GRAPHICAL EVOLVING TRANSFORMATION SYSTEM MACHINE

A Dissertation

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

THANH THIEU

Prof. Dmitry Korkin, Dissertation Supervisor

May 2015

The undersigned, appointed by the dean of the Graduate School, have examined the dissertation entitled

GRAPHICAL EVOLVING TRANSFORMATION SYSTEM MACHINE

presented by Thanh Thieu,

a candidate for the degree of Doctor of Philosophy,

and hereby certify that, in their opinion, it is worthy of acceptance.

Prof. Dmitry Korkin

Prof. Yi Shang

Prof. Chi-Ren Shyu

Prof. Alina Zare

To my mom and dad, for their unconditioned love, has given me courage to step on a unprecedented journey

To my wife, for her unwavering support to what I do

To my son, the love of my life

To my sister, who is tirelessly pursuing her dream

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my advisor, Dr. Dmitry Korkin, who has been supporting me not only in academic research, but also emotional balance. His patience and wit are the most important ingredients that push me ahead whenever the research faces challenges. Then, I would like to show gratitude towards my excellent committee that have given me feedbacks and pointers that transformed my work to a higher level. I also thanks to my lab members, who show me examples of pushing through years of a Ph.D. program. Finally, I would like to express gratitude to my parents, my wife, and my son for their unconditioned love and unwavering encouragement.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF FIGURES vi

LIST OF TABLES vii

ABSTRACT viii

CHAPTER 1: INTRODUCTION 1

1.1. Motivation 1

1.2. Objective 2

1.3. Dissertation Structure 4

CHAPTER 2: MACHINE LEARNING WITH STRUCTURED INPUT-OUTPUT 6

2.1. Structured Prediction 6

2.2. Statistical Relational Learning 8

2.3. Graph-based Pattern Recognition 9

2.4. Syntactic Pattern Recognition 12

CHAPTER 3: EVOLVING TRANSFORMATION SYSTEM 17

3.1. Literature Review 17

3.2. Elements of ETS 20

3.2.1. Structure Measurement Device 20

3.2.2. Primitives 20

3.2.3. Structs and Composites 22

3.2.4. Transformations 26

3.2.5. Class-centric Modelling 31

CHAPTER 4: GRAPH-BASED EVOLVING TRANSFORMATION SYSTEM MACHINE 37

4.1. Introduction 37

4.2. Feature Graphs 38

4.3. Graphical Representation of ETS Formalism 42

4.3.1. Fundamental Entities 42

4.3.2. Class Representation 46

4.4. Induction Problem Formulation 50

4.5. Objective Function 51

4.5.1. Likelihood Probability 52

4.5.2. Family Resemblance Typicality 56

4.5.3. Minimum Description Length 63

4.5.4. Normalizing Measurements 67

4.6. Induction Algorithm 68

4.6.1. Graph Alignment 69

4.6.2. Simultaneous Graph Summarization 77

4.6.3. Hill Climbing with Random Restart 82

4.7. Prediction Problem Formulation 84

4.8. Prediction Algorithm 87

4.9. The Machine 90

CHAPTER 5: CLASSIFYING HUMAN ACTIONS IN STILL IMAGES USING NORMALIZED VIEW 2D POSE GRAPHS 92

5.1. Introduction 92

5.2. Data Collection 93

5.3. Feature Extraction 100

5.4. Datasets Distribution 110

5.5. Performance 113
CONCLUSION 116
Appendix A: Induction Performance 118
A.1. Measurement Ranges 119
A.2. Family Resemblance 121
A.3. Description Length 127
A.4. Likelihood 128
Appendix B: Prediction Performance 131
B.1. Walking 133
B.2. Running 136
B.3. Jumping 139
B.4. Riding Bike 142
BIBLIOGRAPHY 145
VITA 159

LIST OF FIGURES

Figure 1: Illustration of three primitives 21

Figure 2: Illustration of a struct. 23

Figure 3: Illustration of a struct composition 24

Figure 4: Illustration of multiple representational levels of a structural process 27

Figure 5: Illustration of a transformation in 2D 28

Figure 6: Application of a transformation. 30

Figure 7: A potato head family model 34

Figure 8: Illustration of a class representation and a constructive history 35

Figure 9: The ETS ecosystem 36

Figure 10: Symbolic expansion of semantic features. 40

Figure 11: Graphical representation of fundamental gETS elements 45

Figure 12: Forming a history of a square from a model of equilateral polygons 49

Figure 13: Constructing a super-composite from two component composites 75

Figure 14: Summarization of a super-composite 78

Figure 15: The gETS Machine 91

Figure 16: Samples from four action classes 96

Figure 17: 3D pose inferred from 2D joint annotation.. 99

Figure 18: Normalized-view projection of 3D poses 104

Figure 19: Extracting angle features from normalized 2D projections 106

Figure 20: A complete composite graph representation. 109

Figure 21: A super-composite of Walking class 113

LIST OF TABLES

Table 1: The representational dilemma. 10

Table 2: Similarity and difference between graph grammar and graph prototype. 13

Table 3: Concept correspondence between ETS and CFGs..... 31

Table 4: Differences between ETS and CFGs. 32

Table 5: Data distribution of four action classes according to VOC's splitting. 94

Table 6: Training and testing datasets distribution of four action classes. 110

Table 7: Average precision of state-of-the-art methods..... 115

Table 8: Precision and F-measure of *complete* history test..... 115

GRAPHICAL EVOLVING TRANSFORMATION SYSTEM MACHINE

Thanh Thieu
Dr. Dmitry Korkin, Dissertation Supervisor

ABSTRACT

For years, scientists have challenged the machine intelligence problem, which aims at building algorithms that can learn patterns from evidence and use the knowledge to solve unseen problems. Learning classes of objects followed by the classification of objects into their classes is a common task in machine intelligence. For this task, two objects representation schemes are often used: a vector-based representation, and a graph–based representation. While the vector representation has sound mathematical background and optimization tools, it lacks the ability to encode relations between the patterns and their parts, thus lacking the complexity of human perception. On the other hand, the graph-based representation naturally captures the intrinsic structural properties, but available algorithms usually have exponential complexity. In this work, we build an inductive learning algorithm that relies on graph-based representation of objects and their classes, and test the framework on a competitive dataset of human actions in static images. The method incorporates three primary measures of class representation: likelihood probability, family resemblance typicality, and minimum description length. Empirical benchmarking shows that the method is robust to the noisy input, scales well to real-world datasets, and achieves comparable performance to current learning techniques. Moreover, our method has the advantage of intuitive representation regarding both patterns and class representation. While applied to a specific problem of human pose

recognition, our framework, named graphical Evolving Transformation System (gETS), can have a wide range of applications and can be used in other machine learning tasks.

CHAPTER 1

INTRODUCTION

1.1. Motivation

Recently, solving structured data problems receives lots of interest in the machine learning community. Problems with structured input-output includes: parsing dependency structures of sentences [1], protein structure prediction [2], image segmentation [3], etc. The primary hurdle in processing structures comes from the combinatorial sets of correlation between features. Structured output prediction methods based on vector representation [4] often employ advanced techniques from both probabilistic graphical models and large margin machines. However, due to the rigid nature of vectors, those methods do not fully deal with the rich set of relations between features. An example of probabilistic graphical model is Markov random field [5] that fixes the number of random variables. Moreover, learning edges of the probabilistic networks is in general a hard problem. In the large margin realm, struct-SVM [6] is an exemplar that converts structured input into vectors by an aggregation function, and later maps output vectors to structures by solving an optimization problem. Apart from conventional algorithms that treat a predictive distribution as a mixture model, topic

models [7-11] successfully learn multiple generative themes of input text documents. Such topic models are useful in organization, classification, and knowledge discovery of textual sources as well as other form of data analogous to documents [12-15]. However, a framework that can learn the latent theme of a collection of generic structures is not yet to be developed. Expanding topic models by introducing word dependency into documents [16, 17] is one direction. However, such approach only serves one-dimensional structures like sequences of symbols.

We want an algorithm that can learn the theme of multi-dimensional structures. The induced theme can then be used for knowledge discovery, prediction of new test instances, or synthesize novel data.

1.2. Objective

In this work, we would like to take a pioneer step to develop an inductive machine learning algorithm that accepts a set of multi-dimensional structures as input. The job of the algorithm is to infer a set of context-augmented substructures as the latent generative class. In the search for an appropriate solution, we considered several domains:

First, we attempt to relate our problem to data mining since extraction of the set of frequent sub-graphs (patterns) is a form of knowledge discovery [18]. However, this problem deviates from common techniques in data mining because of several reasons: (i) there is no universal support threshold; rather, each frequent sub-graph should have its own support; (ii) the supports are not fixed; they are learning parameters; and (iii) the goal of this problem is to find a model (a set of frequent

sub-graphs) that best explains the training data, which is a learning task.

Second, we relate our problem to graph summarization [19]. Some differences that deviate our problem from common techniques in graph summarization are: (i) graph summarization deals with individual graphs, while our problem involves simultaneous summarization and alignment of multiple graphs; (ii) graph summarization uses node attributes provided by a user to cluster nodes [20], while our solution should automatically clusters nodes using both node attribute and sub-structure topology; (iii) graph summarization generates frequent sub-structures (features) by an offline algorithm before performing feature indexing and matching-up [21], while doing so is detrimental to our problem because of the exponential number of substructures.

Third, we relate our problem to the structure learning task of probabilistic graphical models. Representation with a Markov network [22] comes with some challenges: (i) the number of random variables is unknown because an input graph cannot be converted to a fixed-length vector; (ii) partitioning of an input graph into subgraphs may not produce cliques; and (iii) frequent sub-graphs may have different topology and thus cannot be represented by template-based models [23]. Representation with a lifted graphical model [24] is more promising, since a frequent sub-graph could be captured by a par-RV (parameterized random variable), and sub-graphs partitioning could be captured by par-factors (parameterized factors) [25]. However, variability of the topology of frequent sub-graphs makes it challenging to define either the par-RVs or par-factors.

Fourth, we attempt to translate our problem to a vector-based representation. Aggregating feature vectors for a vector-based machine (e.g. struct-SVM [6]) by computing local and global properties of input graphs (e.g. degree distributions, diameter, centrality, etc) is straightforward, but defining operations on a feature vector that correspond to partitioning it into subgraphs is an open issue.

Finally, our problem seems similar to learning structure of a graph grammar [26, 27]. However, graph grammars originates from formal language theory of strings and are heavily oriented to fit in the Chomsky hierarchy. A limitation of context-free graph grammars is the production rules contain only one symbol in the left-hand-side, while we want the context substructure to be generic. Context-sensitive grammars are more powerful, but the complication of graph embedding (rewriting) operation does not fit specifically to our purpose.

By developing an algorithm that learns the structured latent theme, we hope to make several contributions: (i) introduce a novel problem of learning the latent theme of structured input, (ii) develop an inductive algorithm that finds the optimal model using structure learning methods of probabilistic networks, (iii) develop an effective prediction algorithm that classifies test input in a limited amount of time, and (iv) show case knowledge discovery using the learnt model.

1.3. Dissertation Structure

This dissertation is structured as followed: Chapter 1 introduces the current state of machine learning methods and the rationale for developing our approach; Chapter 2 recaps the history of machine learning with the structured input-output; Chapter 3

provides an overview of Evolving Transformation System (ETS), the primary structural representation formalism this work is based on; Chapter 4 elaborates the development of our inductive machine on a graph-based version of ETS; Chapter 5 showcases our algorithm on a competitive task of classifying human actions in static images; and finally the Conclusion section summarizes the work and gives some future direction. In addition to the main chapters, Appendix A and B contains details on response of objective functions, and benchmarking performance on the PASCAL VOC 2011's action classification task. At the end, Bibliography section holds references to relevant publications.

CHAPTER 2

MACHINE LEARNING WITH STRUCTURED INPUT-OUTPUT

This chapter provides an overview of the current methods dealing with learning structured evidence and gives structured prediction. This chapter is by no mean attempts to be comprehensive. Interested readers are referred to [4, 24, 28-30] and similar books and survey publications for a complete coverage.

2.1. Structured Prediction

Structured prediction concerns with predicting structured output [4] instead of single-valued or categorical output. The primary foundation of using vector-based representation on structures is via kernel construction [31-35]. The fundamental idea is transforming non-linear high-dimensional into a linear space by means of vector dot product. A kernel measures the similarity between two examples given a mapping of each example into a feature space. Positive definite kernels (with positive eigen values of the covariance matrix) are proven to factor into dot products. A Hilbert space corresponding to dot products of feature vectors is called a reproducing kernel Hilbert space (RKHS). New kernels can be constructed from existing kernels by five operation: linear

combination, product, tensor product, limit, higher-order function. Some popular kernels are: polynomial kernels, Gaussian kernels, and spline kernels. Representer theorem [36, 37] helps to express functions as a linear combination of kernel products. Some operations in RKHS include translation, centering, distance measure, subspace projection, and the well-known kernel principal component analysis (kPCA) [38] help map a non-dot-product algorithms into the RKHS. Some popular kernels for structured data are: set kernels, kernels for weighted automata, n-gram kernels, convolution kernels [39], tree kernels [40], graph kernels [41, 42], density kernels [43]. Using the kernel trick, one can map structured data into a dot product space and use existing technique to learn models. However, the prediction task requires mapping output to the original structural form independently of the learning problem. The latter mapping is called the *pre-image* or *decoding* problem [44].

Another scheme to deal with structured data is probabilistic graphical models (PGM) [23]. The advantage of PGM is its graph structure allows us to model qualitative dependency between random variables (RVs). A central concept in PGM is the conditional independency between RVs. A popular PGM for structured data is Markov networks [5] with factorization theorem [45, 46]. Likewise, Bayesian networks are directed acyclic PGMs with nice factorization property and probabilistic distribution tables (or functions) at vertices that allow easier learning [47]. In general, exact inference in PGMs is computationally expensive [48], but approximate inference often shows good convergence [22, 49]. Using PGMs, one can model the conditional probability of structured output, or define a joint feature map to combine input-output.

2.2. Statistical Relational Learning

This branch of machine learning not only care about statistical distribution of RVs, but also the structural relation between them [24]. The primary tool of SRL is PGM combined with first-order logic to describe relations. Multi-relational data could be thought of as several tables, some of them contains attributes, while others contains relationship between entities. There are two main trends in SRL: one use a logical language to define a probabilistic graphical model (lifted PGMs), and another imposes probability distribution on logical inference [24]. A common point for lifted PGMs are they can be represented by parameterized factor graphs [25], a generalization of factor graph [50]. For undirected PGMs, versions of lifted Markov networks were differentiated by which language was used to encode relation. Relational Markov networks (RMNs) [51] defines Markov networks through SQL. Markov logic networks (MLNs) [52] uses first-order logic to define par-factors. Probabilistic soft logic (PSL) uses logical atoms to represent par-RVs, and ground atoms to represents RVs. Imperative defined factor graphs [53] uses object-oriented programming to define par-RVs and par-factors as classes and RVs as instantiations. In the directed PGMs counterpart, lifted Bayesian networks are also differentiated by the choice of relational language. Bayesian logic programs (BLPs) [54], par-RVs are represented by logical atoms, and parent-child relation is defined by a definite clause. Probabilistic relational models (PRMs) [55] uses object-oriented programming in a relational database style to define both entities and relations by classes with reference slots. Bayesian LOGic (BLOG) [56] uses a typed relational language to define par-RVs as logical atoms, RVs as ground atoms, and par-

factors as dependency statements. The pros and cons of directed versus undirected lifted PGMs are similar to those of regular PGMs [23].

Inference in lifted PGMs aims at avoiding repeated computation [57, 58] and comprises of two strategies: top-down, and bottom-up [59]. Two popular methods for inference are first-order variable elimination [25, 60, 61], and lifted belief propagation [62, 63]. While lifted models exhibit symmetry that reduces inference burden, evidence in instantiated models may break up the symmetry and thus complicates inference. Knowledge based model construction [64] minimizes instantiation only enough to answer the query. MCMC [65] on regular PGMs [23] is also used in lifted PGMs for approximate inference [53, 66-68].

Parameter learning of lifted PGMs reuses methods of regular PGMs [23], both in generative manners [69-71] and discriminative manners [70, 72, 73]. Parameter learning in lifted PGMs is often difficult and techniques that find overlapping sub-networks by decomposition are employed [74]. Structure learning for lifted PGMs is also formulated as a heuristic search [75] similar to regular PGMs. Some popular methods for directed networks are [69, 76], and undirected networks are [77-80].

2.3. Graph-based Pattern Recognition

Employing graph representation in pattern recognition has been studied over 40 years [28, 30, 81]. The fundamental dilemma presented to the machine learning community when making the choice whether to use graph or vector for encoding patterns is summarized in Table 1. As we can see, the strength of one approach is the weakness of the other.

Table 1: The representational dilemma.

Graph Representation	Vector Representation
+ Pros: • Rich representational power. • Ability to encode structures. + Cons: • High computational complexity. • Few optimization algorithms.	+ Cons: • Simple, flat representation. • No structure. + Pros: • Simple, fast computation. • Lots of strong and sound optimization algorithms.

The first and foremost direction with graph-based pattern recognition is by exact and inexact graph matching. Three popular types of graph/subgraph exact matching are: isomorphism (bijective monomorphism), monomorphism (edge-induced), and homomorphism (edge-preserving). Exact matching algorithms are usually based on tree-search technique [82-84]. Inexact matching can be formulated as sub-optimal tree search [85, 86], continuous optimization [87, 88], spectral property (stable eigenvalues) [89, 90], or graph edit distance (GED) [74, 91] .

The next trend is graph embedding, i.e. mapping of the whole graph to a point in a vector space. In that way, graph similarity is formulated as distance between points. Some popular embedding techniques are: isometric embedding, spectral embedding, subpattern embedding, and prototype-base embedding. Isometric embedding attempts to fit a mapping function based on an existing similarity measure: self-organizing map [92], using neural networks [93], constant shift embedding [94]. Spectral embedding exploit

the property of eigenvalues and eigenvectors of the adjacency matrix that are invariant to vertex permutation [95, 96]. Subpattern embedding aims at detection of common substructures in the set of graphs [97, 98]. Finally, prototype-based embedding uses a set of prototype graphs as basis and measure the distances from an embedding graph to the prototype graphs [99, 100].

An important improvement for graph-based pattern recognition is graph kernels, i.e. similarity measure of two graphs that conform to properties of a dot product. Using kernels, problems with graph input can reuse all kernel methods in statistical machine learning. Some popular graph kernels are: marginalized kernels [42], kernels based on shortest paths [41], kernels based on graph edit distance [101], diffusion / convolution / random walk kernels [102], Laplacian / treelet kernels [103], graphlet kernels [104], Jesen-Shannon divergence kernels [105].

In unsupervised learning, clustering of graphs represent each object as a graph and group them based on some distance function. Some methods are: learning vector quantization [106], using function-described graphs [107], k-means clustering [108]. On the other hand, graph summarization concerns with grouping nodes and/or edges based on some criteria. Some methods for summarization are: cocoons [109], based on gaph-coloring [110], based on continuous optimization [111], using a k-nearest neighbor graph [105], using minimum spanning tree [112], using Erdos-Renyi model [113], using normalized cut [114], hypergraph clustering [115]. Maulik's pattern mining method [116] used evolutionary programming to compress graphs by repeatedly replacing common substructures by pointers.

In supervised learning, objects are represented as graphs, and class labels are also associated with objects. Self-organizing map couple with expectation maximization has been used to pull intra-class graphs closer while inter-class graphs apart [117]. Adaptive learning helped adjust the cost of graph edit distance to response to human expert feedback [118]. A centroid graph was also used to represent its class by GED [119]. Besides, other techniques such as embedding [120] and prototype [121] have been developed. On the other hand, semi-supervised graph summarization used various techniques to classify examples as vertices of a giant domain graph [46, 78, 122, 123].

2.4. Syntactic Pattern Recognition

In contrast to graph-based pattern recognition that learns a prototype graph having major properties of training data, syntactic pattern recognition learns a trivial start graph together with rewriting rules to represent a class. Employing a prototype graph together with graph edit operations is similar to employing a graph grammar, but different in where properties of the generating class are stored. Table 2 summarizes the similarity and difference between the two approaches.

Table 2: Similarity and difference between a graph grammar and employing a graph prototype.

Graph grammars	Graph prototype + A distance measure
Similarity	
• Have a start graph. • Apply production rules to the start graph to obtain a new graph.	• Have a prototype graph to start with. • Apply edit operations on the prototype graph to transform it to a new graph.
Difference	
• The start graph could be any subgraph that is common to all training examples. • Graph production rules have to capture almost all important properties of training examples.	• The prototype graph has to capture almost all important properties of training examples. • Graph edit operations are small, trivial graph deformation rules.

Syntactic pattern recognition has two approach: the first one encodes patterns by strings and reuses existing algorithms from formal language domain, and the second one encodes patterns by graphs and attempt to learn graph grammars similar to string grammars. Enhanced string grammars used in pattern recognitions are presented by Flasinski [29] including: indexed grammars, linear indexed grammars, head grammars, combinatory categorical grammars, conjunctive grammars, programmed grammars, dynamically programmed grammars.

This section focuses on graph grammars, i.e. generalization of string grammars to graph representation. For historical reason, graph grammars are heavily oriented by

formal language theory and the Chomsky's hierarchy [27]. There are two types of graph grammars that receive interest in the community. The first type is node replacement graph grammars [124], in which production rules contains a single non-terminal node on the left hand side, and the right hand side is a frequent subgraph. The second type is hyperedge replacement graph grammars [125], in which the left hand side of production rules contains a single hyperedge, and the right hand side is a frequent subgraph. Some pioneering methods were first introduced in 1970's to reconstruct the web using formal grammars [126-128]. Two primary problems with graph grammars are: parsing and learning.

