d object a)), ((ontable object m))
1	atmostonce: ((on object f object c)), ((ontable object m))
2	sometime_before: ((clear object a),(holding object m))
3	sometime_before: ((holding object f),(holding object o))
4	atmostonce: ((on object f object c)), ((ontable object j))
5	sometime_before: ((holding object m),(ontable object b))
6	atmostonce: ((on object g object l)), ((ontable object e))
7	atmostonce: ((holding object i)), ((ontable object l))

Table A.1: Blocks Examples of Clusters and LTL using Vector Space Method

Examples of LTL cluster definitions derived from the Blocks domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Cluster	LTL Definition
0	eventual: ((carry object ball23 object right)), ((carry object ball29 object right))
1	eventual: ((carry object ball32 object right)), ((carry object ball9 object right))
2	eventual: ((carry object ball11 object left)), ((carry object ball7 object left))
3	eventual: ((carry object ball8 object right)), ((carry object ball9 object right))
4	eventual: ((carry object ball11 object left)), ((carry object ball9 object right))
5	eventual: ((at object ball25 object roomb)), ((carry object ball2 object right)), ((carry object ball4 object left))
6	eventual: ((carry object ball28 object left))
7	eventual: ((carry object ball16 object left))

Table A.2: Gripper Examples of Clusters and LTL using Vector Space Method

Examples of LTL cluster definitions derived from the Gripper domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Cluster	LTL Definition
0	response: ((at rover rover2 waypoint waypoint5),(at rover rover2 waypoint waypoint1))
1	eventual: ((at rover rover2 waypoint waypoint10)), ((calibrated camera camera2 rover rover1))
2	response: ((at rover rover0 waypoint waypoint2),(empty store rover3store))
3	response: ((at rover rover2 waypoint waypoint10),(at rover rover0 waypoint waypoint8)), ((have_soil_analysis rover rover2 waypoint waypoint10),(empty store rover0store))
4	response: ((at rover rover0 waypoint waypoint10),(have_image rover rover1 objective objective1 mode high_res)), ((at_soil_sample waypoint waypoint0),(at rover rover1 waypoint waypoint6))
5	response: ((at_rock_sample waypoint waypoint8),(at_soil_sample waypoint waypoint0))
6	response: ((at_soil_sample waypoint waypoint0),(at rover rover3 waypoint waypoint1))
7	eventual: ((at rover rover0 waypoint waypoint5))

Table A.3: Rovers Examples of Clusters and LTL using Vector Space Method

Examples of LTL cluster definitions derived from the Rovers domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Cluster	LTL Definition
0	response: ((pointing object satellite5 object star15),(pointing object satellite5 object planet21))
1	response: ((pointing object satellite5 object star15),(pointing object satellite5 object planet21))
2	eventual: ((pointing object satellite5 object planet21))
3	sometime_before: ((pointing object satellite6 object planet23),(pointing object satellite1 object phenomenon5))
4	sometime_before: ((pointing object satellite6 object planet23),(pointing object satellite1 object phenomenon5))
5	eventual: ((pointing object satellite5 object planet21))
6	response: ((pointing object satellite5 object planet14),(pointing object satellite6 object planet23))
7	response: ((pointing object satellite5 object star15),(pointing object satellite6 object star3))

Table A.4: Satellite Examples of Clusters and LTL using Vector Space Method

Examples of LTL cluster definitions derived from the Satellite domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Cluster	LTL Definition
0	eventual: ((loaded goods goods2 truck truck4 level level1))
1	response: ((loaded goods goods10 truck truck1 level level0),(at truck truck1 depot depot1)), ((ready-to-load goods goods5 market market1 level level1),(loaded goods goods10 truck truck3 level level1))
2	sometime_before: ((loaded goods goods10 truck truck2 level level1),(loaded goods goods2 truck truck3 level level1))
3	eventual: ((loaded goods goods7 truck truck2 level level1))
4	response: ((loaded goods goods7 truck truck1 level level1),(at truck truck1 market market3))
5	eventual: ((loaded goods goods1 truck truck2 level level1))
6	global: ((loaded goods goods3 truck truck2 level level0)), ((on-sale goods goods4 market market1 level level0))
7	response: ((loaded goods goods1 truck truck1 level level0),(on-sale goods goods5 market market1 level level0)), ((on-sale goods goods3 market market2 level level1),(on-sale goods goods10 market market1 level level1))

Table A.5: TPP Examples of Clusters and LTL using Vector Space Method

Examples of LTL cluster definitions derived from the TPP domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Cluster	LTL Definition
0	response: ((in object person2 object plane2),(in object person6 object plane2))
1	eventual: ((at object plane2 object city9))
2	response: ((at object plane2 object city11),(at object plane2 object city2))
3	response: ((at object plane5 object city5),(at object person11 object city4)), ((in object person6 object plane2),(at object plane4 object city1))
4	atmostonce: ((fuel-level object plane4 object fl1))
5	response: ((at object person2 object city10),(at object plane5 object city6))
6	response: ((in object person7 object plane2),(in object person5 object plane2))
7	stability: ((at object plane2 object city6))