The first attempt to cope with the graph grammar parsing problem is by Shi et al [129] that translates graphs with tree basis into equivalent strings for efficient processing. Later, two parsers for plex grammars were indipendently introduced by Bunke et al [130] and Peng [131] that have exponential time complexity in general. Predictive parsing for Relational Grammars (RG: a set of objects + a set of relations) was developed following Earley style [132] on subclasses of RGs that either have partial ordering or one-nonterminal restriction in production rules [133, 134]. $O(n^2)$ parsing complexity [135, 136] has been achieved for ETPL(k) subclasses of edNLC graph grammars [124] by virtual of node indexing assumption and restriction on left-derivation with at most *k* symbols. Researchers have designed graph grammars that work with restricted classes of graphs for efficient parsing. Chiang [137] elaborated on Lautemann's generalization of CKY algorithm for edge-replacement graph grammar [138] that achieves polynomial time parsing connected graphs with bounded degree. This achievement is due to tree decomposition of right-hand side of production rules. To cope with noisy input,

probability is introduced into the model. Skomorowski [139] introduced probabilistic distribution to nodes and edges of indexed edge-unambiguous graphs to enable parsing of distorted patterns.

While above methods deal with context-free graph grammars (i.e. the left-hand side of a production only contains a single non-terminal), context-sensitive graph grammars are more expressive and thus more expensive to parse. In context-sensitive graph grammars, left-hand side of production rules may contain as many nodes and edges as desired. Reserved graph grammars (RGGs) [140] were developed based on layered graph grammars [141] with additional node marking mechanism to easy graph embedding operation. Polynomial time parsing is also achievable on a subclass of RGGs that contains unambiguous grammars identified by selection-free production rules.

Learning with graph grammars is in general a hard problem, and one of the reasons is context-free graph grammars have no normal form [26]. Cook et al. [142] used minimum description length (MDL) principle [143] to build Subdue system that discovers subgraphs that give highest compression to a input graph. Jonyer et al [144] expanded the MDL principle to build SubdueGL that infer recursive and variable production rules for a node replacement context-free graph grammar. Despite its compactness, SubdueGL is limited to learning from individual graphs basis. Doshi et al [145] also expanded Subdue [142] to learn production rules of a *node replacement* context-free graph grammar from a set of input graphs. However, the method focuses on preliminary estimation of probability of production rules based solely on frequency of substructures, and has no special treatment to further reduce the description length like SubdueGL [144]. Oates et al. [146] continued the road to probability estimation of hyper-edge replacement

production rules by using Expectation Maximization (EM) [147] with a variation of the Inside-Outside algorithm [148]. However, the method is limited to graphs with logarithmic k-separability property [138] that restricts the number of subgraphs in a polynomial bound of the size of the parent graph. In [149],an application in visual programming of node replacement, context-free graph grammar based on SubdueGL [144] is presented by Ates et al. The method is based on the same idea of maximizing compression by identifying frequent substructures, but the compression is measured by graph sizes instead of description length.

CHAPTER 3

EVOLVING TRANSFORMATION SYSTEM

3.1. Literature Review

Evolving Transformation System (ETS) is a symbolic representation of structural processes pioneered by Prof. Lev Goldfarb and his research group in University of New Brunswick, Fredericton, Canada. The formalism complements traditional numeric and logic representation by emphasizing on the structure of objects instead of calculating numeric features. While symbolic representation is classical in Artificial Intelligence, ETS advances the field by allowing: (i) inductive learning, (ii) class-centric representation, and (iii) dynamic evolution of the structural transformation patterns.

Over 30 years, Goldfarb and his research team have diligently worked on a novel representation formalism to complement the ubiquitous numeric vector representation. He initially introduced a framework that unifies vector and syntactic representation in pattern recognition in [150], and studied the effect of distance metrics in vector space in [151]. In [152], he proposed ETS as a new model for pattern recognition that preserve geometric and syntactic patterns. The model leverages on a transformation system, which is generalization of a production system. An indirect application of ETS's primitive

concept was exploited in [153] to learn primitive features of synthetic images. A more elaborating analysis of distance metric was performed in [154], and a mathematical formulation of an Evolving Transformation System (ETS) was finally carried out in [155].

After that, Golfarb and his team continued pushing the inductive learning direction based on ETS with a series of publications.

In [156], the authors argued that numeric vector representation cannot capture inductive generalization of structured objects, due to the fact that vector has only one geometric topology, while ETS class representation allows for dynamic change in topology of generating structures. In [157], they showed that ETS can capture the compositional, symbolic structure of events that otherwise cannot be dynamically represented by the rigid form of numeric measures.

In [158], they posed the problem of measuring the distance between two objects using numeric vector operations, and proposed using a symbolic distance instead to reflect structural dissimilarity. A powerful inductive learning framework based on ETS that can dynamically update the set of structural transformation operations, in contrast to a fixed set of axiomatic operations imposed on numeric vector computation, is also further studied. In [159], Korkin et al showed an application of ETS formalism to construct an evolutionary genome graph (EG-graph) for each genome family. The author used a similarity measure based on genome transformations, and showed that EG-graph is close to a phylogenetic tree.

In [160], generalization of Peano axioms to a structural representation that can capture change in the structure of objects is investigated. Each numeric quantity (e.g. 1, 2, 3…) is represented by an atomic element called numeric primitive, and numeric primitives could be concatenated in a parallel fashion to represent arithmetic operations over time. A concept about class representation was also introduced, which emphasized that a finite representation of a class might embrace an infinite generative power. The author also assumed the existence of a *structural measurement* process that extracts features into a structural form, rather than a numeric vector. Inadequacy of vector, logic, and other discrete representations such as strings, trees, and graphs were also briefly discussed.

In [161], a summary of arguments are compiled. First, the author restated that numeric vector representation (i.e. points in a hyper-space) is not enough to capture the full structure of real world objects, because of the fact that vector representation is unstructured by nature. Second, point representation induces metric distance (i.e. Euclidean distance, …) between objects, while it is still unproved whether human brain forms similarity concept based on exact measure of numeric features, or structural configuration of objects, or both. Third, an interesting observation is human tends to group things into categories, suggesting that objects and classes have close relationship and should be learned at the same time. Such observation is also exploited in numeric statistical learning where clusters and distributions group related points under the same roof. And fourth, the author emphasized the concept of "(temporal) structural process" by which an object is viewed as a dynamic structure that evolves over time under a set of structural transformation rules.

3.2. Elements of ETS

3.2.1. Structure Measurement Device

Similar to the ubiquitous metric system, ETS formalism assumes there is a standard measurement device that converts a real world object to a structural representation.

Definition: A structure measurement device $\mathcal{SMD}$ is a function $\mathcal{SM}$: $\mathbb{W} \mapsto \mathbb{S}$, where:

- $\mathbb{W}$ is the real world space
- $\mathbb{S}$ is a structural space - a space of structures

Thus, given $\mathcal{o} \in \mathbb{W}$ is a real world object, a $\mathcal{SM}(\mathcal{o}) = \sigma$ is a mapping to a composite structure $\sigma \in \mathbb{S}$.

This loose definition of a structure measurement device allows for flexibility in designing such a conversion function. Such flexibility is also prevalent in machine learning community, where a numeric feature could be extracted from any combination of properties of an object, and there is no restriction on the length of the feature vector as well as which specific dimension to house a specific feature.

3.2.2. Primitives

A primitive is an atomic building block of a structure from the perspective of a specific representational level. Primitives can have different *types*, corresponding to different features of the object. Primitives are equipped with *connectors* to allow them to connect to each other. A connection represents an inter-relationship between a pair of features. Connectors are further divided into *initials* and *terminals*. An initial represents

the willingness of the primitive to accept a connection, whereas a terminal represents the intention of the primitive to connect to others. This division could be viewed as analogous to natural classification of males and females. Figure 1 provides an illustration of primitives.

Definition: A primitive is a tuple $\pi = (\ell, \mathbb{IS}, \mathbb{TS})$ where:

- ℓ is a string label
- $\mathbb{IS} = \{ \mathit{is}_1, \mathit{is}_2, \ldots, \mathit{is}_n \mid \mathit{is} \in \mathcal{N} \}$ is a set of initial sites
- $\mathbb{TS} = \{ {}_1, \mathit{ts}_2, \ldots, \mathit{ts}_m \mid \mathit{ts} \in \mathcal{N} \}$ is a set of terminal sites.

Definition: A primitive π_1 can connect to a primitive $\pi_2 \Leftrightarrow \exists\ \mathit{ts}_m \in \pi_1, \mathit{is}_n \in \pi_2$ such that $\mathit{ts}_m = \mathit{is}_n$.

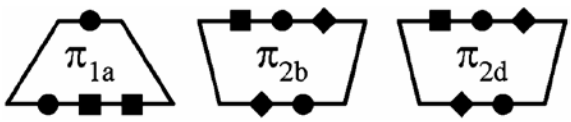


Figure 1: Illustration of three primitives, adapted from [161]. A primitive is an atomic (indivisible) structure that constitute objects. Each primitive has a label to differentiate its type. Primitive types have different shapes to assist human visualization. A primitive has connecters on top (called initials) and under its bottom (called terminals). Connecters also have labels to differentiate between connector types. Different connector types have different shapes to assist human visualization. In a structural process, only terminals of a primitive can connect to initials of another primitive, provided that terminals and initials are of the same connector type.

3.2.3. Structs and Composites

A struct represents a structure from the real world. A struct is created by connecting primitives through their corresponding initial and terminal sites. Unlike a graph where an edge is added based on a global rule, a struct delegates the connection decision to its local primtives. Thus, in a struct, primitives plays an active role in selecting which other primitives to establish relationship with. Moreover, the order of primitive connection maintains an implicit temporal dimension for the struct. Figure 2 provides an illustration of a struct.

Definition: A struct is a tuple $\sigma = (\Pi, \mathbb{CS}, \mathfrak{T})$ where:

- $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is a countable set of primitives constituting the struct.
- $\mathbb{CS} = \{(\ell(\pi_k), \ell(\pi_m), \mathcal{s}) \mid \mathcal{s} \in \mathbb{TS}(\pi_k), \mathcal{s} \in \mathbb{IS}(\pi_m), 1 \leq k, m \leq n\}$ is a set of pairs of connecting primitives identified by their labels and the connecting site.
- $: \Pi \mapsto \mathcal{N}$ is a time stamp function associating each primitive to a time when it joins the struct.

A powerful property of structs is the ability to combine shared portions to create a new, bigger struct. This ability marks an important capstone for lifting from a low level of representation to a higher level of representation, and the existence of transformations. Figure 3 illustrates composition of three structs σ_1, σ_2, and σ_3 into a big struct.

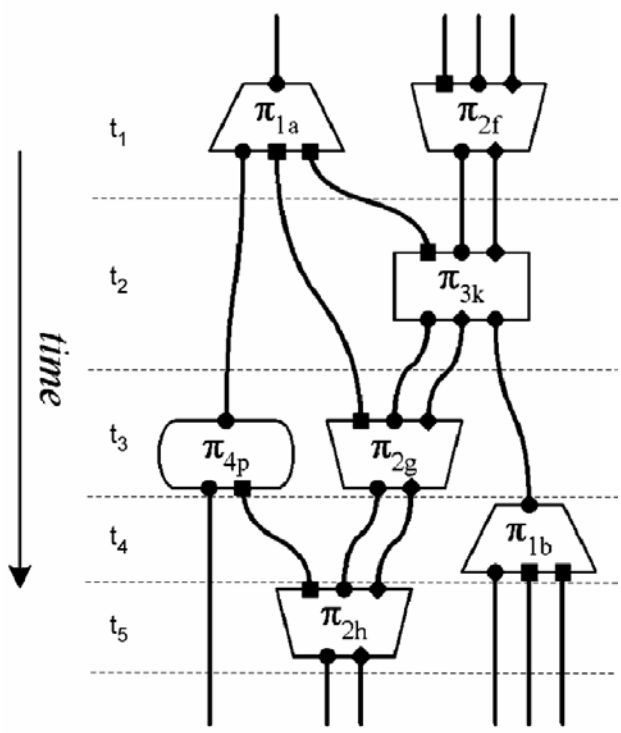


Figure 2: Illustration of a struct, adapted from [161]. In this structure formation process, only terminals and initials of the same connecter type can connect. Time slices are separated by dashed lines. Primitives connected formed within the same time slice have the same temporal order. Connections can be formed across discontiguous time slices.

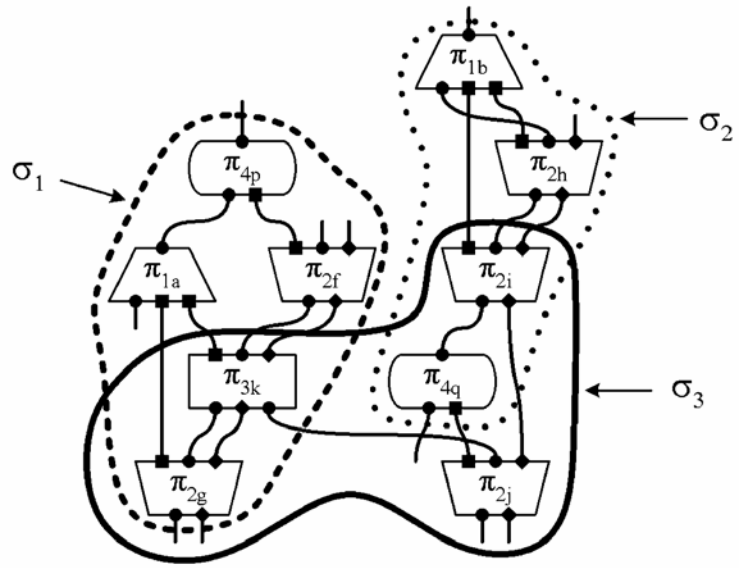


Figure 3: Illustration of a struct composition, adapted from [161]. Three structs σ_1, σ_2, and σ_3 can be combine using common/overlapping sub-structures. Struct σ_1 and σ_2 shares two primitives π_{3k} and π_{2g}, while struct σ_2 and σ_3 shares two other primitives π_{2i} and π_{4q}.

An important aspect of struct composition is the weak ordering of primitive timestamps. Usually, a struct is designated as the prior struct, which contains weakly ealier (smaller) primitive timestamps than the other struct. Which of the two struct to be designated as the prior struct is application dependent.

Definition: Composition of two structs $\sigma_1 = (\Pi_1, \mathbb{CS}_1, \mathfrak{T}_1)$ and $\sigma_2 = (\Pi_2, \mathbb{CS}_2, \mathfrak{T}_2)$, in which σ_1 is the prior struct, is a struct $\sigma = (\Pi, \mathbb{CS}, \mathfrak{T})$ where:

- $\Pi = \Pi_1 \cup \Pi_2$
- $\mathbb{CS} = \mathbb{CS}_1 \cup \mathbb{CS}_2$
- $\mathfrak{T}(\pi) = \begin{cases} \mathfrak{T}_1(\pi)\ if\ \pi \in \Pi_1 \\ t^1_{max} + \mathfrak{T}_2(\pi)\ if\ \pi \in \Pi_2 - \Pi_1 \end{cases}$, $t^1_{max} = \max\limits_{\pi_k \in \Pi_1} (\mathfrak{T}_1(\pi_k))$

In real-world applications, it is not always feasible to obtain temporal information of the structure of an object. Not without saying that the object structure itself is not always fully available. In such cases, a representational form of object structure without temporal information is called a *composite*. Like structs, two smaller composites could combines to form a larger composite. Note that since composites do not contain temporal information, combination of two composites does not require an explicit temporal order like a prior struct does.

Definition: A composite is a tuple $\mathcal{x} = (\Pi, \mathbb{CS})$ where:

- $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is a countable set of primitives constituting the structure of the composite.
- $\mathbb{CS} = \{(\ \ell(\pi_k), \ell(\pi_m), \mathcal{s}) \mid \mathcal{s} \in \mathbb{TS}(\pi_k), \mathcal{s} \in \mathbb{IS}(\pi_m), 1 \leq k, m \leq n\}$ is a set of pairs of connecting primitives identified by their labels and the connecting site.

Definition: Composition of two composites $x_1 = (\Pi_1, \mathbb{CS}_1)$ and $x_2 = (\Pi_2, \mathbb{CS}_2)$, is a composite $x = (\Pi, \mathbb{CS})$ where:

- $\Pi = \Pi_1 \cup \Pi_2$
- $\mathbb{CS} = \mathbb{CS}_1 \cup \mathbb{CS}_2$

3.2.4. Transformations

Transformation emerges as a tool to accommodate multiple level representation of a structural process. As Figure 4 shows a sub-struct at a lower level of representation is compressed into a single primitive at a higher level of representation. An important note is the smaller sub-struct above the one bounded by a dashed rectangle becomes a connector of the higher-level primitive. It implies that a connector could be a sub-struct of a lower-level representation.

Figure 5 shows a detailed visualization of a transformation at a lower level (left image) and at a higher level (right image). As a recursive embodiment of a connector, terminal site of a prior transformation is initial sites of a successor transformation. Thus, a transformation does not necessarily hold both initial sites and terminal sites. ETS names initial sites of a transformation its "context", and the actual attaching additional structure its "tail". In this sense, tail of the prior transformation should encapsulate context of the successor transformation. The entire (context + tail) of a transformation is called its "body". It is important to point out that a transformation is viewed as a single primitive at a high-level representation, thus it does not contain temporal information.

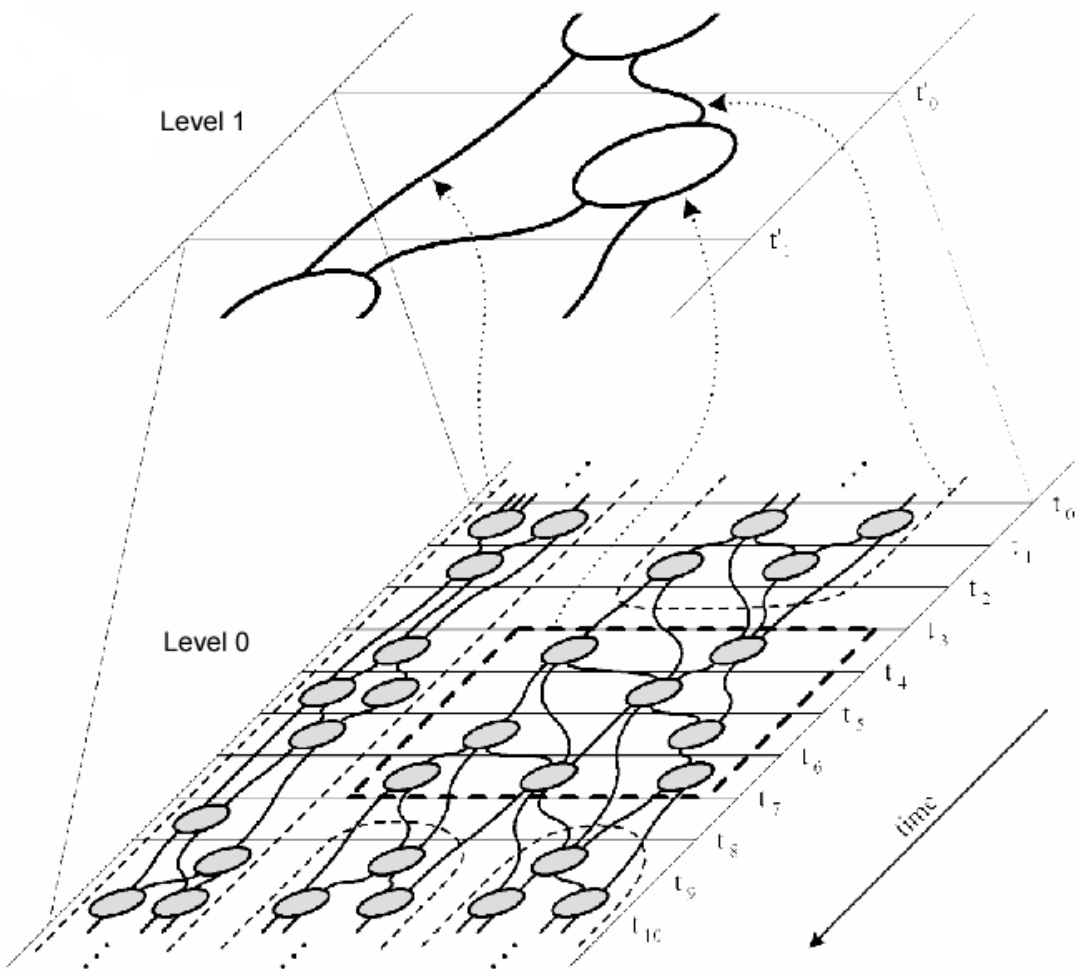


Figure 4: Illustration of multiple representational levels of a structural process in 3D, adapted from [162]. Level 0 is a low-level representation, while level 1 is a higher-level representation. A sub-struct at level 0 (bounded by thick, dashed rectangle) is represented as a single primitive (a big, solid oval) at level 1.

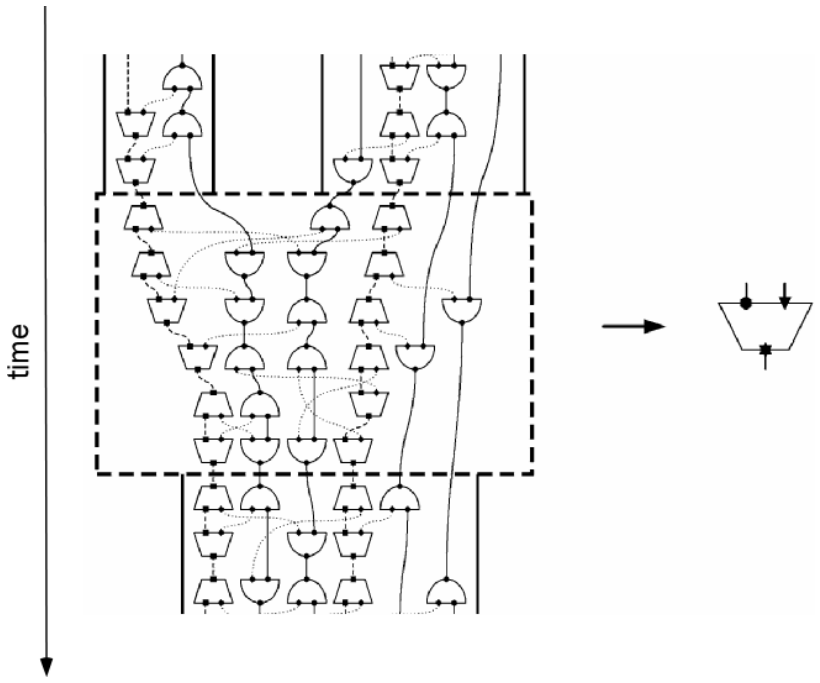


Figure 5: Illustration of a transformation in 2D, adapted from [161]. Left image shows low-level processes: The transformation (heavy dashed rectangle) originates from two structural processes (context), unites to a transformation process, then leads to the next process / next transformation. Right image shows the transformation in a high-level process, in which the transformation is represented as a high-level primitive.

Definition: A transformation is a tuple $\tau = (x_{\text{context}}, x_{\text{tail}})$, where:

- x_{context} is a composite representing the transformation's context
- x_{tail} is a composite representing the transformation's tail
- $\exists \{(\pi_1, \pi_2)\} : \pi_1 \in x_{\text{context}}, \pi_2 \in x_{\text{tail}}, \pi_1$ is connected to π_2

Reminding ourselves that two or more composites could be combined to form a larger composite. A transformation is essentially a pair of composites, one of which (the context) holds the information of where to combine, and the other (the tail) is an additional structure as the result of combination. We should expect that a transformation could "attach" to a composite.

Definition: Application of a transformation $\tau = (x_{\text{context}} = (\Pi_1, \mathbb{CS}_1), x_{\text{tail}} = (\Pi_2, \mathbb{CS}_2))$ to a composite $x = (\Pi, \mathbb{CS})$, is a composite $x' = (\Pi', \mathbb{CS}')$ where:

- $\Pi_1 \subset \Pi \land \mathbb{CS}_1 \subset \mathbb{CS}$
- $\Pi' = \Pi \cup \Pi_2 \land \mathbb{CS}' = \mathbb{CS} \land \mathbb{CS}_2$

We denote $x' = x \blacktriangleleft \tau$.

Figure 6 illustrates application of a transformation to a composite. The figure provides a good example of the property that a transformation could attach to more than one contextual location of the targeting composite, and the attachment could happen multiple times.

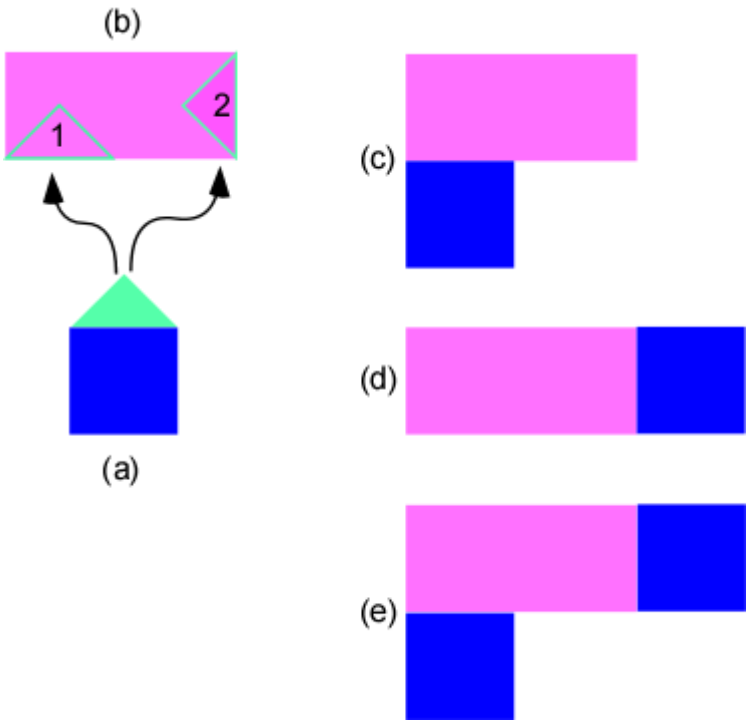


Figure 6: Application of a transformation. (a) is a transformation with the green triangle as its context and the blue square as its tail. (b) is a composite containing two substructures (1) and (2) that match the context of (a). (c) is a composite resulting from applying (a) to (b) at contextual location (1). (d) is a composite resulting from applying (a) to (b) at contextual location (2). (e) is a composite resulting from applying (a) to (b) at both contextual locations (1) and (2).

3.2.5. Class-centric Modeling

In [160], the author introduced the concept of a class representation. ETS's class representation model is motivated by a basic principle in evolutionary biology that each class (clade) of objects (species) has an ultimate common ancestor called a *progenitor* [159, 161, 163]. This notation is similar to the start symbol in Context-Free Grammars (CFGs) [164]. Secondly, ETS hypothesizes that there is a common set of transformations, by which all objects of the same class are derivable from the progenitor. This set of transformations is analogous to the set of production rules in CFGs. A list of concept analogy between ETS and CFGs is presented in Table 3.

The central difference is while CFGs operate on textual sentences, ETS operates on general structures with an arbitrary number of dimensions. This important difference makes ETS exponential in representation complexity and leads to other detail differences in both model representation and properties. A list of detail differences between ETS and CFGs is presented in Table 4.

Table 3: Concept correspondence between ETS and CFGs.

Evolving Transformation System	**Context-Free Grammars**
+ Progenitor	+ Start symbol
+ Transformation	+ Production rule
+ Context	+ Left-hand side
+ Tail	+ Right-hand side
+ Formative history	+ Parse tree

Table 4: Differences between ETS and CFGs.

Evolving Transformation System	Context-Free Grammars
+ Primitives are uniform in their roles. + There could be as many primitives in the transformation context as needed. + ETS naturally satisfies the no-cycle constraint, because a struct is always expanded when a transformation is applied. + There is no boundary in application of transformations; however, degree of membership of the resulting object reduces as the number of applied transformations saturate.	+ Characters / variables are classified into terminals and non-terminals. + Left-hand side of a production rule is strictly a non-terminal. + A constraint on a proper CFG is that it has no cycle in any of its chain of production rule application. + Application of production rules is bounded by terminal variables.

Definition: A class representation (model) is a tuple $\mathcal{M} = (\rho, \mathcal{T})$ where:

- ρ is a composite representing a progenitor
- $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a set of transformations

Figure 7 illustrates the concept of a class representation. The model consists of a potato (a) as the progenitor, and a set of body parts (b) as transformations. By attaching the parts to the potato, one could construct a correct Mr. Potato Head in (c), or a strange character in (d). This illustrates the effect of multiple ways to apply a transformation to a composite. By consistently attaching parts to the potato, one could obtain the class represented by the model - in this example is a potato head family.

Given a class representation, we now have the ability to form any member struct/composite of a class by successively applying transformations in the provided transformation set, starting from the progenitor. ETS presents the generating process by a temporal sequence of transformation applications called a *constructive (or formative) history*. An interesting by-product of constructive histories is, at a high-level view point, they are structs. Figure 8 gives an illustration of a constructive history, and figure 9 provides an overview of the ETS ecosystem.

Definition: A constructive / formative history (or simply a history) $\mathcal{h}$ of a composite x under a class representation $\mathcal{M} = (\rho, \mathcal{T})$ is a *temporal sequence* of transformation applications $(x) = (\tau_1, \tau_2,.., \tau_k \mid \tau_i \in \mathcal{T}\ \forall\ i = 1\ldots k)$ such that:

$$x = \rho \blacktriangleleft \tau_1 \blacktriangleleft \tau_2 \blacktriangleleft \ldots \blacktriangleleft \tau_k$$

Note that a transformation may repeat several times in the application sequence, and the order of transformations in the sequence implies a temporal order.

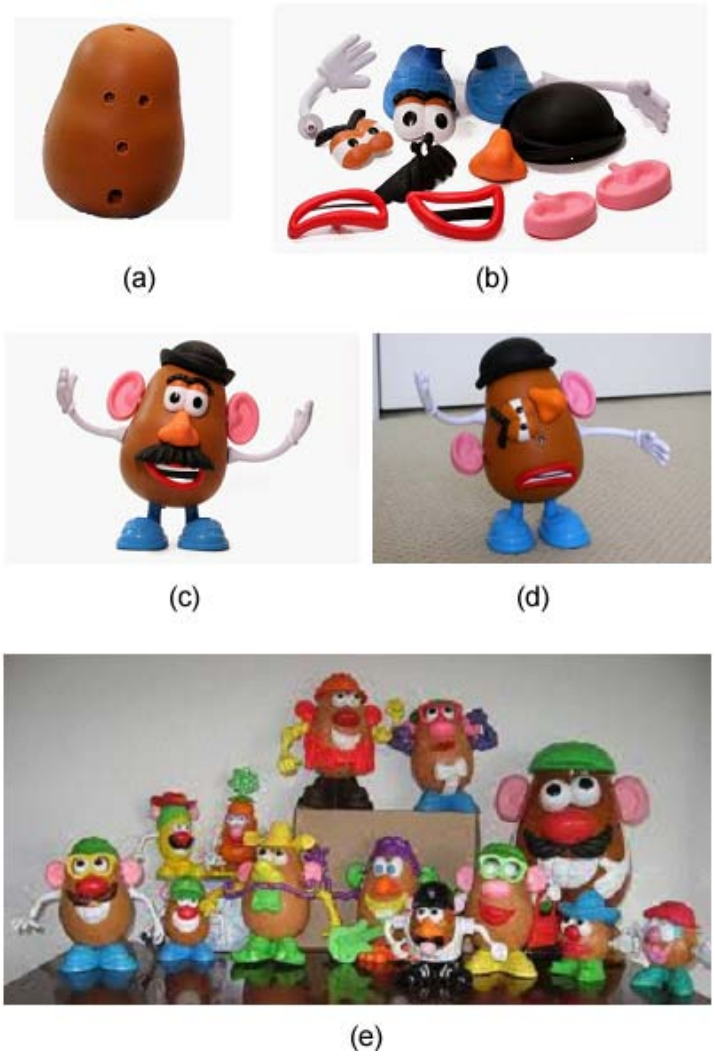


Figure 7: A potato head family model. (a) is a progenitor. (b) is a set of transformations. (c) is a correctly constructed Mr. Potato Head. (d) is a incorrectly constructed Mr. Potato Head. (e) is a class constructed from the model comprised of (a) and (b).

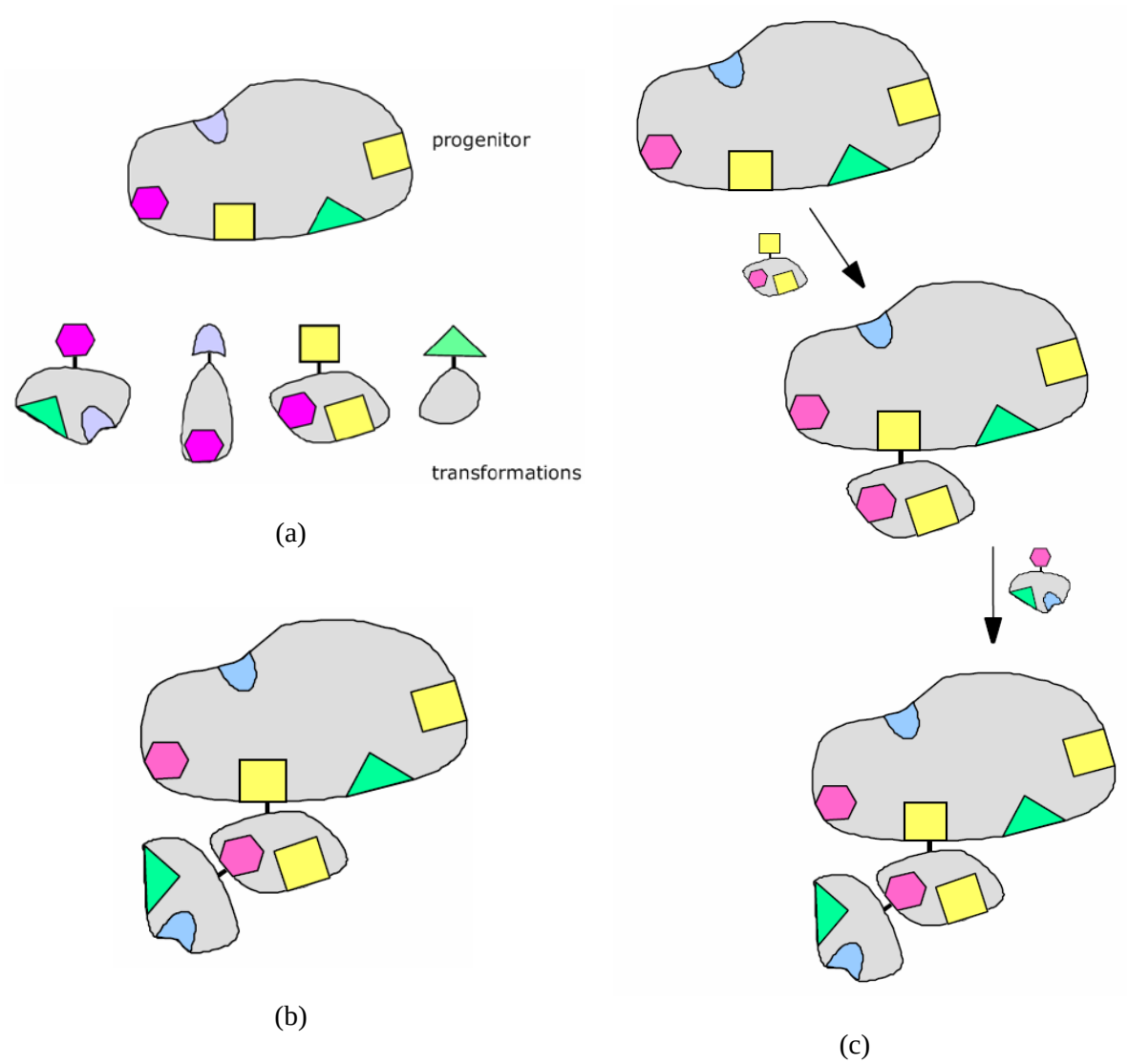


Figure 8: Illustration of a class representation and a constructive history, adapted from [165]. (a) A class representation comprising of a progenitor and a set of four transformations. Notice that the progenitor and tails of transformations contains substructures that match contexts of transformations. (b) A composite. (c) A constructive history recording the sequence of transformation applications that produces the composite in (b) from the class representation in (a).

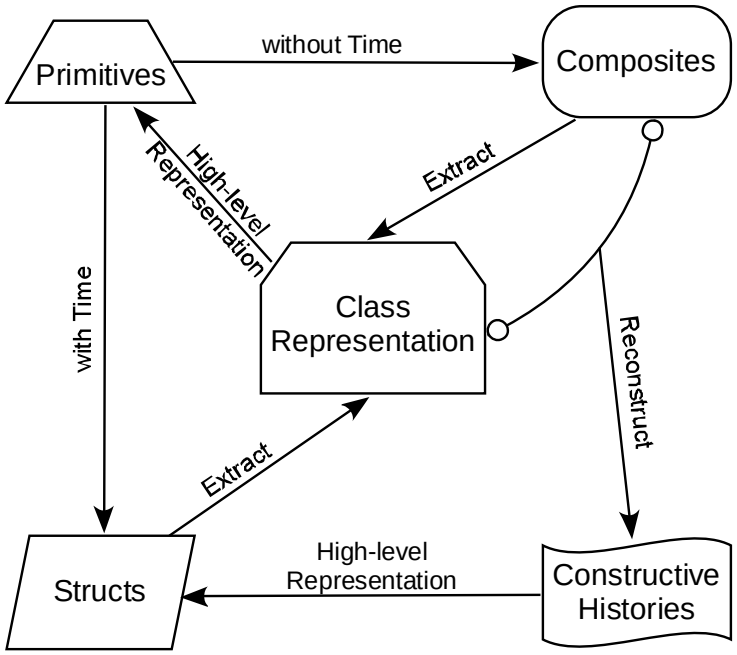


Figure 9: The ETS ecosystem. Composition of primitives with temporal information creates structs, and without temporal information creates composites. From composites and/or structs, one could extract a progenitor and a set of transformations, i.e. a class representation model. With a class representation, constructive histories of a composite could be parsed. Finally, under a high-level view point, a constructive history is a struct, and transformations of a class representation model are primitives.

CHAPTER 4

GRAPH-BASED EVOLVING TRANSFORMATION SYSTEM MACHINE

4.1. Introduction

While ETS is an elegant and powerful symbolic representational formalism, no algorithm for inductive learning of its transformation system exists to date. The problem is the ETS representational scheme is too general and a proper interpretation is necessary to convert it to a concrete computer program. In this work, we employ graph theory to encode entities of the ETS ecosystem. To our understanding, there is no formal work to date that compares the representational power as well as applicable operations between ETS formalism and graph theory.

Since ETS structural representation could be embodied in form of graphs, we devise a novel learning paradigm based on the mixture of ETS formalism, graph-based pattern recognition, and statistical learning. The new machine learning method is called a Graphical Evolving Transformation System Machine (gETSM). The method aims at learning the latent set of graphical transformations using an optimization technique from

statistical learning. We explore both the ubiquitous probability measure together with a psychology-based typicality measure.

4.2. Feature Graphs

In graph-based pattern recognition, Relational Graphs (RGs) [166] are used to encode syntactic feature of the object of interest. Nodes in a RG represent syntactic primitives (e.g. the roof, the walls, the floor of a house), and edges represent relation between the primitives (e.g. the roof is above the walls, and the walls are above the floor). The problem with RGs is node and edge labels are usually constituted of symbols only, so that semantic relation (e.g. how far is the roof from the walls) is not captured.

To alleviate this weakness, Tsai and Fu introduced Attributed Relational Graphs (ARGs) [167, 168] that embeds semantic feature into node and edge labels in the form of continuous numeric vectors. The addition brings the encoded graphs to a hybrid state that combines both symbolic and numeric representation. In this work, we intent to use a purely symbolic representation that also has the ability to carry semantic features.

Attempts to encode continuous information in the form of syntactic primitives have been carried out in the literature [169] by mean of thresholds and quantization (e.g. dividing 360^o into 6 bins of 60^o each and assign real-value angles to those bins). However, our method follows the encoding schema promoted by ETS formalism [161], which uses discretization instead of quantization. Under this schema, first a primitive unit is chosen (e.g. a 10^o angle), then a continuous feature is represented as a connected series of primitive units. (e.g. a 30^o angle is represented as a series of three 10^o angles). Figure 10 shows representation of a unit measure and encoding of three angles 10^o, 20^o, and 30^o.

This discretized representation enable three operations on semantic features that quantized representation does not offer: (i) quantity comparison: e.g. a 20^o angle is smaller than a 30^o angle because the former has less primitives; (ii) addition: e.g. a 30^o angle is composed of a 10^o angle (one primitive) and a 20^o angle (two primitives); and (iii) subtraction: e.g. by taking out a primitive (10^o angle) from a 30^o angle, we obtain a 20^o angle (two primitives).

In a sense, our encoding scheme could be thought of as an expansion of ARGs such that each semantic feature is expanded into a series of symbolic primitives. Lastly, our encoding scheme is far different from the ones employed by Function-Described Graphs [86] or Relaxation Labeling [170].

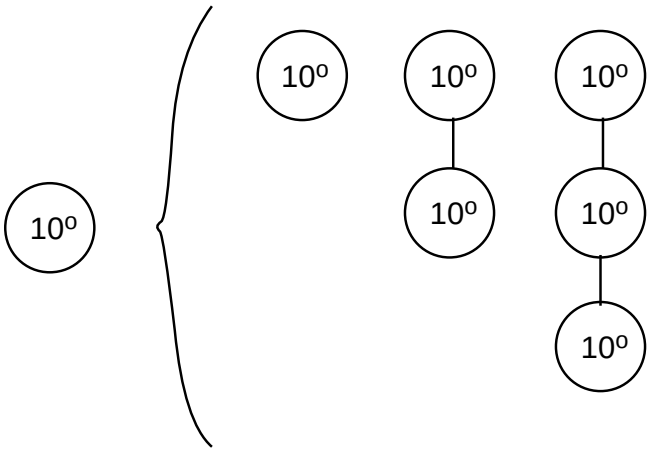


Figure 10: Symbolic expansion of semantic features. (a) A primitive representing a unit 10^o angle. (b) Three angles 10^o, 20^o, and 30^o represented by concatenating the primitive angles.

Definition: A *feature graph* is a labeled, undirected graph $G = (V = V_S \cup V_M, E, \mathfrak{L})$ where:

- V_S is a set of syntactic vertices, i.e. primitives represents structural information.
- V_M is a set of semantic vertices, i.e. primitives represents unit continuous measures.
- $V = V_S \cup V_M$ is the total set of vertices of this graph.
- E is a set of undirected edges connecting pairs of vertices in V.
- $\mathfrak{L}$ is a labeling function such that $\begin{cases} \mathfrak{L}(e) = \emptyset \; \forall \; e \in E \\ \mathfrak{L}(v) = \ell_v \; \forall \; v \in V \end{cases}$, where $\ell_v \in \mathcal{L}$ is a finite set of symbols.

In plain language, a feature graph is an undirected graph with symbols at vertices and edges having no label. This simple choice of representation helps narrow down the search space for graph matching, and is still sufficient to encode entities of the ETS ecosystem. Some implications of this representation schema are:

- Traditionally, edges are labeled to encode relations between feature vertices. In a feature graph, such relations are considered another type of features, and thus edge labels are converted to vertices, leaving the edge no labeling information.
- Directional edges are special cases of undirected edges. In either case, the learning algorithm is still the same, except for more constraint in graph matching if the feature graphs have directional edges.

4.3. Graphical Representation of ETS Formalism

Using feature graphs, we map entities of the ETS formalism to a graph space, thus effectively merge ETS to the main stream of graph-based pattern recognition. Specifically, by representing a composite with a feature graph, other ETS entities such as transformations, class representations and formative histories follow recursively. We call the graphical representation of ETS a Graphical Evolving Transformation System (gETS).

4.3.1. Fundamental Entities

Definition: A *primitive feature* (denoted π) is a symbolic label, and different types of primitive features have different labels.

Definition: A *composite graph* (denoted x) is a feature graph that has primitive features as its vertices.