Table A.6: ZenoTravel Examples of Clusters and LTL using Vector Space Method

Examples of LTL cluster definitions derived from the ZenoTravel domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Appendix B

Tree-based Methodology

Size	Blocks15 Example Formulas
2	eventual: ((on object m object g))
5	sometime_before: ((clear object m),(on object g object f))
6	atmostonce: ((ontable object m))
11	sometime_before: ((holding object j),(clear object g)), ((holding object m),(ontable object o))
13	atmostonce: ((holding object m)), ((ontable object o))
16	sometime_before: ((clear object i),(clear object d)), ((clear object m),(clear object c)), ((clear object m),(ontable object g))
19	atmostonce: ((on object f object c)), ((on object l object f)), ((ontable object m))
25	atmostonce: ((on object a object k)), ((on object f object c)), ((on object o object f)), ((ontable object l))

Table B.1: Blocks Varying Sizes of LTL Examples using Tree-based Method

Examples of LTL specifications discovered within trees from the Blocks domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Size	Gripper15 Example Formulas
2	eventual: ((carry object ball8 object left))
5	eventual: ((carry object ball11 object left)), ((carry object ball6 object left))
6	response: ((carry object ball27 object right),(carry object ball32 object left))
7	eventual: ((carry object ball14 object right)), ((carry object ball22 object left)), ((carry object ball30 object right))
9	eventual: ((at object ball1 object roomb)), ((carry object ball4 object left)), ((carry object ball7 object right)), ((carry object ball9 object right))
11	eventual: ((at object ball2 object roomb)), ((at-robby object roomb)), ((carry object ball15 object left)), ((carry object ball27 object right)), ((carry object ball29 object right))
13	response: ((at object ball3 object rooma),(carry object ball32 object right)), ((carry object ball27 object right),(carry object ball9 object left))
15	eventual: ((at object ball16 object rooma)), ((at object ball28 object rooma)), ((carry object ball12 object right)), ((carry object ball15 object left)), ((carry object ball24 object left)), ((carry object ball28 object left)), ((carry object ball8 object right))
19	response: ((at object ball14 object rooma),(carry object ball22 object left)), ((at object ball22 object rooma),(carry object ball6 object left)), ((carry object ball25 object right),(carry object ball4 object left))
25	response: ((at object ball30 object roomb),(at object ball16 object roomb)), ((carry object ball11 object left),(at object ball1 object roomb)), ((carry object ball15 object left),(carry object ball21 object right)), ((carry object ball24 object left),(carry object ball7 object right))

Table B.2: Gripper Varying Sizes of LTL Examples using Tree-based Method

Examples of LTL specifications discovered within trees from the Gripper domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Size	Rovers15 Example Formulas
2	eventual: ((have_soil_analysis rover rover2 waypoint waypoint10))
5	sometime_before: ((full store rover0store),(have_image rover rover0 objective objective1 mode low_res))
6	response: ((have_soil_analysis rover rover0 waypoint waypoint2),(have_image rover rover1 objective objective1 mode low_res))
7	eventual: ((at rover rover0 waypoint waypoint0)), ((at rover rover2 waypoint waypoint10)), ((at_soil_sample waypoint waypoint2))
11	sometime_before: ((at rover rover0 waypoint waypoint0),(have_image rover rover0 objective objective1 mode high_res)), ((at rover rover0 waypoint waypoint7),(have_image rover rover0 objective objective4 mode high_res))
13	response: ((at rover rover2 waypoint waypoint4),(have_image rover rover1 objective objective1 mode low_res)), ((at_soil_sample waypoint waypoint8),(have_soil_analysis rover rover0 waypoint waypoint0))
19	response: ((communicated_image_data objective objective1 mode low_res),(communicated_soil_data waypoint waypoint0)), ((empty store rover3store),(empty store rover2store)), ((have_rock_analysis rover rover3 waypoint waypoint2), (communicated_soil_data waypoint waypoint8))
25	response: ((at rover rover2 waypoint waypoint4),(have_rock_analysis rover rover3 waypoint waypoint1)), ((at rover rover3 waypoint waypoint2), (at rover rover3 waypoint waypoint4)), ((calibrated camera camera2 rover rover1),(empty store rover0store)), ((have_soil_analysis rover rover0 waypoint waypoint10),(at rover rover0 waypoint waypoint8))