Commonly, graph-based pattern recognition concerns with deformation of graphs so that to allow inexact matching and prototype construction. Cordella's transformation model [171, 172] and graph edit distance [167, 168, 173] are some good examples of graph transformations. In this aspect, gETS's transformations are graph deformation rules too. However, instead of having five categories of transformations (i.e. addition of vertices, removal of vertices, addition of edges, removal of edges, and label substitution), gETS only accommodate one category of transformations: simultaneous addition of vertices and edges. This simplicity is attributed to the format of feature graphs and the underlying ETS foundation. Having only one category of transformations while still

maintaining the deformation power is an advantage of gETS over other transformation models. Lastly, a transformation could be thought of as summarization (or compression) of a subgraph, and thus is similar to a unification of a tail intranode graph and a context positive superedge graph [174].

Definition: A *graph transformation* is a tuple $\tau = (\mathcal{x}_{context}, \mathcal{x}_{tail}, E_{connect})$, where:

- $\mathcal{x}_{context}$ is a feature graph representing the transformation's context
- $\mathcal{x}_{tail}$ is a feature graph representing the transformation's tail
- $E_{connect}$ is a set of undirected edges connecting the context to the tail:

  $E_{connect} = \{e = (\pi_1, \pi_2)\}$ where $\pi_1 \in \mathcal{x}_{tail}$, $\pi_2 \in \mathcal{x}_{context}$, source(e) = π_1, and target(e) = π_2.

Definition: *Applicability of a transformation*. A transformation $\tau = (\mathcal{x}_{context} = (V_1, E_1, \mathfrak{Q}_1), \mathcal{x}_{tail} = (V_2, E_2, \mathfrak{Q}_2))$ is *applicable* to a composite graph $\mathcal{x} = (V, E, \mathfrak{Q})$ if and only if:

- The context subgraph is a subgraph of the composite graph: $G_{context} = (V_1, E_1) \subseteq G = (V, E)$.

Definition: *Application of a transformation* $\tau = (\mathcal{x}_{context} = (V_1, E_1, \mathfrak{Q}_1), \mathcal{x}_{tail} = (V_2, E_2, \mathfrak{Q}_2))$ which is applicable to a composite graph $\mathcal{x} = (V, E, \mathfrak{Q})$, is a composite graph $\mathcal{x}' = (V', E', \mathfrak{Q}')$ where:

- $V' = V \cup V_2$
- $E' = E \cup E_2$
- $\mathfrak{Q}' = \mathfrak{Q} \cup \mathfrak{Q}_2$

We denote $x' = x \blacktriangleleft \tau$.

Figure 11 provides graphical representation of primitive features, composites, transformations, and transformation application. We notice that a transformation can be applied multiple times, at multiple locations of a composite graph.

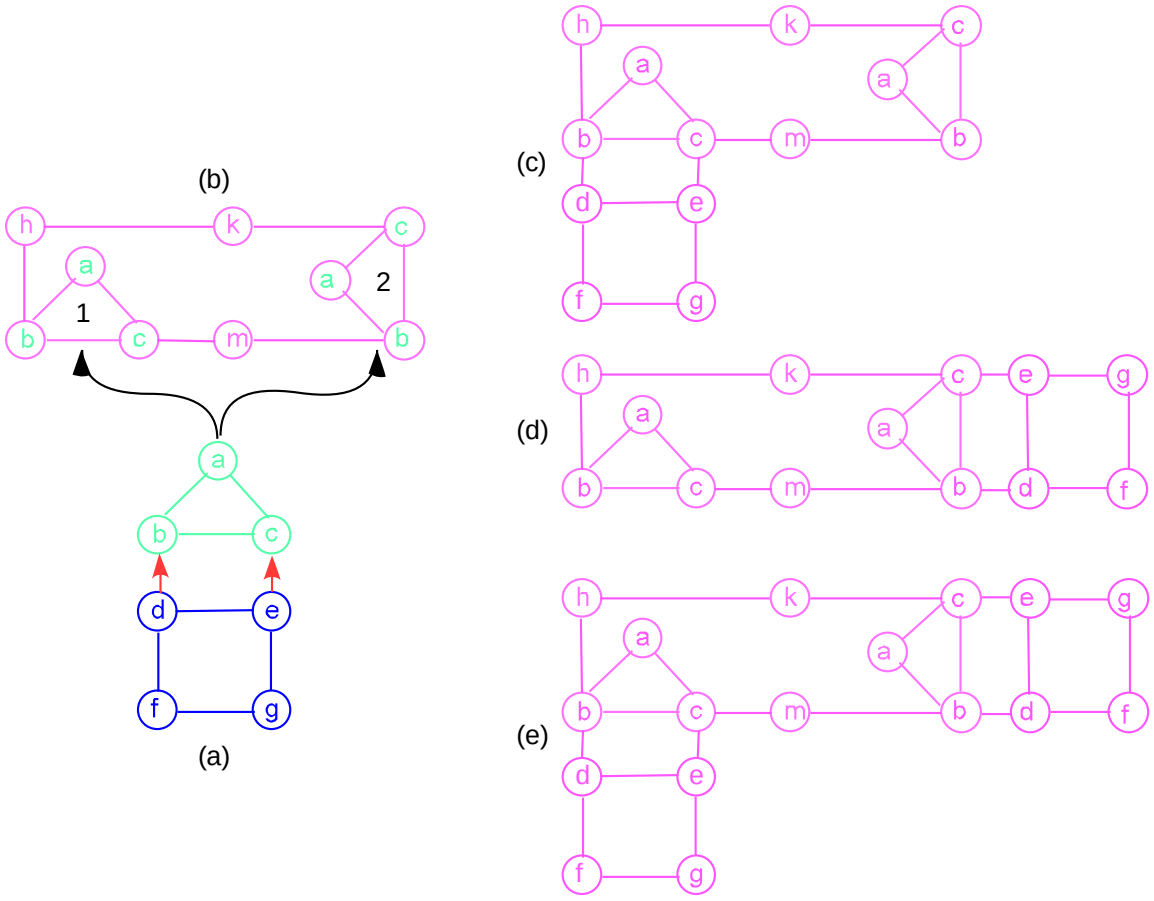


Figure 11: Graphical representation of fundamental gETS elements. Each primitive feature is represented as a labeled vertex. (a) A transformation τ comprises of a green triangle subgraph as its context, a blue square subgraph as its tail, and a set of red edges with *pseudo direction* pointing from vertices in the tail subgraph to vertices in the context sugraph. In this illustration, connecting edges are $\overrightarrow{(\boldsymbol{d},\boldsymbol{b})}$ and $\overrightarrow{(\boldsymbol{e},\boldsymbol{c})}$. (b) A composite graph x encoding a pink rectangle which contains two triangle (a-b-c)'s at locations 1 and 2 that match the context of the transformation (a). (c) A composite graph as the result of applying τ to x at location 1. (d) A composite graph as the result of applying τ to x at location 2. (e) A composite graph as the result of applying τ to x at both locations 1 and 2.

4.3.2. Class Representation / Model

The closest model in graph-based pattern recognition that has a start graph, and a set of graphical transformation rules are graph grammars [27]. gETS seems to borrow the idea of context-free production rules (transformations) from Context-Free Graph Grammar [26], but it does not rely on the distinction between terminal and non-terminal nodes to bound the derivation. gETS is also different from programmed graph grammars [175] because it does not rely on a global control diagram. Rather, context of transformations collectively regulates derivation. On the other hand, gETS's transformations simplify the embedding problem [27] by only allowing adjunction of subgraphs to an existing state, i.e. none of existing nodes or edges is removed. A formalism close to gETS is tree adjoining grammar (TAG) of Joshi et al [176, 177]. A transformation application operation of gETS is similar to TAG's tree substitution operation. However, TAG is restricted to tree representation and productions' left-hand side is constituted of a single non-terminal, which are not the case for gETS. That said, gETS identifies itself as a novel representation in graph-based pattern recognition.

Definition: A *class representation* (or a *model*) is a system $\mathcal{M} = (\rho, \mathcal{T})$ where:

- ρ is a feature graph representing the progenitor.
- $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a set of graph transformations.

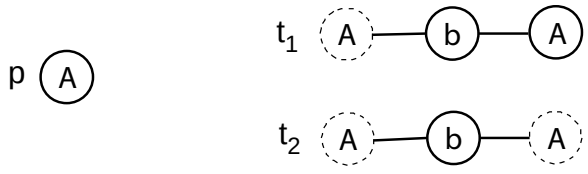
Since there is no mechanism in the model that constraints a unique derivation for an input graph, gETS class representation is *ambiguous*. A derivation of a graph by the model is similar to a supernode graph [174] - a summarization (or compression) of the original graph by means of graph transformations.

Definition: A *transformative history* (or simply a *history*) $\mathcal{h}$ of a composite graph $\mathcal{x}$ under a model $\mathcal{M} = (\rho, \mathcal{T})$ is a *directed acyclic graph* with an implicit temporal function $\mathcal{h}(\mathcal{x}) = (V, E, \mathfrak{T})$ where:

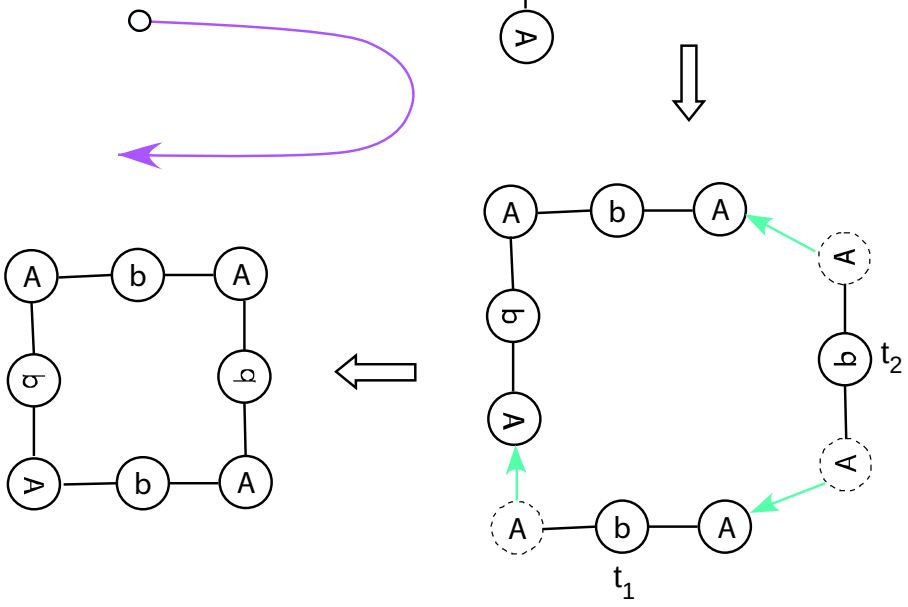
- $V = \{\tau_1, \tau_2, \ldots, \tau_k \mid \tau_i \in \mathcal{T} \ \forall \ i = 1 \ldots k\}$ is a set of vertices where each vertex represents a graph transformation.
- $E = \{(\tau_i, \tau_j)\}$ is a set of directed edges from τ_i to τ_j such that $(\tau_i) > (\tau_j)$.
- $: \mathcal{T} \mapsto \mathcal{N}$ is a time stamp function.

Figure 12 illustrates a model of a class of equilateral polygons, the process to construct a square from the model, and the resulting transformative history.

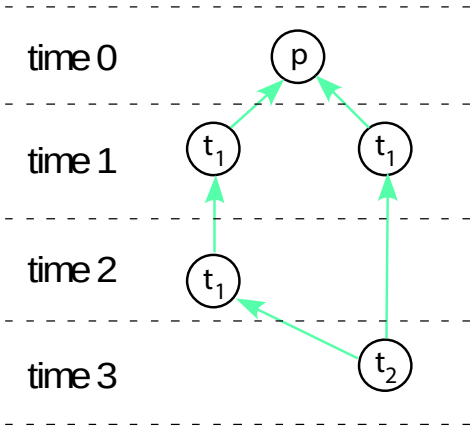
(a)


p
A
t1
A
b
A
t2
A
b
A

(b)


p
A
t1
A
A
b
A
A
b
A
t1
A
b
A
A
b
A
A
b
A
A
b
A
t2
A
b
A
t1

(c)


time 0
time 1
time 2
time 3
p
t1
t1
t1
t2

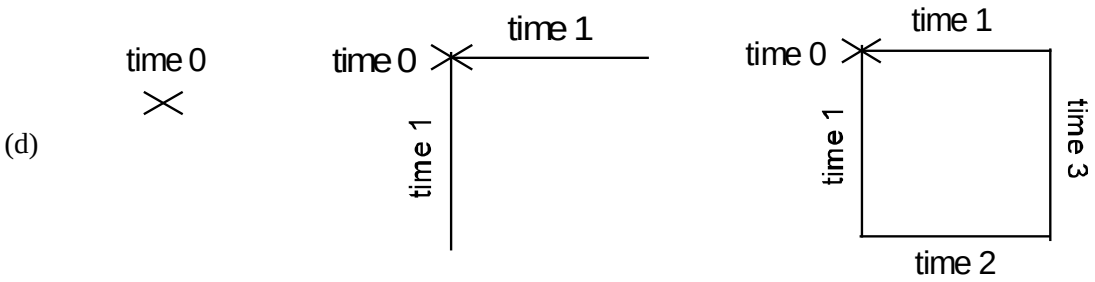


Figure 12: Forming a history of a square from a model of equilateral polygons. (a) A model of a class of equilateral polygons, including: a progenitor ρ as a single vertex, a transformation τ_1 that attaches an edge and a vertex to an existing vertex, and a transformation τ_2 that attaches an edge to two existing vertices. Dotted circles represent context of the transformations. (b) Constructing a square using the given model: first, two τ_1's are attached to the progenitor; second, another τ_1 is attached to the tail of an existing τ_1; and third, a τ_2 is attached to the tails of two existing τ_1's. (c) A directed acyclic graph representing the transformative history of the square formed by process (b). Time slices are annotated between dashed lines. The arrows indicate order and location of attachments, and thus regulate temporal information. At a high-level representation, this transformative history effectively encodes a struct. (d) A concrete visualization of the process that constructs the square. This form of visualization is easy to comprehend but does not reveal the underlying structural process.

4.4. Induction Problem Formulation

In this section, we formulate a supervised learning problem on gETS. We assume there is a *structure measurement device* (SMD, or a feature extractor) that converts each real world object into a composite graph (i.e. a feature graph). Noticing that different SMDs may produce different graph structures from the same object, we requires that all composites are produced by the same SMD.

Let $\mathcal{D} = \{\mathcal{D}^+, \mathcal{D}^-\}$ be a training set of composite graph examples, where $\mathcal{D}^+ = \{x_1, x_2 \ldots x_n\}$ is a set of *n* positive examples, and $\mathcal{D}^- = \{y_1, y_2 \ldots y_m\}$ is a set of *m* negative examples. Note that the number of negative examples is not necessarily equal to the number of positive examples. Let $\mathbb{M}$ be a set of all gETS models, and $\mathbb{C}$ be a set of all feature-graph composites.

Definition: A scoring function $\mathcal{F} : \mathbb{M} \times \mathbb{C} \rightarrow \mathcal{R}$ is a mapping from a pair $(\mathcal{M}, x)$ comprising of a gETS model and a composite graph to a real number that measures the degree to which x belongs to the class $\mathcal{L}(\mathcal{M})$ generated by gETS model $\mathcal{M}$.

When measuring the degree of a composite graph x belonging to the class $\mathcal{L}(\mathcal{M})$, we reconstruct x by starting from a progenitor ρ and applying a set of graph transformations $\mathcal{T}$ of model $\mathcal{M}$. Since $\mathcal{M}$ might be ambiguous, it is possible that reconstruction result is a set of histories in which none of the history is a subgraph of another history. We denote $\mathcal{M} \vDash x$ a set of unique, non-overlapping histories obtained by reconstructing x using $\mathcal{M}$.

$\mathcal{M} \vDash x = \{\hbar_1(x), \hbar_2(x), \ldots, \hbar_k(x)\}$ such that $\forall$ i, j $\in$ [1, k] : $\hbar_i(x) \not\subset \hbar_j(x)$.

It is possible that a transformative history $\hbar_i(x)$ is partial, i.e. $\hbar_i(x)$ is a subgraph of x, given that it is either impossible to apply any transformation of $\mathcal{M}$ to $\hbar_i(x)$, or applying any transformation of $\mathcal{M}$ to $\hbar_i(x)$ results in violation of the constraints set by x. However, in the training phase, we impose that all histories are complete.

Definition: Applying a scoring function $\mathcal{F}$ on a dataset $\mathcal{D} = \{\mathcal{D}^+, \mathcal{D}^-\}$ is equal to summation of measures of composites in $\mathcal{D}^+$ less summation of measures of composites in $\mathcal{D}^-$.

$$\mathcal{F}(\mathcal{M} : \mathcal{D}) = \sum_{x \in \mathcal{D}^+} \mathcal{F}(\mathcal{M}, x) - \sum_{y \in \mathcal{D}^-} \mathcal{F}(\mathcal{M}, y)$$

Intuitively, this scoring function rewards the ability of $\mathcal{M}$ to derive positive training examples and penalizes the ability of $\mathcal{M}$ to derive negative training examples. The induction problem is now transformed to an optimization problem that searches for a model $\mathcal{M}^*$ that maximizes the scoring function with respect to a training dataset.

$$\mathcal{M}^* = \underset{\mathcal{M}}{argmax}\, \mathcal{F}(\mathcal{M} : \mathcal{D})$$

In this work, we develop a novel induction algorithm that scales to real datasets and combines the power of statistical learning with graph-based pattern recognition.

4.5. Objective Function

Following statistical machine learning convention, we divide the scoring function into two terms: one primary term that guide the objective of the scoring function, and one secondary term that acts as regularization to prevent model overfitting. For the primary

term, we investigate two measures: (i) log-likelihood of probability of the training dataset, and (ii) typicality of the training dataset. For the secondary term, we use the degree of compression that penalizes the model when it over-compresses the training dataset. To express compression, we use graph description length that calculates number of bits required to describe training data given a model. The use of compression degree as a regularization term is contrary to methods that uses compression as the primary guide [142, 145, 149]. We call the primary term a "fitness" term, and the secondary term a "complexity" term. Note that the fitness term may incorporate both positive and negative training data, or just the positive data. Similar incorporation applies to the complexity term.

$$\mathcal{F}(\mathcal{M} : \mathcal{D}) = fitness(\mathcal{M} : \mathcal{D}) - \lambda \times complextiy(\mathcal{M} : \mathcal{D})$$

Where λ is a real valued, regularization parameter that governs how much overfitting is permitted.

4.5.1. Likelihood Probability

The first and foremost fitness term when trying to merge graph-based pattern recognition to the realm of statistical learning is probability. Our goal is maximizing the likelihood to observe the training dataset $\mathcal{D}$ given a candidate model $\mathcal{M}$. We derive a formulation based on stochastic context-free grammars [148, 164, 178-181] and stochastic context-free graph grammars [26, 146].

Under this scheme, each graph transformation is associated with a probability such that summing over all transformation with the same context yields 1.

Definition: A probability measure on graph transformations is a function from a countable set of transformations to a unit real number interval $p : \mathcal{T} \rightarrow [0, 1]$ such that:

$$\sum_{\substack{\tau \in \mathcal{M} \\ x = context(\tau)}} p(\tau \mid x) = 1$$

Having a probability measure on transformations, we make an I.I.D (independently and identically distributed) assumption that composite examples are generated by independently applying transformations at random order. Thus, probability of a history is product of probability of participating transformations.

Definition: Probability of a transformative history (x) of composite graph x generated under a model $\mathcal{M}$ is product of probability of transformations participating in the history.

$$p(\hbar(x) \mid \mathcal{M}) = \prod_{\substack{\tau \in \hbar(x) \\ \tau \in \mathcal{M}}} p(\tau)$$

Given that the model $\mathcal{M}$ might be ambiguous, i.e. there are more than one history for a composite graph $\mathcal{M} \vDash x = \{\hbar_1(x), \hbar_2(x), \ldots, \hbar_k(x)\}$. Probability of generating a composite graph is summation of probability of generating all of its histories.

Definition: Probability of a composite graph x under a model $\mathcal{M}$ is summation of probability of its histories under $\mathcal{M}$.

$$p(x \mid \mathcal{M}) = \sum_{\hbar(x) \in \mathcal{M} \vDash x} p(\hbar(x) \mid \mathcal{M})$$

As the last condition, probability of all composites generated by a model has to sum to 1. This condition is similar to the *tightness* [178] property of formal grammars.

Definition: A model $\mathcal{M}$ is *tight* (or *consistent*) when summing over all composites in its class $\mathcal{L}(\mathcal{M})$ yields 1.

$$\sum_{x \in \mathcal{L}(\mathcal{M})} p(x \mid \mathcal{M}) = 1$$

In our inductive learning problem, we are given a training set comprising of positive and negative examples. The goal is identifying the best model that gives high probability to positive examples and low probability to negative examples. Here, we assume the existence of a model, and we want to calculate the likelihood of training examples given the model.

The challenging induction task is to assign a probability to each graph transformation in the model. We acknowledge that $\mathcal{M}$ might be an ambiguous model, and estimating likelihood probability of graph transformations using a method similar to the Inside-Outside algorithm [148] with Expectation Maximization (EM) [147] is exponentially expensive. The problem lies in enumeration of all subgraphs to calculate the inside and outside potential functions.

We choose to estimate the transformation probability from formative histories, rather than from composites. Chi and Geman [178] showed that either direct maximum likelihood estimation using formative histories or via an iterative expectation maximization (EM) estimation of likelihood using composites are *tight*.

As a result, we assume that beside a grammar $\mathcal{M}$, we also have at hands formative histories of training composites: $\mathcal{D} = \{\mathcal{D}^+, \mathcal{D}^-\}$, $\mathcal{D}^+ = \{\hbar(x_1), \hbar(x_2),\ldots, (x_n)\}$ and $\mathcal{D}^- = \{\hbar(y_1), \hbar(y_2),\ldots, \hbar(y_m)\}$.

Definition: Maximum likelihood probability of a graph transformation τ is simply count of τ divided by count of transformations having the same context as τ [178].

$$p(\tau) = \frac{count(\tau \mid \mathcal{D})}{\sum_{context(\tau')=context(\tau)} count(\tau' \mid \mathcal{D})}$$

where:

- $\tau, \tau' \in \mathcal{M}$, and
- count($\tau \mid \mathcal{D}$) is number of times τ is used in formative histories in $\mathcal{D}$.

Given probability of transformations, probability of a transformative history is the product of participating transformations as above. The last piece of the puzzle is to formulate a fitness function based on likelihood probability that bias towards positive training set. We also take into account the difference in size between the positive set and the negative set by taking the average of the score on each dataset.

Definition: Likelihood score of a training set $\mathcal{D} = \{\mathcal{D}^+, \mathcal{D}^-\}$ given a model $\mathcal{M}$ is average of probability of positive formative histories less average of probability of negative formative histories, given probability of transformations is estimated by maximum likelihood.

$$L(\mathcal{M} : \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} p\big(\hbar(x_i)\big) - \frac{1}{m}\sum_{j=1}^{m} p\left(\hbar\big(y_j\big)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \prod_{\substack{\tau \in \hbar(x_i) \\ \tau \in \mathcal{M}}} p(\tau) - \frac{1}{m}\sum_{j=1}^{m} \prod_{\substack{\tau \in \hbar(y_j) \\ \tau \in \mathcal{M}}} p(\tau)$$

Note that this formulation effectively trains a mixture model that maximize likelihood of positive examples and minimize likelihood of negative examples.

4.5.2. Family Resemblance Typicality

While probability has an objective interpretation of events as the degree of uncertainty, it does not take into account the psychological, subjective human perception of events. The typicality measure is motivated by psychological research on how human beings perceive object-class relationship.

Loken and Ward performed extensive experiments on measuring typicality [182] with a interdisciplinary research that involves psychology and business marketing. In the research, participants were first shown a list of product categories, and then instructed to rank how typical a test product is subject to some categories. They [183] showed that there are three groups of factors that govern typicality perception: (i) Family resemblance, or physical similarity, (ii) Attribute structure, or ideals, and (iii) Frequency, or familiarity. Interestingly, the work showed that frequency of presenting an attribute in a class, so called familiarity, is the least determinant of perceived typicality. On the other hand, family resemblance was shown to work well in measuring the degree of membership of prototypical objects within the context of a class [183]. This property fits our goal as we hypothesize training examples are generated from a distribution of a class of graphs.

In this work, we utilize two prominent research to measure typicality: (i) Rosch's family resemblance (FR) formulation [184], and (ii) Tversky's similarity models [185]. According to Rosch, typicality of an object is measured by its similarity to objects of the

same class, and dissimilarity to objects of other classes. The original formulation of family resemblance typicality is a weighted sum of unique attributes of an object, where the weights are outputs of a monotonically increasing function of supports of the attributes. Mapping to our gETS framework:

- An object is represented by a transformative history (x) of a composite graph x.
- An attribute is a graph transformation participating in constructing x.

Definition: Family resemblance (FR) typicality of a transformative history $\hbar(x)$ with respect to a gETS model $\mathcal{M}$ is summation of a monotonically increasing function of support of graph transformations in the model that participate in parsing composite graph x. The support is counted from the training dataset $\mathcal{D}^?$, where $\mathcal{D}^?$ is either $\mathcal{D}^+$ or $\mathcal{D}^-$.

$$FR(\hbar(x) \mid \mathcal{M}) = \sum_{\substack{\tau \in \hbar(x) \\ \tau \in \mathcal{M}}} \phi(support(\tau \mid \mathcal{D}^?))$$

where

$$support(\tau \mid \mathcal{D}^?) = \sum_{x \in \mathcal{D}^?} \delta(x, \tau)$$

$$\delta(x, \tau) = \begin{cases} 1 \Leftrightarrow \exists \hbar(x) \in \mathcal{M} \vDash x \ such\ that\ \tau \in \hbar(x) \\ 0\ otherwise \end{cases}$$

Two popular choices of ϕ are: (i) an identity function, and (ii) a logarithm function.

If $\mathcal{M}$ is an ambiguous model, there could be more than one history for a particular composite graph $\mathcal{M} \vDash x = \{\hbar_1(x), \hbar_2(x), \ldots, \hbar_k(x)\}$. If we follow the probability train of

thought that assigns overall typicality of the composite graph to be summation of typicality of its valid histories, we may end up in two adverse situations that are contrary to human perception: (i) a composite graph with many untypical histories is promoted to become typical; and (ii) a composite graph with few typical histories is downgraded to become untypical. For that reason, we choose to follow the normal practice in psychology that finds a prototypical object to represent a group of similar objects. The prototype could be thought of as the mean of a cluster of points, in which each point represents an object. We apply this convention to compute the mean of typicality of histories of a composite graph.

Definition: Family resemblance typicality of a composite graph x under an ambiguous model $\mathcal{M}$ is mean of typicality of all its formative histories.

$$FR(x) = \frac{1}{|\mathcal{M} \vDash x|} \sum_{\hbar(x) \in \mathcal{M} \vDash x} FR(\hbar(x) \mid \mathcal{M})$$

where

- $| \mathcal{M} \vDash x |$ is cardinality of the derived set.

Although Rosch showed by experiments [184] that this formulation works in practice, Loken et al pointed out that the original family resemblance formulation is better suited to intra-class, object-based tasks [183]. Since our problem involves inter-class differentiation between positive and negative training data, we incorporate Tversky's similarity models [185] to discriminate between a positive class and a negative class. We interpret a dissimilarity measure is equal to a complement of the similarity measure:

$$dissimilarity = \begin{cases} -similarity : Contrast\ model \\ 1 - similarity : Ratio\ model \end{cases}$$

Paring Tversky's models with Rosch's family resemblance, we derive a new formulation for measuring the typicality of a positive dataset reflected on a negative dataset.

Definition: Family resemblance of a dataset $\mathcal{D} = \{\mathcal{D}^+, \mathcal{D}^-\}$ is split into three portions:

- $(\mathcal{D}^+ \setminus \mathcal{D}^-)$ is the portion belongs to $\mathcal{D}^+$ but does not belong to $\mathcal{D}^-$,
- $(\mathcal{D}^- \setminus \mathcal{D}^+)$ is the portion belongs to $\mathcal{D}^-$ but does not belong to $\mathcal{D}^+$,
- and $(\mathcal{D}^+ \cap \mathcal{D}^-)$ is the common portion belonging to both $\mathcal{D}^+$ and $\mathcal{D}^-$.

And family resemblance measures of the three portions are combined using complementary formulation of Tversky's contrast / ratio models [185].

Contrast model:

$$FR(\mathcal{D}) = \alpha \times FR(\mathcal{D}^+ \setminus \mathcal{D}^-) + \beta \times FR(\mathcal{D}^- \setminus \mathcal{D}^+) - \gamma \times FR(\mathcal{D}^+ \cap \mathcal{D}^-)$$

Ratio model:

$$FR(\mathcal{D}) = \frac{\alpha \times FR(\mathcal{D}^+ \setminus \mathcal{D}^-) + \beta \times FR(\mathcal{D}^- \setminus \mathcal{D}^+)}{\alpha \times FR(\mathcal{D}^+ \setminus \mathcal{D}^-) + \beta \times FR(\mathcal{D}^- \setminus \mathcal{D}^+) + \gamma \times FR(\mathcal{D}^+ \cap \mathcal{D}^-)}$$

Notice that this new formulation effectively trains two models simultaneously, one for the positive class, and another for the negative class. We call this training scheme *dual-model training*. The first term FR($\mathcal{D}^+ \setminus \mathcal{D}^-$) fits a model $\mathcal{M}^+$ to the positive class, so that to maximize the typicality of the positive training dataset given the positive model. The second term FR($\mathcal{D}^- \setminus \mathcal{D}^+$) fits a model $\mathcal{M}^-$ to the negative class, so that to maximize the

typicality of the negative training dataset given the negative model. And the third term FR($\mathcal{D}^+ \cap \mathcal{D}^-$) penalizes the overlapping portion between two classes generated by the two models.

To compute the typicality of the difference and intersection between $\mathcal{D}^+$ and $\mathcal{D}^-$, we first compute the portion of graph transformations that belong to $\mathcal{M}^+$ but does not belong to $\mathcal{M}^-$ and vice versa. Given $\mathcal{M}^+ = (\rho^+, \mathcal{T}^+)$ and $\mathcal{M}^- = (\rho^-, \mathcal{T}^-)$, we have:

$$\mathcal{T}^{+\setminus -} = \mathcal{T}^+ \setminus \mathcal{T}^-$$

$$\mathcal{T}^{-\setminus +} = \mathcal{T}^- \setminus \mathcal{T}^+$$

$$\mathcal{T}^{\cap} = \mathcal{T}^+ \cap \mathcal{T}^-$$

In calculating family resemblance of the non-overlapping and overlapping portions between the two training datasets, we notice that the two datasets may have different number of examples: i.e. *n* positive examples, and *m* negative examples. To avoid bias towards a dataset with more examples, averaging over the number of examples contained in each dataset is performed.

Definition: Family resemblance typicality of each non-overlapping or overlapping portion between the positive and negative datasets is average of typicality of all composites in the respective dataset, measured using only transformations in the corresponding portion.

$$FR(\mathcal{D}^+ \setminus \mathcal{D}^-) = \frac{1}{n} \sum_{\substack{x \in \mathcal{D}^+ \\ \tau \in \mathcal{T}^{+\setminus -}}} FR(x) = \frac{1}{n} \sum_{x \in \mathcal{D}^+} \frac{1}{|\mathcal{M} \vDash x|} \sum_{\substack{\hbar(x) \in \mathcal{M} \vDash x \\ \tau \in \mathcal{T}^{+\setminus -}}} FR(\hbar(x) \mid \mathcal{M}^+)$$

$$= \frac{1}{n} \sum_{x \in \mathcal{D}^+} \frac{1}{|\mathcal{M} \vDash x|} \sum_{\hbar(x) \in \mathcal{M} \vDash x} \sum_{\substack{\tau \in \hbar(x) \\ \tau \in \mathcal{T}^{+\setminus -}}} \phi(support(\tau \mid \mathcal{D}^+))$$

$$FR(\mathcal{D}^- \setminus \mathcal{D}^+) = \frac{1}{m} \sum_{\substack{x \in \mathcal{D}^- \\ \tau \in \mathcal{T}^{-\setminus +}}} FR(x) = \frac{1}{m} \sum_{x \in \mathcal{D}^-} \frac{1}{|\mathcal{M} \vDash x|} \sum_{\substack{\hbar(x) \in \mathcal{M} \vDash x \\ \tau \in \mathcal{T}^{-\setminus +}}} FR(\hbar(x) \mid \mathcal{M}^-)$$

$$= \frac{1}{m} \sum_{x \in \mathcal{D}^-} \frac{1}{|\mathcal{M} \vDash x|} \sum_{\hbar(x) \in \mathcal{M} \vDash x} \sum_{\substack{\tau \in \hbar(x) \\ \tau \in \mathcal{T}^{-\setminus +}}} \phi(support(\tau \mid \mathcal{D}^-))$$

$$FR(\mathcal{D}^+ \cap \mathcal{D}^-) = \frac{1}{n} \sum_{\substack{x \in \mathcal{D}^+ \\ \tau \in \mathcal{T}^{\cap}}} FR(x) + \frac{1}{m} \sum_{\substack{x \in \mathcal{D}^- \\ \tau \in \mathcal{T}^{\cap}}} FR(x) =$$

$$= \frac{1}{n} \sum_{x \in \mathcal{D}^+} \frac{1}{|\mathcal{M} \vDash x|} \sum_{\hbar(x) \in \mathcal{M} \vDash x} \sum_{\substack{\tau \in \hbar(x) \\ \tau \in \mathcal{T}^{\cap}}} \phi\big(support(\tau \mid \mathcal{D}^+)\big)$$

$$+ \frac{1}{m} \sum_{x \in \mathcal{D}^-} \frac{1}{|\mathcal{M} \vDash x|} \sum_{\hbar(x) \in \mathcal{M} \vDash x} \sum_{\substack{\tau \in \hbar(x) \\ \tau \in \mathcal{T}^{\cap}}} \phi(support(\tau \mid \mathcal{D}^-))$$

We notice that family resemblance typicality depends on two factors: (i) support of transformations, and (ii) uniqueness of transformations. However, according to Apriori downward closure property [186], support and size of transformations are contradicting factors: i.e. the higher the support, the more likely the transformation' size is small and vice versa. On the other hand, size and uniqueness tends to go together: i.e. the larger a transformation, the more likely there are many unique transformations with the same size. Combining the two trends, we realize that support and uniqueness of transformations are

contradictory factors: i.e. the higher the support of a transformation, the lesser the number of transformations with the same support. Therefore, at the quantitative level, maximizing family resemblance typicality means a competition between support and size of transformations.

When applying this formulation to a real dataset, we acknowledge that it is exponentially expensive to compute the full set $\mathcal{M} \vDash x$ for each composite graph. As a result, we assume that beside a model $\mathcal{M}$, we also have at hands formative histories of training composites: $\mathcal{D}$ = {$\mathcal{D}^{+}$, $\mathcal{D}^{-}$}, $\mathcal{D}^{+}$ = {$\hbar(x_1)$, $\hbar(x_2)\ldots$ (x_n)} and $\mathcal{D}^{-}$ = { $\hbar(y_1)$, $\hbar(y_2)\ldots$ $\hbar(y_m)$}. Formulation of family resemblance typicality of each non-overlapping or overlapping portion between the positive and negative datasets becomes:

$$FR(\mathcal{D}^{+} \setminus \mathcal{D}^{-}) = \frac{1}{n}\sum_{i=1}^{n} FR\left(\hbar(x_i)|\mathcal{T}^{+\setminus-}\right) = \frac{1}{n}\sum_{i=1}^{n} \sum_{\substack{\tau \in \hbar(x_i) \\ \tau \in \mathcal{T}^{+\setminus-}}} \phi(support(\tau \mid \mathcal{D}^{+}))$$

$$FR(\mathcal{D}^{-} \setminus \mathcal{D}^{+}) = \frac{1}{m}\sum_{j=1}^{m} FR\left(\hbar(y_j)|\mathcal{T}^{-\setminus+}\right) = \frac{1}{m}\sum_{j=1}^{m} \sum_{\substack{\tau \in \hbar(y_j) \\ \tau \in \mathcal{T}^{-\setminus+}}} \phi(support(\tau \mid \mathcal{D}^{-}))$$

$$FR(\mathcal{D}^{+} \cap \mathcal{D}^{-}) = \frac{1}{n}\sum_{i=1}^{n} FR\left(\hbar(x_i)|\mathcal{T}^{\cap}\right) + \frac{1}{m}\sum_{j=1}^{m} FR\left(\hbar(y_j)|\mathcal{T}^{\cap}\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \sum_{\substack{\tau \in \hbar(x_i) \\ \tau \in \mathcal{T}^{\cap}}} \phi(support(\tau \mid \mathcal{D}^{+}))$$

$$+ \frac{1}{m}\sum_{j=1}^{m} \sum_{\substack{\tau \in \hbar(y_j) \\ \tau \in \mathcal{T}^{\cap}}} \phi(support(\tau \mid \mathcal{D}^{-}))$$

As a side discussion, we notice that in contrast to family resemblance formulation where an implicit prototype is assumed, another formulation of typicality that directly measures the distance from an explicit prototype is used in fuzzy clustering [187]. This possibilistic typicality is based on the possibility theory founded by Zadeh [188] and further clarified by Dubois et.al. [189]. In either case, typicality (possibility) measures the degree of belief that, consequently, does not required to sum to 1 across all categories like probability. This property makes typicality robust to noisy data [190].

4.5.3. Minimum Description Length

We are motivated by an intuition that a good model does not only fit closely to training data and generalize well to test data, but also yield the best compression in general. Our intuition aligns with the *readability* issue in structural pattern recognition [27] which concerns with the simplicity of learnt models so that human experts can interpret the meaning of production rules. To this extend, we hypothesize that minimizing complexity might help *knowledge discovery*.

In Graphical Transformation System (gETS) formalism, we have a model consists of a progenitor and a set of graph transformations. Using the model, composites are represented by formative histories which are directed acyclic graphs of transformations going towards a single sink at the progenitor. The complexity of the entire system could be formulated as:

$$\text{complexity}(\mathcal{M} : \mathcal{D}) = \text{complexity}(\mathcal{M}) + \text{complexity}(\mathcal{D}^{+} \mid \mathcal{M})$$

Above formula implies that we only care about complexity of the positive training dataset, because it is the data we want to discover and understand. In general, we do not care how complex is the negative data organized.

Given our system is represented as graphs, it is natural to follow this chain of logic that we want to have a good measure for graph complexity. However, there are more than one way to measure graph complexity; some examples are: using Boolean functions [191], adjacency matrix linear complexity [192], and structural entropy [193] etc. In this work, we use minimum description length principle to measure graph complexity in terms of the number of bits required to encode the graphs.

Minimum description length (MDL) principle [143, 194] was introduced by Rissanen stating that the best model is the one that minimizes description length of the training dataset. Along the line of methods that uses MDL principle, Subdue [142] was the first success that mine subgraphs that yield highest compression to an input graph. SubdueGL [144] was an expansion of Subdue that infers recursive and variable production rules for a node replacement context-free graph grammar. Doshi et al [145] expanded Subdue to learn a simpler graph grammar than SubdueGL and also gave preliminary inference on probability of production rules based on frequency of substructures. Ates et al [149] relaxed the minimum description length principle to a minimum graph size problem that learns a graph grammar based on SubdueGL and has application in visual programming.

Our formulation of graph description length is based on Cook's method [142].

Definition: Number of bits required to encode vertices of a composite graph c, a transformation τ, and a transformative history $\mathcal{h}$ are:

$$VB(c) = \log_2|V| + |V| \times \log_2 L(\mathcal{D})$$

$$VB(\tau) = \log_2|V| + |V| \times \log_2 L(\mathcal{D})$$

$$VB(\hbar) = \log_2|V| + |V| \times \log_2 L(\mathcal{M})$$

Where:

- |V| is the number of vertices in the composite graph, transformation, and history respectively.
- $L(\mathcal{D})$ is the number of unique primitive features in the training dataset.
- $L(\mathcal{M}) = |\mathcal{T}| + 1$ is the number of transformations plus the progenitor of the model.

Definition: Number of bits required to encode the adjacency matrix of a composite graph c, a transformation τ, and a transformative history $\hbar$ are:

$$RB(c) = (|V| + 1) \times \log_2\left(\max_{v \in c} \deg(v) + 1\right) + \sum_{v \in c} \log_2 \binom{|V|}{\deg(v)}$$

$$RB(\tau) = (|V| + 1) \times \log_2\left(\max_{v \in \tau} \deg(v) + 1\right) + \sum_{v \in \tau} \log_2 \binom{|V|}{\deg(v)}$$

$$RB(\hbar) = (|V| + 1) \times \log_2\left(\max_{v \in \hbar} \deg(v) + 1\right) + \sum_{v \in \hbar} \log_2 \binom{|V|}{\deg(v)}$$

Where:

- |V| is the number of vertices in the composite graph, transformation, and history respectively.

- v is a primitive vertex.
- deg(v) is degree of vertex v.
- $\binom{n}{k}$ is number of combination (without permutation) of k elements out of n elements.

Finally, we do not need to encode the edges individually because with feature graph representation, edges has no label and there is only one undirected edge between two adjacent vertices. For a transformative history, edge direction is fixed by the progenitor, so that the history becomes a single-sink graph.

Next, we formulate additional parts required by our framework.

Definition: In a transformation τ with $|V_{context}|$ number of vertices in the context, we need additional $\log_2(|V_{context}| + 1)$ bits to encode the number of context vertices, and $\log_2 \binom{|V|}{|V_{context}|}$ bits to encode the combination of context vertices.

$$AB(\tau) = \log_2(|V_{context}| + 1) + \log_2 \binom{|V|}{|V_{context}|}$$

Definition: Description length of a composite graph c, a transformation τ, and a transformative history $\mathcal{h}$ is sum of number of bits VB to encode the vertices, number of bits RB to encode the adjacency matrix, and number of bits AB to encode additional structural information.

$$DL(c) = VB(c) + RB(c)$$

$$DL(\tau) = VB(\tau) + RB(\tau) + AB(\tau)$$

$$DL(\hbar) = VB(\hbar) + RB(\hbar)$$

Definition: Description length of a model $\mathcal{M}$ is sum of description length of its composite graph progenitor ρ and its transformations $\{\tau_1, \tau_2, \ldots, \tau_k\}$.

$$DL(\mathcal{M}) = DL(\rho) + \sum_{i=1}^{k} DL(\tau_i)$$

Definition: Description length of a training dataset $\mathcal{D}^+$ given a model $\mathcal{M}$ is sum of description length of formative histories of training examples derived by the model.

$$DL(\mathcal{D}^+|\mathcal{M}) = \sum_{x \in \mathcal{D}^+} DL(\hbar_{\mathcal{M}}(x))$$

Definition: Description length (or complexity) of the entire system is sum of description length of the model and description length of the training dataset.

$$DL(\mathcal{M} : \mathcal{D}) = DL(\mathcal{M}) + DL(\mathcal{D}^+ \mid \mathcal{M})$$

4.5.4. Normalizing Measurements

When combining a fitness measure (likelihood probability or family resemblance typicality) with a complexity measure (minimum description length), it is noticeable that each measure type has a different scale. Even for the probability measure, the overall range is [0, 1], but it is easy to realize that probability of a composite graph as a product of probability of transformations is most likely close to 0. Other measures, such as family resemblance with contrast model or description length has no theoretical bound.

As an empirical approach, we run the algorithm on each measure type individually one time to record the measurement ranges, then use the ranges to normalize the measures in combination. Specifically, denote min(m) and max(m) to be emperical minimal and maximal values of a measure m, we have normalized value of m to range [0, 1]:

$$\text{norm}(m) = \frac{m - \min(m)}{\max(m) - \min(m)}$$

4.6. Induction Algorithm

A naïve, brute-force approach would: (i) generates all frequent subgraphs of the training database $\mathcal{D}$ by an offline algorithm such as FSG [195], (ii) concatenates each pair of subgraphs to form a transformations, (iii) groups transformations to form candidate models $\mathcal{M}$s, (iv) computes score $\mathcal{F}(\mathcal{M} : \mathcal{D})$ of each model, and (v) retains the model with best score. The first challenge with this approach is intractability of the set of candidate models. Specifically, the set of frequent subgraphs might be exponential, leading to the set of transformations (a Cartesian product of the set of subgraphs) to be exponential, which in turn leads to the set of candidate models (a power set of the set of transformations) to be hyper-exponential. The second challenge with this approach is parsing. Given a candidate model, we have to find the transformative history that generate each training example, so that the scoring function can be evaluated. This implies multiple subgraph matchings of transformations onto composites, which is equivalent to subgraph isomorphism, a NP-complete problem [196].