Table B.3: Rovers Varying Sizes of LTL Examples using Tree-based Method

Examples of LTL specifications discovered within trees from the Rovers domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Size	Satellite15 Example Formulas
2	eventual: ((pointing object satellite5 object planet21))
5	sometime_before: ((have_image object phenomenon5 object spectrograph2),(pointing object satellite1 object phenomenon10))
6	response: ((pointing object satellite3 object star4),(pointing object satellite5 object planet21))
11	sometime_before: ((calibrated object instrument13),(pointing object satellite5 object planet14)), ((have_image object phenomenon10 object image1), (have_image object phenomenon16 object image1))
13	response: ((pointing object satellite1 object phenomenon5),(pointing object satellite1 object phenomenon16)), ((pointing object satellite1 object star18), (have_image object planet6 object spectrograph2))
16	sometime_before: ((have_image object planet21 object infrared0),(power_avail object satellite4)), ((have_image object star15 object thermograph4), (calibrated object instrument13)), ((pointing object satellite6 object star3),(pointing object satellite3 object planet14))
19	response: ((have_image object planet11 object image1),(pointing object satellite6 object star13)), ((pointing object satellite1 object phenomenon10), (pointing object satellite1 object planet11)), ((pointing object satellite6 object planet21),(power_avail object satellite0))

Table B.4: Satellite Varying Sizes of LTL Examples using Tree-based Method

Examples of LTL specifications discovered within trees from the Satellite domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Size	TPP15 Example Formulas
2	eventual: ((loaded goods goods8 truck truck2 level level1))
5	eventual: ((at truck truck2 market market4)), ((on-sale goods goods5 market market1 level level0))
6	atmostonce: ((at truck truck2 market market2))
7	eventual: ((at truck truck2 market market2)), ((loaded goods goods10 truck truck4 level level1)), ((on-sale goods goods10 market market2 level level0))
9	eventual: ((at truck truck1 market market3)), ((loaded goods goods1 truck truck3 level level0)), ((on-sale goods goods4 market market3 level level0)), ((ready-to-load goods goods5 market market1 level level1))
11	eventual: ((at truck truck4 market market4)), ((loaded goods goods7 truck truck1 level level1)), ((ready-to-load goods goods1 market market4 level level0)), ((ready-to-load goods goods6 market market2 level level1)), ((stored goods goods2 level level3))
13	response: ((ready-to-load goods goods1 market market4 level level1),(on-sale goods goods5 market market1 level level0)), ((ready-to-load goods goods6 market market2 level level1),(ready-to-load goods goods10 market market1 level level1))
16	sometime_before: ((loaded goods goods10 truck truck3 level level1),(at truck truck2 market market3)), ((loaded goods goods8 truck truck1 level level1), (ready-to-load goods goods9 market market3 level level0)), ((on-sale goods goods5 market market1 level level0),(loaded goods goods2 truck truck1 level level1))
19	atmostonce: ((loaded goods goods10 truck truck2 level level1)), ((loaded goods goods2 truck truck2 level level1)), ((on-sale goods goods9 market market3 level level2))
31	response: ((at truck truck2 market market2),(stored goods goods1 level level0)), ((at truck truck4 depot depot1),(loaded goods goods3 truck truck2 level level0)), ((loaded goods goods1 truck truck1 level level1),(on-sale goods goods2 market market4 level level0)), ((loaded goods goods1 truck truck3 level level1), (ready-to-load goods goods3 market market1 level level0)), ((on-sale goods goods1 market market2 level level1),(stored goods goods10 level level0))

Table B.5: TPP Varying Sizes of LTL Examples using Tree-based Method

Examples of LTL specifications discovered within trees from the TPP domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.

Size	ZenoTravel15 Example Formulas
2	eventual: ((at object plane2 object city5))
5	eventual: ((at object plane5 object city4)), ((in object person1 object plane3))
6	response: ((fuel-level object plane4 object fl0),(at object plane5 object city1))
7	eventual: ((at object plane2 object city6)), ((fuel-level object plane2 object fl3)), ((in object person11 object plane2))
9	stability: ((at object plane2 object city2))
11	sometime_before: ((in object person11 object plane2),(at object plane2 object city8)), ((in object person7 object plane3), (at object person15 object city6))
13	response: ((at object plane5 object city6),(at object plane2 object city2)), ((fuel-level object plane1 object fl0),(fuel-level object plane3 object fl1))
16	sometime_before: ((in object person11 object plane5),(at object plane5 object city6)), ((in object person4 object plane4), (fuel-level object plane3 object fl0)), ((in object person7 object plane3),(at object plane2 object city3))
17	stability: ((at object plane2 object city2)), ((at object plane4 object city9))
19	response: ((at object person4 object city5),(fuel-level object plane4 object fl1)), ((fuel-level object plane5 object fl1),(at object person11 object city4)), ((in object person6 object plane2),(at object person13 object city8))
25	response: ((at object person10 object city11),(fuel-level object plane3 object fl1)), ((at object person13 object city8),(in object person13 object plane2)), ((at object plane5 object city5),(in object person4 object plane2)), ((in object person5 object plane2),(in object person1 object plane2))

Table B.6: ZenoTravel Varying Sizes of LTL Examples using Tree-based Method

Examples of LTL specifications discovered within trees from the ZenoTravel domain with a vocabulary size of 15. See Table 2.2 for descriptions of templates.