We approach this problem with an approximate algorithm. First, all training composites are aligned to create an alignment graph called a *super-composite*. After that,

different ways to summarize [19] the super-composite are explored. The effect of summarizing the super-composite is instantaneous summarization of all training examples. From the summarization, we can infer a model and corresponding formative histories of all composites, and thus enables evaluation of the scoring function.

4.6.1. Graph Alignment

Keeping only the *sufficient statistics* [197], information on a training dataset relevant to a learning algorithm, is a normal practice in statistical machine learning as a way to reduce memory consumption. In the syntactic counterpart, constructing a sufficient statistics is not only meant for space savings, but also helps speeding up the learning algorithm due to the compression of input information. An example is the super-tree of Torsello and Hancock [97] that is a union of input tree-patterns. Our method aims at reducing complexity of the model induction problem by alignment (superimposition, or union) of training examples into a *super-composite*. This operation can be thought of as a preprocessing step that pre-computes the sufficient statistics of the training examples. Although out method is developed independently, it can be related as a generalization of the union tree [97].

In our problem formulation, the graph alignment task is usually referred to as a global network alignment [198]. Depending on the choice of similarity function, an optimal alignment could be a NP-hard problem [198]. However, having a perfect alignment is not our goal. In the asymptotic limit, no matter what quality of the alignment, the learning algorithm converges to the same global optimum. The subtlety is we may not have enough memory to store all discovered transformations, and we do not want to wait for

too long for the learning algorithm to converge. In general, the better alignment quality we have, the faster we reach convergence.

As a starting point, we notice that candidate models have a special element called a progenitor. Intuitively, the progenitor is the maximal frequent subgraphs with 100% support among training examples. However, mining maximal frequent subgraphs is a computationally expensive problem [199], and we want to save computational power to the model induction task. As a result, we relax the progenitor detection problem to a *primitive progenitor* detection problem, i.e. a single primitive feature that has 100% support among training examples. Finding primitive progenitors only requires linear time in number of vertices in the training dataset.

Having the primitive progenitors, we align graphs around that common vertex in a breadth-first style. The algorithm is also linear in number of vertices in the training dataset.

Algorithm 1: Find primitive progenitors

Input: A collection $\mathcal{D}^+$ of positive feature graphs

Output: Primitive progenitors

1: count[v] = 0 $\forall$ unique primitive feature $v \in \mathcal{D}$

2: For each feature graph $g \in \mathcal{D}^+$

3: visited[v] = false $\forall$ unique primitive feature $v \in g$

4: For each primitive vertex $v \in g$

5: If not visited[v]

6: count[v] = count[v] + 1

7: visited[v] = true

8: Return {v | count[v] == | $\mathcal{D}^+$|}

End

In the algorithm that find primitive progenitors, we only concern the positive training dataset $\mathcal{D}^+$. The logic is if a negative example does not contains such a progenitor, it can be safely disregarded as not belonging to the class that generate the positive dataset. That said, our model induction algorithm only discriminates between close enough positive and negative examples.

Having the progenitor, the goal of building the super-composite is vertex-preserving, i.e. two vertices are superimposed if and only if they share the same property in the two parent composites. We call a vertex of the super-composite a super-vertex, and a vertex of a training example a composite vertex. Internally, the super-composite maintains a mapping $\mathcal{M}$ that tells which composite vertex is aligned with which super-vertex. Algorithm 2 provides pseudo code for building a super-composite given a primitive progenitor, and figure 13 illustrates a super-composite built from two composites.

Algorithm 2: Build a super-composite

Input: A dataset $\mathcal{D}$ of composite graphs + A primitive progenitor ρ.

Output: A super-composite $\mathbb{s}$, containing a map $\mathcal{M}$ from each super-vertex to a composite vertex.

1: $\mathcal{M} = \emptyset$

2: $\mathbb{s} \leftarrow \rho$

3: For each composite graph $c \in \mathcal{D}$

4: For each location v_0 of ρ in c // each location of ρ starts a layer

5: $\mathcal{M}[\rho, c] = v_0$

6: From ρ, scan both $\mathbb{s}$ and c by BFS to find ∀ i, j:

7: L($\mathbb{s}$, i) = the set of vertices in $\mathbb{s}$ with shortest path to ρ is i

8: L(c, j) = the set of vertices in c with shortest path to ρ is j

9: For k = 1 : depth(c)

10: For each vertex $v_c \in$ L(c, k)

11: $e(v_c)$ = a set of edges incident on v_c that come from L(c, k-1)

12: For each vertex $v_s \in$ L($\mathbb{s}$, k)

13: $e(v_s)$ = a set of edges incident on v_s that come from L($\mathbb{s}$, k-1)

14: If $e(v_s) \equiv e(v_c)$

15: $\mathcal{M}[v_s, c] = v_c$ // align v_c with v_s

16: Break

17: If not found any matched $e(v_s)$

18: $v_s = \mathbb{s} \leftarrow v_c$ // copy v_c to $\mathbb{s}$ as a new vertex

19: Connect v_s to all vertices in $\mathcal{M}^{-1}$[L(c, k-1)]

20: Return $\mathbb{s}$

End

Intuitively, the algorithm constructs a super-composite incrementally from each example. It matches each composite vertex to the corresponding super-vertices at the same BSF depth. If a match is found, it records the alignment by using the vertex mapping $\mathcal{M}$. Otherwise, a new super-vertex is created to match with the composite vertex.

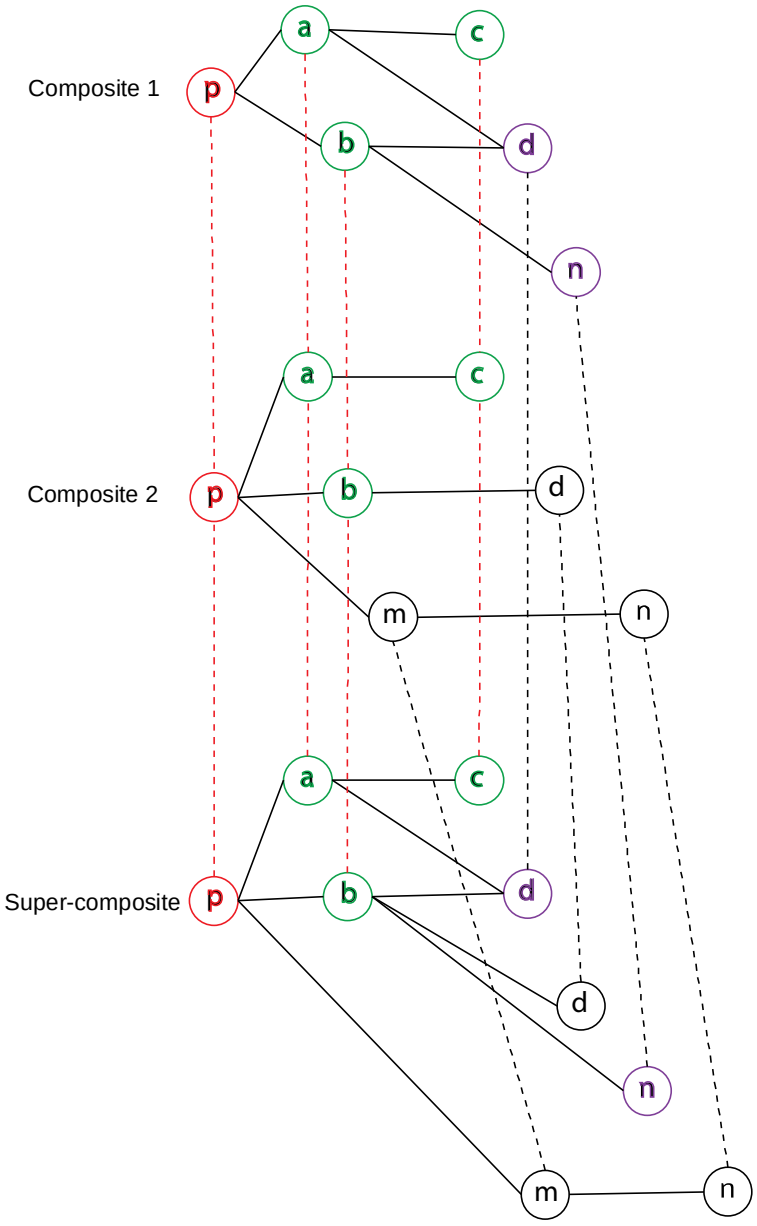

Composite 1
p
a
c
b
d
n
Composite 2
p
a
c
b
d
m
n
Super-composite
p
a
c
b
d
d
n
m
n

Figure 13: Constructing a super-composite from two component composites. The progenitor is represented by a red *p* vertex. Vertices *a*, *b*, and *c* that match both composites are colored in green. Vertices *d* and *n* that only match composite graph 1 are colored in purple. Vertices *d*, *m*, and *n* that only matches composite graph 2 are colored in black. Red dotted lines project through vertices that match both composites. Black dotted lines project through vertices that only match one composite graph. Notice that vertex *d* in composite graph 1 is adjacent to both *a* and *b*, so it does not match vertex *d* in composite graph 2 that is adjacent to only *b*. As a result, the two vertices *d* in two component composites projects to two different vertices *d* in the super-composite. The same argument applies to vertex *n*.

It is noted that each primitive progenitor yields a super-composite, and we may end up with lots of super-composites due to the amount of primitive progenitors. This issue arises because we avoid finding the maximal frequent subgraphs. We could reasonably hypothesize that the number of maximal frequent subgraphs with 100% support is rare, and thus the number of progenitors is rare. Secondly, a progenitor might contain several primitive progenitors, so their super-composite should be equivalent. As a result, primitive progenitors can be partitioned into equivalent classes so that each class yields the same super-composite.

Algorithm 3: Heuristic filtering and grouping of super-composites

Input: A list of super-composites $\{\mathbb{s}_1, \mathbb{s}_2, \ldots, \mathbb{s}_k\}$.

Output: Assignment to equivalent classes $\{q_1, q_2, \ldots, q_m\}$ where $m \ll k$.

1: visited[i] = false ∀ i = 1 : k
2: For i = 1 : k
3: If not visited[i] && $|\mathbb{s}_i| <$ threshold
4: Form a new equivalent class q_m
5: $q_m \leftarrow \mathbb{s}_i$
6: For j = i + 1 : k
7: If not visited[j] && $\mathbb{s}_j \equiv \mathbb{s}_i$
8: $q_m \leftarrow \mathbb{s}_j$
9: visited[j] = true

End

In line 3, $|\mathbb{s}_i|$ is total number of vertices across all layers, and the threshold is empirically set to 50,000. In line 7, $\mathbb{s}_j \equiv \mathbb{s}_i$ means the two super-composites are isomorphic. The rationale behind rejection of super-composites with too many vertices is that a noisy primitive progenitor (one that does not belong to any maximal frequent subgraph) tends to appear many times in each example, and thus yields a super-composites with many layers.

4.6.2. Simultaneous Graph Summarization

Given a super-composite, we want to explore different ways to summarize it. Since the super-composite keeps a map $\mathcal{M}$ that tells which composite vertices is mapped to a super-vertex, a summary of the super-composite provides simultaneous summarization of its constituting composites. As a metaphor, the super-composite can be thought of as a birthday cake, and its constituting composites are layers of the cake. Summarizing the super-composite is similar to cutting the cake into pieces, and each piece must go through all layers. The left part of figure 14 illustrates the idea that a summary of the super-composite induces simultaneous summary of its constituting composites.

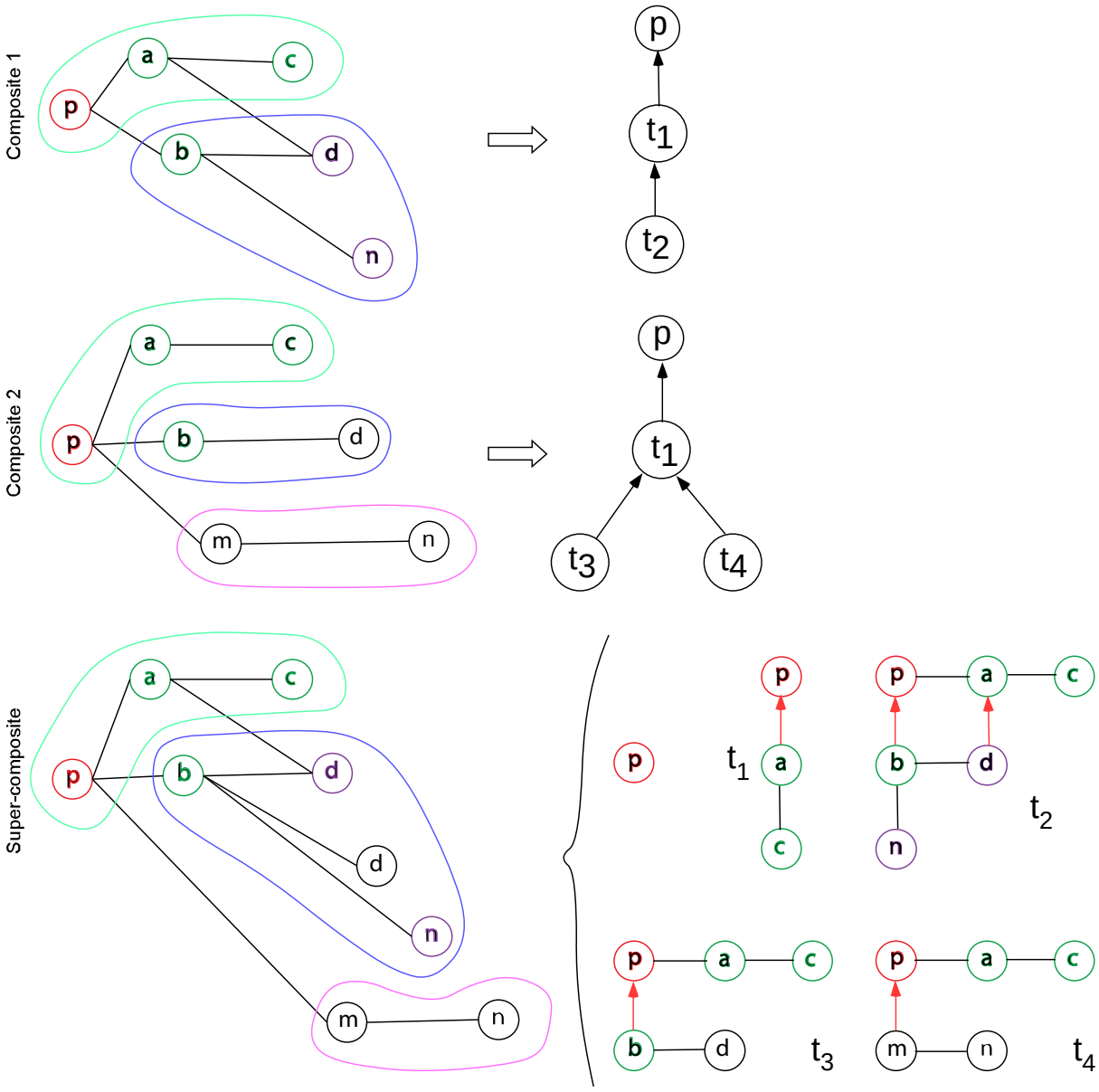


Figure 14: Summarization of a super-composite built from two constituting composites. On the left is a summary of the super-composite into three partitions: cyan, purple, and pink. This summary induces simultaneous summary of composite graph 1 and composite graph 2. On lower right corner is the model induced from the summarization. The model contains the progenitor and four transformations τ_1, τ_2, τ_3, and τ_4. In a transformation, red directed edges points from its tail to its context. On the right of the two composites are corresponding formative histories using the induced model.

Going back to our hypothesis that composites are generated by applying transformations to the progenitor, we notice that a summary gives us both the transformations and the transformative history. Given a summary of a composite graph, if we perform a Breadth-First Search (BFS) from the progenitor, we will encounter the summarized partitions at incremental depth. Denote the partition that contains the progenitor is the depth-0 partition, partitions with shortest path is 1 to the depth-0 partition to be depth-1 partitions, partitions with shortest path is 2 to the depth-0 partition to be depth-2 partitions, and so on. If we assign direction to the super-edges that connects partitions so that they point towards the progenitor (the progenitor becomes a sink vertex), we have a directed acyclic graph that is similar to a transformative history, in which each partition is the tail of a transformation. To find the context of transformations, we model each transformation as a latent variable that we know some partitions are instances of its tails, but its context is still hidden. Notice that the directed acyclic graph that connects partitions is similar to a probabilistic graphical model [23] in which each transformation is a random variable. According to the conditional independency property of the network, a partition is independent of all other partitions given its parents. If a partition is at depth k, its parents are all adjacent partitions at depth $k - 1$. As a result, we assign the context of a transformation with a depth-k tail to be all adjacent depth-(k-1) partitions. Such assignment strategy is also utilized in learning the structure of probabilistic context-free grammars [178, 200]. We call this context assignment strategy *MinimaxParents*.

Algorithm 4: Minimax transformations

Input: A composite graph c and its summary into partitions $P = \{p_1, p_2, \ldots, p_m\}$.

Output: A set $\mathcal{T}$ of unique transformations with MinimaxParents context.

1: $\mathcal{T} = \emptyset$

2: From the progenitor ρ, perform a BFS on P and record:

3: L(i) ⊂ P = the set of partitions with shortest path to ρ is i

4: For each partition $p_m \in P$

5: k = depth of the partition $p_m \in L(k)$

6: τ = a new transformation

7: If k == 0

8: context(τ) = ρ

9: tail(τ) = L(0) \ {ρ}

10: Else

11: tail(τ) = p_m

12: context(τ) = {p_j | $p_j \in L(k-1)$ AND p_j is connected to p_m}

13: $\mathcal{T} \leftarrow \tau$

End

To find induced formative histories, we have to first extract all Minimax transformations, then replace each partition by its corresponding transformation.

Algorithm 5: Build formative histories

Input: A composite graph c, its summary P, and a set $\mathcal{T}$ of Minimax transformations across $\mathcal{D}$.

Output: A transformative history $\hbar$ of c with respect to P and $\mathcal{T}$.

1: From the progenitor ρ, perform a BFS on P and record:

2: L(i) ⊂ P = the set of partitions with shortest path to ρ is i

3: For each partition $p_k \in P$

4: v = a pointer to p_k such that P(v) = p_k

5: $\hbar \leftarrow$ add vertex v

6: For each vertex $v \in \hbar$

7: k = depth of partition P(v) ∈ L(k)

8: For each vertex $u \in \hbar$ such that P(u) ∈ L(k-1) and P(u) is connected to P(v)

9: $\hbar \leftarrow$ add edge $\overrightarrow{(v,u)}$

10: For each vertex $v \in \hbar$

11: v = a pointer to a transformation $\tau \in \mathcal{T}$ such that Tail(τ) = P(v) and Context(τ) = MinimaxParents(v)

12: Return $\hbar$

End

Figure 14 illustrates a model comprising of Minimax transformations (lower right corner) extracted from a summary of the super-composite, and corresponding formative histories (right side) of constituting composites.

4.6.3. Hill Climbing with Random Restart

Motivated by the search-and-score approach to learn the structure of Bayesian networks [23, 201-203], we develop a heuristic Hill Climbing with Random Restart algorithm to find the optimal summarization.

We view the optimization problem as a search problem in a state space. Each state is a candidate summary of the super-composite that induces a candidate model and formative histories of constituting composites. A score $\mathcal{F}(\mathcal{M} : \mathcal{D})$ can be evaluate using a fitness and a complexity measure (Section 4.5). Our goal is to find a state with the best score.

Hill climbing proceeds by computing scores of all neighbors of a state $\mathcal{H}^{i}$, and move to the neighbor state $\mathcal{H}^{i+1}$ with highest score. If there is no neighbor state with higher score than the current state, then *random restart* is triggered to randomly move to one of the neighbor states. In our problem settings, a neighbor state is obtained by contraction of one edge (or two adjacent vertices) of the current super-composite. Thus, the search process terminates when the super-composite has only one vertex left. In the pseudo-code for the algorithm, we write $\mathcal{F}(\mathbb{s})$ as a short notation for $\mathcal{F}(\mathcal{M} : \mathcal{H})$ where $\mathcal{M}$ is a model and $\mathcal{H}$ is a set of formative histories induced by current summary of the super-composite $\mathbb{s}$. Explicit elaboration of $\mathcal{F}(\mathbb{s})$ is provided when necessary to avoid ambiguity.

Algorithm 6: Hill Climbing with Random Restart

Input: A super-composite $\mathbb{s}$.

Output: A model $\mathcal{M}^*$ with highest score.

1: $\mathcal{F}^* = \mathcal{F}(\mathbb{s})$ // initial best score

2: $\mathbb{s}^* = \mathbb{s}$

3: $\mathcal{F}_{\text{current}} = \mathcal{F}(\mathbb{s})$ // current score

4: While number of vertices of $\mathbb{s} > 1$

5: $\mathcal{N} = \emptyset$ // a list of neighbor states

6: For each edge $e \in \mathbb{s}$

7: $\mathbb{s}' = \mathbb{s}$ with e being contracted to a single vertex

8: Extract a model $\mathcal{M}'$ and a set $\mathcal{H}'$ of formative histories from $\mathbb{s}'$

9: Evaluate $\mathcal{F}(\mathcal{M}' : \mathcal{H}')$ using a fitness and a complexity measure

10: $\mathcal{N}[e] = \mathcal{F}(\mathcal{M}' : \mathcal{H}')$

11: $e^* = \text{argmax}_e(\mathcal{N}[e])$

12: If $\mathcal{N}[e^*] < \mathcal{F}_{\text{current}}$ // need random restart

13: e^* = randomly select one of the edges of $\mathbb{s}$

14: Contract e* to a single vertex

15: $\mathcal{F}_{\text{current}} = \mathcal{F}(\mathbb{s})$

16: If $\mathcal{F}_{\text{current}} \geq \mathcal{F}^*$

17: $\mathcal{F}^* = \mathcal{F}_{\text{current}}$

18: $\mathbb{s}^* = \mathbb{s}$

19: $\mathcal{M}^*$ = the model induced from $\mathbb{s}^*$

20: Return $\mathcal{M}^*$

End

In line 12, random restart is only triggered when all neighbor states yield *strictly* smaller scores than the current state. As a result, we encourage side way move with equal scores to systematically escape plateaux.

As an implementation detail, graph and subgraph comparison are performed using canonical labels. We implement the graph canonical labeling method presented by Kuramochi et.al. [195]. Although computing graph canonical labels is an expensive operation, it helps avoid doing multiple graph isomorphism as well as subgraph isomorphism. Our experiment shows that the graph canonical labeling algorithm scales well to real-world datasets.

4.7. Prediction Problem Formulation

Although prediction is in general easier than induction, it is nevertheless a nontrivial problem. Comparing to sequence predictors that deal with a linear space of subsequences (i.e. there are $\frac{n(n+1)}{2}$ subsequences of a length-n sequence), graph predictors deal with a more complicated problem due to the exponential space of subgraphs. Furthermore, subgraphs have no natural order compared to the natural order of subsequences.

Our goal is designing a practical predictor for gETS models that has several properties: (i) it can find the optimal history in the asymptotic case, and (ii) prediction quality can be controlled to fit in a time constraint.

Definition: A test composite graph c is *complete* w.r.t. a gETS model $\mathcal{M}$ if and only if it can be fully reconstructed using the model.

$$\exists\ \hbar = (\rho, \tau_1, \tau_2, \ldots, \tau_k) \text{ such that } c = \rho \blacktriangleleft \tau_1 \blacktriangleleft \tau_2 \blacktriangleleft \ldots \blacktriangleleft \tau_k.$$

Definition: A *partial history* of a test composite graph c w.r.t. a gETS model $\mathcal{M}$ is a transformative history $\hbar$ that partially cover the composite graph.

$$\hbar = (\rho, \tau_1, \tau_2, \ldots, \tau_k) \text{ such that } \rho \blacktriangleleft \tau_1 \blacktriangleleft \tau_2 \blacktriangleleft \ldots \blacktriangleleft \tau_k \subset c$$

In this work, we develop a predictor that ranks test composites based on scores of partial histories. To be practical, the predictor is bounded to a time constraint governed by an adjustable parameter. Furthermore, in the asymptotic case, it is guarantee that the predictor will produce the best complete histories.

A natural score for partial histories is the amount of coverage of the test composites. However, we use probability and typicality for the fitness score and learnt models have probability (or typicality) value associated to transformations, so it is desirable to have scores that also incorporate probability (or typicality) of the partial histories.

Definition: A *coverage score* of a partial transformative history $\hbar$ of a composite graph c is the fraction of vertices of c that $\hbar$ is able to cover.

$$Cov(\hbar \mid c) = \frac{\#(vertices\ that\ \hbar\ covers)}{\#(vertices\ of\ c)}$$

To devise a score that incorporates both coverage and typicality of a partial history, we compute typicality of the partial history, and penalize it *linearly* based on the amount of coverage. The penalty is linear to align with summation of typicality of transformations.

Definition: A *typicality score* of a partial transformative history $\hbar$ of a composite graph c is the computed typicality of $\hbar$ penalized by the amount of coverage $\hbar$ imposes on c.

$$FR(\hbar) = Cov(\hbar)^{sign} \times \sum_{i=1}^{k} FR(\tau_i)$$

where:

- $\hbar$ = (ρ, τ_1, τ_2,…, τ_m) and {τ_1, τ_2,…, τ_k | k < m} is the set of *unique* transformations in $\hbar$.
- $FR(\tau_i) = \frac{1}{|\mathcal{D}^+|} support(\tau_i|\mathcal{D}^+) - \frac{1}{|\mathcal{D}^-|} support(\tau_i|\mathcal{D}^-)$ is the family resemblance typicality of transformation τ_i w.r.t. the training dataset.
- $sign = \begin{cases} 1 \ \ if \ FR \geq 0 \\ -1 \ \ if \ FR < 0 \end{cases}$

Using the *sign* switch, the penalty always reduces typicality in both positive and negative cases.

To devise a score that incorporates both coverage and probability of a partial history, we compute probability of the partial history, and penalize it *exponentially* based on the amount of coverage. The penalty is exponential to align with product of probability of transformations. Instead of imposing an exponential penalty, we equally impose a linear penalty on the logarithm of the probability.

Definition: A *log-likelihood score* of a partial transformative history $\hbar$ of a composite graph c is the computed log-likelihood of $\hbar$ penalized by the amount of coverage $\hbar$ imposes on c.

$$L(\hbar) = Cov(\hbar)^{-1} \times \sum_{i=1}^{m} L(\tau_i)$$

where:

- $\hbar = (\rho, \tau_1, \tau_2, \ldots, \tau_m)$.
- $L(\tau_i) = log\ (p(\tau_i))$ is the log probability of transformation τ_i.

The coverage penalty factor is always reciprocal because logarithm of a probability (in range [0, 1]) is always a negative number.

4.8. Prediction Algorithm

We approach the prediction formulation with Simulated Annealing (SA) algorithm. SA fits our goal because a time constraint can be enforced by setting the initial temperature T_0, and in asymptotic case $T_0 \rightarrow +\infty$ the algorithm reverts to a brute force search that is guaranteed to find the best complete histories.

Algorithm 7: Simulated annealing reconstruction

Input: A composite graph c and a model $\mathcal{M} = (\rho, \{ \tau_1, \tau_2, \ldots, \tau_n\})$.

Output: The highest score and complete histories.

1: T = 100 × #(number of vertices of c)

2: *score* = $-\infty$ // current score

3: *bestscore* = $-\infty$ // best overall score

4: Q = $\emptyset$ // a list of partial histories

5: S = matchTranformation(ρ, *c*, $\emptyset$)

6: Q = Q $\cup$ S

7: While not Q.isEmpty()

8: *cs* = take out a randomly selected element of Q // current state

9: *ap* = acceptanceProbability(*cs*) = $\begin{cases} 1 \ if \ cs.\text{score} > score \\ e^{(cs.\text{score}-score-1)/T} \ otherwise \end{cases}$

10: *dice* = a random number in [0, 1]

11: If *dice* $\leq$ *ap* // accept this move

12: If *ap* < 1

13: T = T × 0.95 // exponential cooling

14: *score* = *cs*.score

15: *bestscore* = *bestscore* < *score* ? *score* : *bestscore*

16: If *state* is complete

17: Propagate a history $\hbar$ from *cs*

18: Output $\hbar$

19: *score* = $-\infty$ // reset the score to encourage other complete histories

20: For i = 1 : n

21: S = matchTranformation(τ_i, *c*, *cs*)

22: For each s $\in$ S

23: s.score = Cov(s) or FR(s) or L(s) // Section 4.7

24: s.parent = *cs*

25: Q = Q ∪ S

26: Else // the candidate move is rejected

27: For each state s ∈ Q

28: If acceptanceProbability(s) < 1 / 10,000

29: Remove s from Q

30: Return *bestscore*

End

Line 1 initializes the temperature by empirical experiment. We keep track of the current score and best score in the algorithm. The algorithm maintains a list Q of partial histories and keeps iterating this list until termination. Lines 5 and 21 invoke the matchTransformation(transformation τ, composite graph c, state s) function that find all matching of τ in c so that context(τ) matches into the region current state s has covered c, while tail(τ) matches into free region $\{c \setminus s\}$. We use Boost graph library implementation of Cordella's subgraph isomorphism algorithm [204, 205]. To improve matching efficiency, we constrain the boundary between context and tail of τ to match with the boundary between s and$\{c \setminus s\}$. Such constraint aligns with the MinimaxParents strategy in extracting transformations so that the extracted context is only one hop from the corresponding tail.

Line 9 computes an acceptance probability of the current state. Minus 1 is experimentally added to the exponent to prevent the system from wandering in a plateau. Such plateau is surprisingly large with typicality measure. Line 13 fixes an exponential cooling whenever a bad move is accepted. Line 15 retains the best score so far. Line 16 to

18 reconstruct a complete history using the back-propagation chain of parents established in line 24. Line 19 resets the current score because score of a complete history is usually high, so we lower it to encourage convergence to other complete states. Line 20 to 25 match all transformations of the provided model to the current state, then assign appropriate scores and parent chain.

Finally, line 27 to 29 remove states that have acceptance probability less than a threshold. We experimentally set the threshold to 1 / 10,000. This extra steps is meant to encourage faster convergence by eliminating bad states with too low a chance of being accepted. The threshold is chosen very small, so that to not affect the convergence of SA to the highest score state.

4.9. The Machine

As a conclusion to the long method elaboration, we collectively call the graphical representation of ETS, the model induction algorithm, and the prediction algorithm a *machine*. Such a combination is capable of doing the job of a machine learning system, from training a model to giving prediction to test examples.

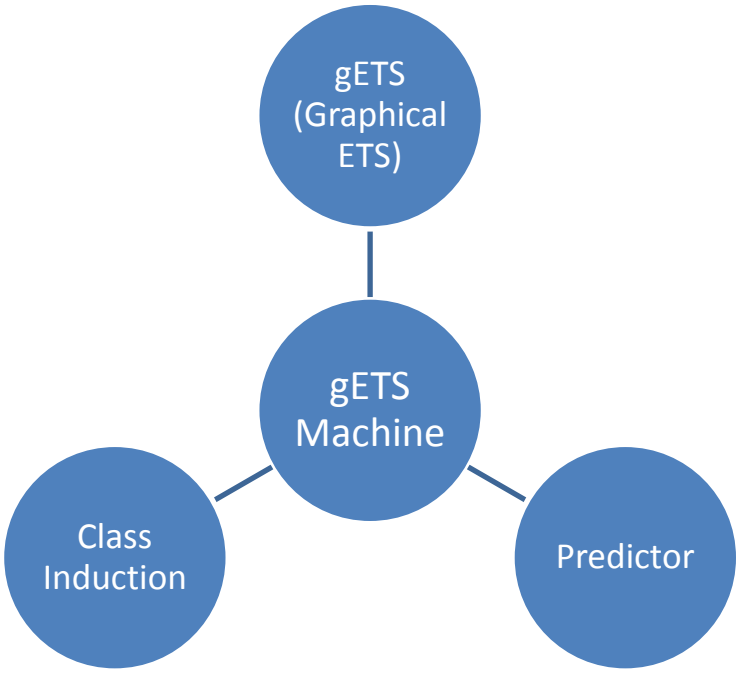


Figure 15: The gETS Machine as a combination of a representation formalism, an induction algorithm, and a prediction algorithm.

CHAPTER 5

CLASSIFYING HUMAN ACTIONS IN STILL IMAGES USING NORMALIZED VIEW 2D POSE GRAPHS

5.1. Introduction

Recognizing human actions has been an active area in computer vision and machine learning. Methods based on spatio-temporal information (i.e. a video segment, or a sequence of images) has been successfully developed [206]. However, as a large body of visual information is kept in static images (e.g. Facebook, Flicks…), and from the fact that it is relatively easy for human beings to discern actions from a single shot, recent interest has turned to recognizing human actions from still images [207]. Despite attractive applications in many domains such as security, elderly care, photo tagging, satellite image analysis etc, the research remains difficult and there were only 34 publications in the last decade [207]. Methods in this area utilize a variety of high-level features such as body information [208], body parts [209], human-object interaction [210], scene information [211], as well as low-level features such as DSIFT [212], HOG [213], GIST [214], etc.

In this work, we consider action recognition includes two subtasks: (i) detection of human figures from still images, and (ii) classifying the action being performed. We specifically solve the second task using a novel structured-based method that infer a transformation system of graphs encoding normalized 2D views of human poses. The inspiration to our approach is the articulated pose representation and action classification using 2.5D graphs of Yao et al [215]. However, our method only extracts pose features from view-independent 3D skeletons, and does not include any 2D appearance features. We use gETS Machine presented in previous chapters to learn model of action classes and subsequently classify test examples using learnt models instead of simple prototype-based graph matching [215].

5.2. Data Collection

In this work, we use PASCAL VOC 2011's action classification dataset. First, Maji 's 2D keypoint annotation of the VOC's human action dataset on Amazon Mechanical Turk [216, 217] are extracted, then Ramakrishna's method [218] to reconstruct 3D skeletons from 2D keypoints is invoked. In this experiment, we only keep complete skeletons that have all of following keypoints: head, left/right shoulders, left/right elbows, left/right wrists, left/write hips, left/right knees, and left/right ankles. After that, we filter out action classes that have too few complete annotations so that the learnt models could be statistical significant. The result is four action classes: walking, running, jumping, and riding bike. Table 5 summarizes number of complete-skeleton examples of the four action classes according to VOC's train and validation splitting.

Table 5: Data distribution of four action classes according to VOC's splitting.

	Walking	**Running**	**Jumping**	**Riding Bike**
Train	56	69	65	43
Validation	54	60	62	37

Figure 15 gives illustration of human figures from four action classes, together with 2D skeletal annotation. Notice that it is easy for human beings to discern between difference action classes using both static pose and appearance.


3842

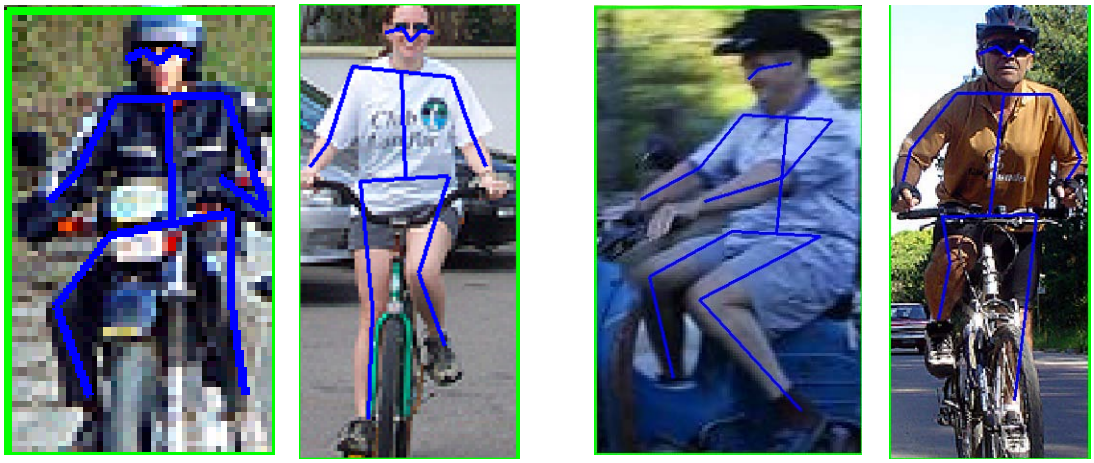


Figure 16: Samples from four action classes together with 2D skeletal annotation. From top to bottom, row 1 illustrates Walking class, row 2 illustrates Running class, row 3 illustrates Jumping class, and row 4 illustrates Riding Bike class. From left to right, the first 2 images belong to VOC's training set, and the last 2 images belong to VOC's validation set.

To infer 3D skeletons from 2D joint annotation, we notice that 2D annotation for pelvis and neck is missing. We interpolate those points by first computing the geometric midpoints between the shoulders and the hips, then move the shoulder midpoint away from the hip midpoint and also the hip midpoint towards the shoulder midpoint by 5% of the original length between the shoulder midpoint and the hip midpoint. Algorithm 8 provides a concise pseudo code for the procedure.

Algorithm 8: Interpolate Neck and Pelvis

Input: left shoulder (L_Shoulder), right shoulder (R_Shoulder), left hip (L_Hip), right hip (R_Hip)

1: shoulder_midpoint = (L_Shoulder + R_Shoulder) / 2;

2: hip_midpoint = (L_Hip + R_Hip) / 2;

3: Neck = shoulder_midpoint + (shoulder_midpoint - hip_midpoint) / 20;

4: Pelvis = hip_midpoint + (shoulder_midpoint - hip_midpoint) / 20;

End.

Figure 16 visualizes 3D skeletons of several human figures from four action classes. A couple of comments are: (1) It is not so intuitive even for human beings to discern action classes based on 3D skeletons alone; (2) The inferred 3D pose contains errors, i.e. they are different from the true pose. Among the four action classes, Walking and Running 3D skeletons contains relatively small errors, while Jumping and Riding Bike 3D skeletons may contain a large amount of errors that deviates the skeletons significantly from the true pose. In the example, both 3D skeletons in the third row deviates from the true action, and the riding bike skeleton in column 2 of row 4 looks like a walking person.

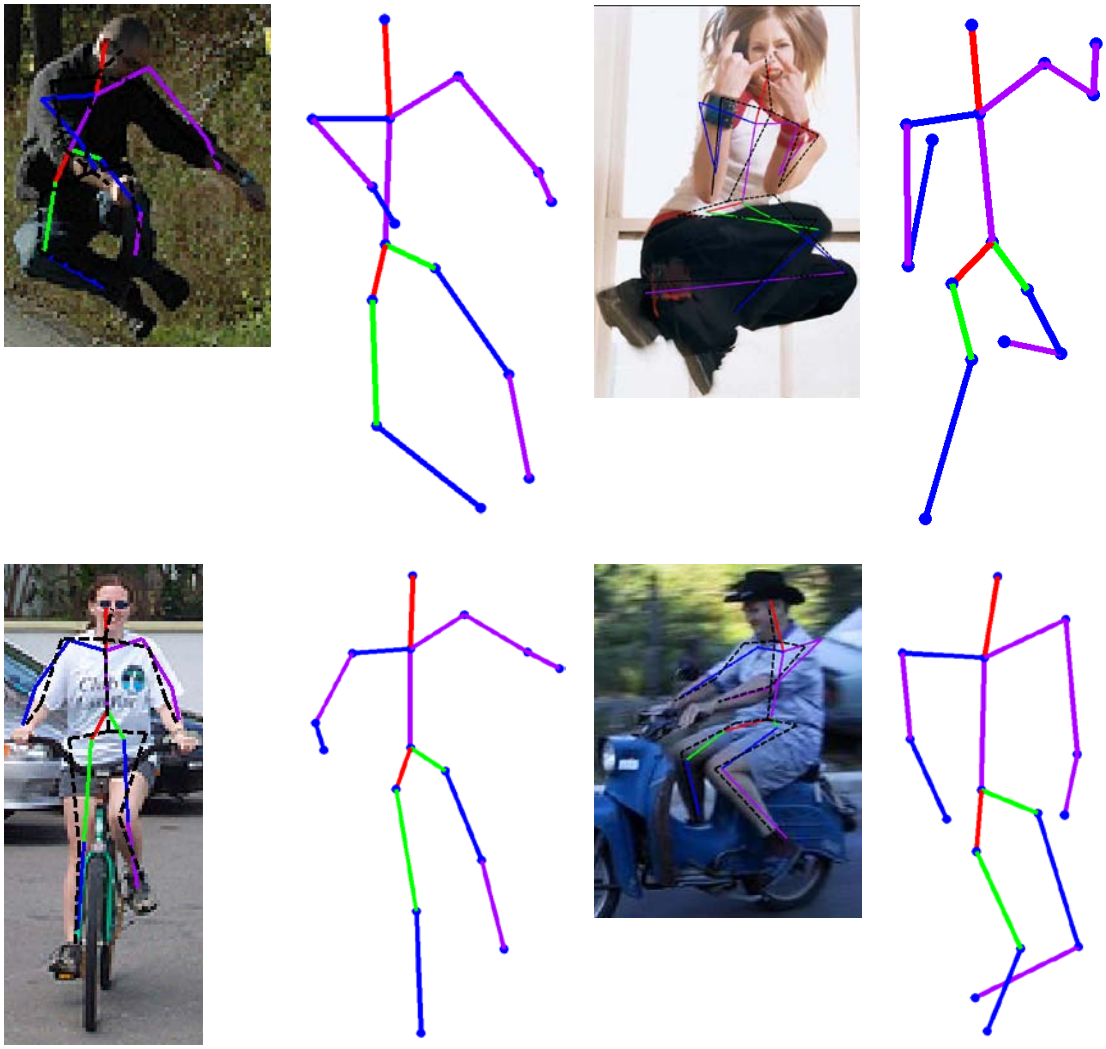


Figure 17: 3D pose inferred from 2D joint annotation. From top to bottom, row 1 illustrates Walking class, row 2 illustrates Running class, row 3 illustrates Jumping class, and row 4 illustrates Riding Bike class. From left to right, column 2 contains 3D skeletons of human figures in column 1, and column 4 contains 3D skeletons of human figures in column 3. In column 1 and 3, human figures are annotated with 2D skeletons in black dashed lines, and the best fit 3D skeletons are projected onto the 2D plane in colored solid lines. In column 2 and 4, 3D skeletons are viewed from a vantage point, approximately 45^{o} degree on the front-right.

5.3. Feature Extraction

The motivation for this wok is driven by our intuition that a human can look at a 2D projection of a performer and correctly identify the associated action. For example, by looking at a side projection (or a front projection) of a performer, human beings can classify whether the performer is walking, running, jumping, or riding bike easily. However, when a 2D projection has noise, as shown in figure 17, it is usually context cue that help separate between action classes. For example, in Figure 16, a riding bike person with noisy 3D pose might be mistaken with a walking or a running person; but, if a bike detector signals the presence of a bike in the scene, then it is relatively easy to classify the action as riding bike.

In this work, we do not utilize any object detector, and thus do not use context cue in classifying actions. Furthermore, the primitive feature extraction process introduces noise to 3D pose as an inevitable addition since reconstructing 3D pose from 2D skeletons is a difficult task [218]. Nevertheless, noisy input raises the bar to our method to cope with real data, as contrast to clean databases [219].

We extract features as angles at the joints of normalized 2D projections of noisy 3D poses. We use two normalized projections: one is the front view projection (viewing towards the face), and second is the right-side view projection (viewing towards the right shoulder). For each 2D projection figure, angles at joints are measured, rounded, and discretized into a chain of primitive angles. In this work, a 10^{o} degree angle is used as the primitive angle. Furthermore, each angle is associated with an additional orientation feature that identifies whether the corresponding body part is in the front or back with

respect to the spine projection in the right-side view, or to the left or right with respect to the spine projection in the front view. Following joints (together with their associated body parts) help extract angle features: left/right shoulders (left/right upper arms), left/right elbows (left/right lower arms), middle hip (pelvis), left/right hips (left/right upper legs), and left/right knees (left/right lower legs). Algorithm 9 demonstrates the feature extraction process, Figure 17 illustrates normalized projections, and Figure 18 illustrates the extraction of angle features.

Algorithm 9: Extract normalized 2D features

Input: 3D joint coordinates J_{3D}

Output: A map M from each joint to a pair (orientation, angle)

1: P_{front} = Fit a plane through 4 points (neck, pelvis, left shoulder, right shoulder)

2: J_{front} = J_{3D} projected on P_{front}

3: For each 2D joint-segment pair (j_{front}, s_{2D}) ∈ J_{front}

4: a = angle between s_{2D} and the $spine_{2D}$ ∈ J_{front}

5: o = [right/left] orientation of s_{2D} w.r.t. $spine_{2D}$

6: M[j_{front}] = (a, o)

7: P_{side} = The plane going through the $spine_{3D}$ and perpendicular to P_{front}

8: J_{side} = J_{3D} projected on P_{side}

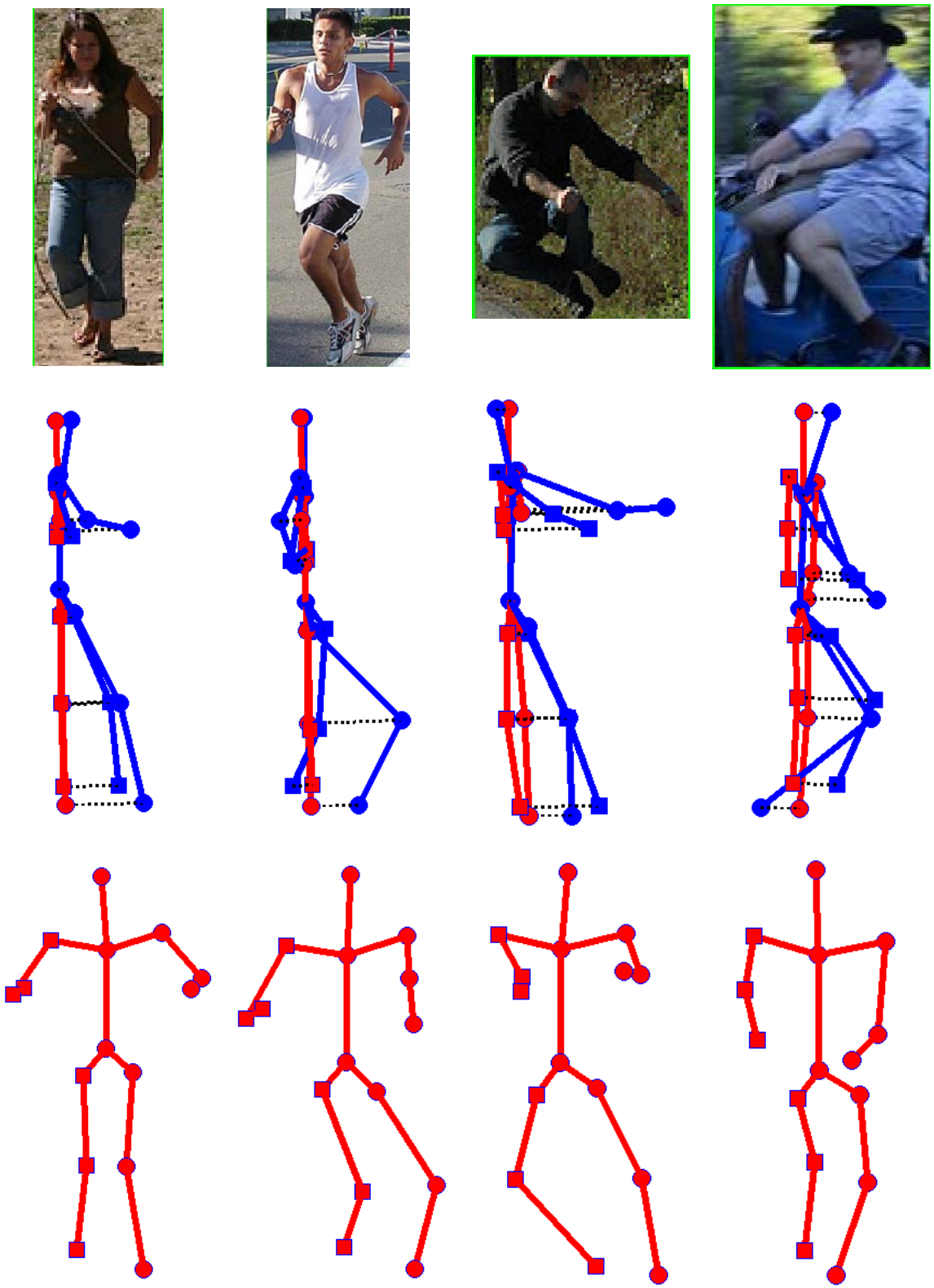
9: For each 2D joint-segment pair (s_{2D}, j_{side}) ∈ J_{side}

10: a = angle between s_{2D} and the $spine_{2D}$ ∈ J_{side}

11: o = [front/back] orientation of s_{2D} w.r.t. $spine_{2D}$

12: M[j_{side}] = (a, o)

End.



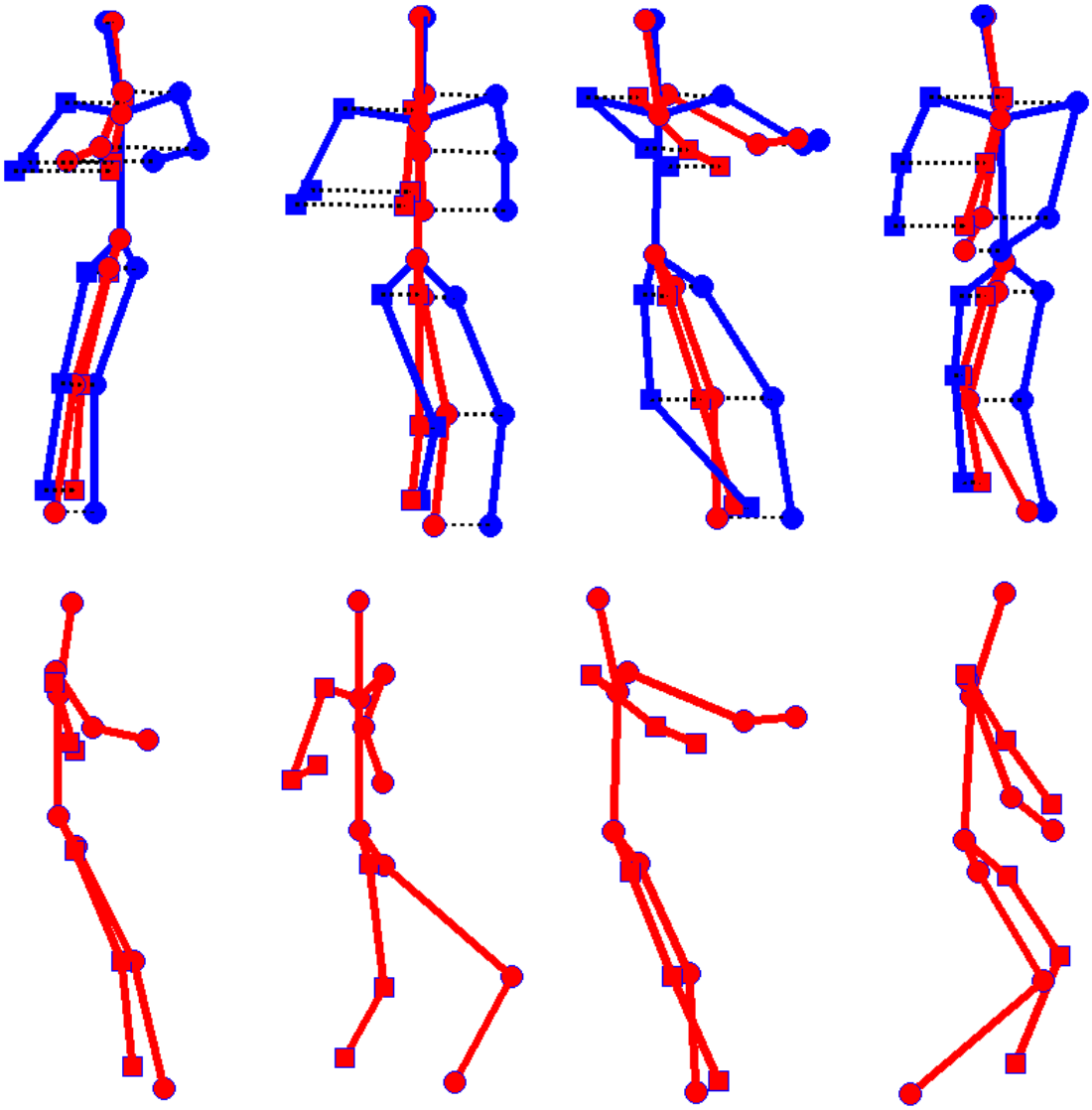


Figure 18: Normalized-view projection of 3D poses to: (i) a front view plane, and (ii) a right-side view plane. From left to right: column 1 contains a walking person, column 2 contains a running person, column 3 contains a jumping person, and column 4 contains a riding-bike person. From top to bottom: row 1 contains the original images, row 2 contains 3D projection on the front-view plane at a vantage view point, row 3 contains 2D projection on the front-view plane, row 4 contains 3D projection on the right-side-view plane at a vantage view point, and row 5 contains 2D projection on the right-side-view plane. 3D skeletons are drawn in blue, while 2D skeletons are drawn in red. Joints on the right side of the body (i.e. right shoulder, right elbow, right wrist, right hip, right knee, and right ankle) are shaped by squares, while other joints are shaped by circles.

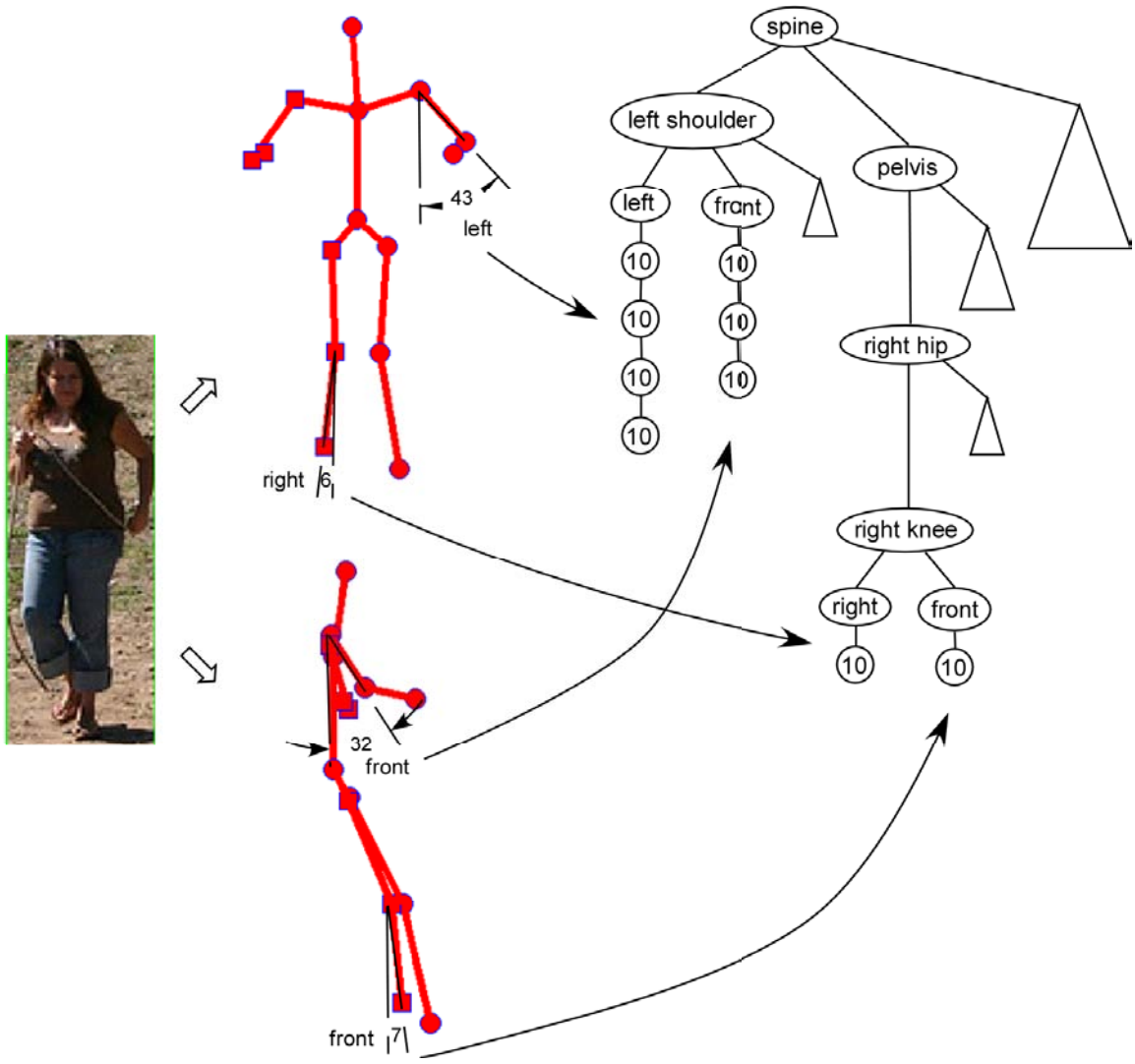

spine
left shoulder
pelvis
left
front
10
10
10
10
10
10
10
right hip
right knee
right
front
10
10
43
left
right
6
32
front
front
7

Figure 19: Extracting angle features from normalized 2D projections. On the left is the original human figure. The middle top skeleton is 2D projection on the front-view plane, and the middle bottom skeleton is 2D projection on the right-side-view plane. Angle features at left shoulder and right knee are measured and extracted in this example. On the front-view plane, the left shoulder is at 43^o degree to the left of the projected spine that transforms into a chain of 4×10^o degree vertices, and the right knee is at 6^o degree to the right of the projected spine that transforms into 1×10^o degree vertex. On the right-side-view plane, the left shoulder is at 32^o degree to the front of the projected spine that transforms into a chain of 3×10^o degree vertices, and the right knee is at 7^o degree to the front of the projected spine that transforms into 1×10^o degree vertex. On the right hand side is a partial composite graph with feature angles at left shoulder and right knee. In the composite structure, triangles represent subgraphs not expanded in this example. Note that in the original human figure, the left shoulder swings to the back, as opposes to the front as in the inferred skeleton. This type of systematic noise is common in real datasets.

After having angle features, we convert each example into a composite graph by creating a vertex for each joint and discretizing each angle into a chain of 10^o degree vertices. Algorithm 10 demonstrates the composite graph construction process. Figure 19 gives an example of a partial composite graph construction process, and Figure 20 provides an example of a complete composite graph.

Algorithm 10: Construct composites

Input: Map M from each joint to a pair (orientation, angle)

Output: A composite graph *c* as a feature graph

1: *c* ← new vertex("spine")
2: For each joint j ∈ M
3: *c* ← new vertex(j.name)
4: If (j == left/right shoulder) || (j == pelvis)
5: *c* ← add edge (j, "spine")
6: ElseIf (j == left/right elbow)
7: *c* ← add edge (j, "left/right shoulder")
8: ElseIf (j == left/right hip)
9: *c* ← add edge (j, "pelvis")
10: ElseIf (j == left/right knee)
11: *c* ← add edge (j, "left/right hip")
12: EndIf
13: j_{orient} = *c* ← new vertex(M[j].orientation)
14: *c* ← add edge (j, j_{orient})

15: n = round(M[j].angle / 10)

16: For i = 1 : n

17: j_{angle} = c ← new vertex("10")

18: c ← add edge (j_{angle}, j_{orient})

19: j_{orient} = j_{angle}

End.

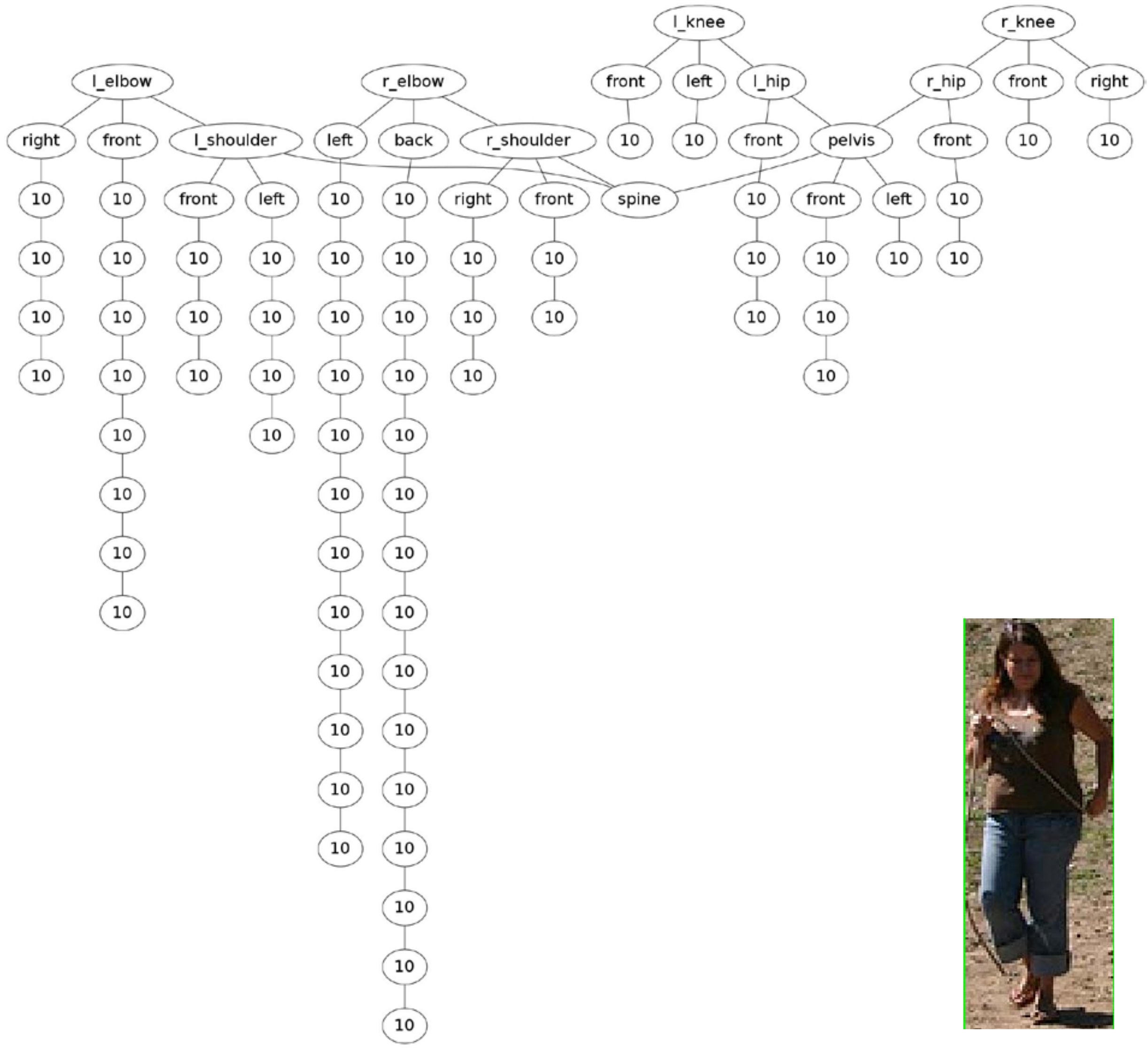


Figure 20: A complete composite graph representation of the walking person example in Figure 18. "r_" and "l_" in vertex labels denote right and left respectively. There is a skeletal structure in the composite graph that connect: spine→ (left/right shoulders, pelvis), left/right shoulders→ left/right elbows, pelvis → left/right hips, and left/rig ht hips → left/right knees. Each joint has two orientations: left/right (front view) and front/back (side view). Each orientation primitive feature is connected to a chain of 10^o degree angles that encodes the magnitude of the orientation.

5.4. Datasets Distribution

We follow PASCAL VOC's action classification task [220] dataset splitting. Specifically, training examples are used for training, and validation examples are used for testing. In training, examples from other classes are used as negative examples, and the same strategy is followed in testing. For example:

- In training a model for the Walking class, training data from VOC Walking class are used as positive examples, and training data from VOC Running + Jumping + Riding Bike classes are used as negative examples.
- In testing a learnt model of the Walking class, validation data from VOC Walking class are used as positive examples, and validation data from VOC Running + Jumping + RidingBike classes are used as negative examples.

Other classes (i.e. Running, Jumping, and Riding Bike) follow the same data splitting strategy. Table 6 summarizes training and testing data for four action classes.

Table 6: Training and testing datasets distribution of four action classes.

	Walking	**Running**	**Jumping**	**Riding Bike**
Train Positives	56	69	65	43
Train Negatives	177	164	168	190
Test Positives	54	60	62	37
Test Negatives	159	153	151	176

According to this strategy that reuses data from other classes as negative examples, we note that it effectively makes all classes having the same training and testing data, only

positive/negative labeling is swapped. As a consequence, all classes have the same super-composite graph when the same primitive progenitor is used. The trimmed VOC 2011 action dataset produces 14 primitive progenitors: *10, front, left/right elbow, left/right hip, left/right knee, left/right shoulder, left, pelvis, right,* and *spine*. We empirically filter out progenitors that result in total number of vertices across all layers greater than 50,000. Those are noisy progenitors as a result of noisy input. This heuristic step removes progenitor "10" (total size 1,558,942), progenitor "front" (total size 148,605), progenitor "left" (total size 76,958), and progenitor "right" (total size 79,995). All other progenitors yield identical super-composites with total size 21,306, and thus learning with any of them should give the same result. Progenitor "left elbow" is the first one in the alphabetically sorted list and is selected as the representative progenitor to perform induction. Figure 20 illustrates a super-composite with "left elbow" as the progenitor.

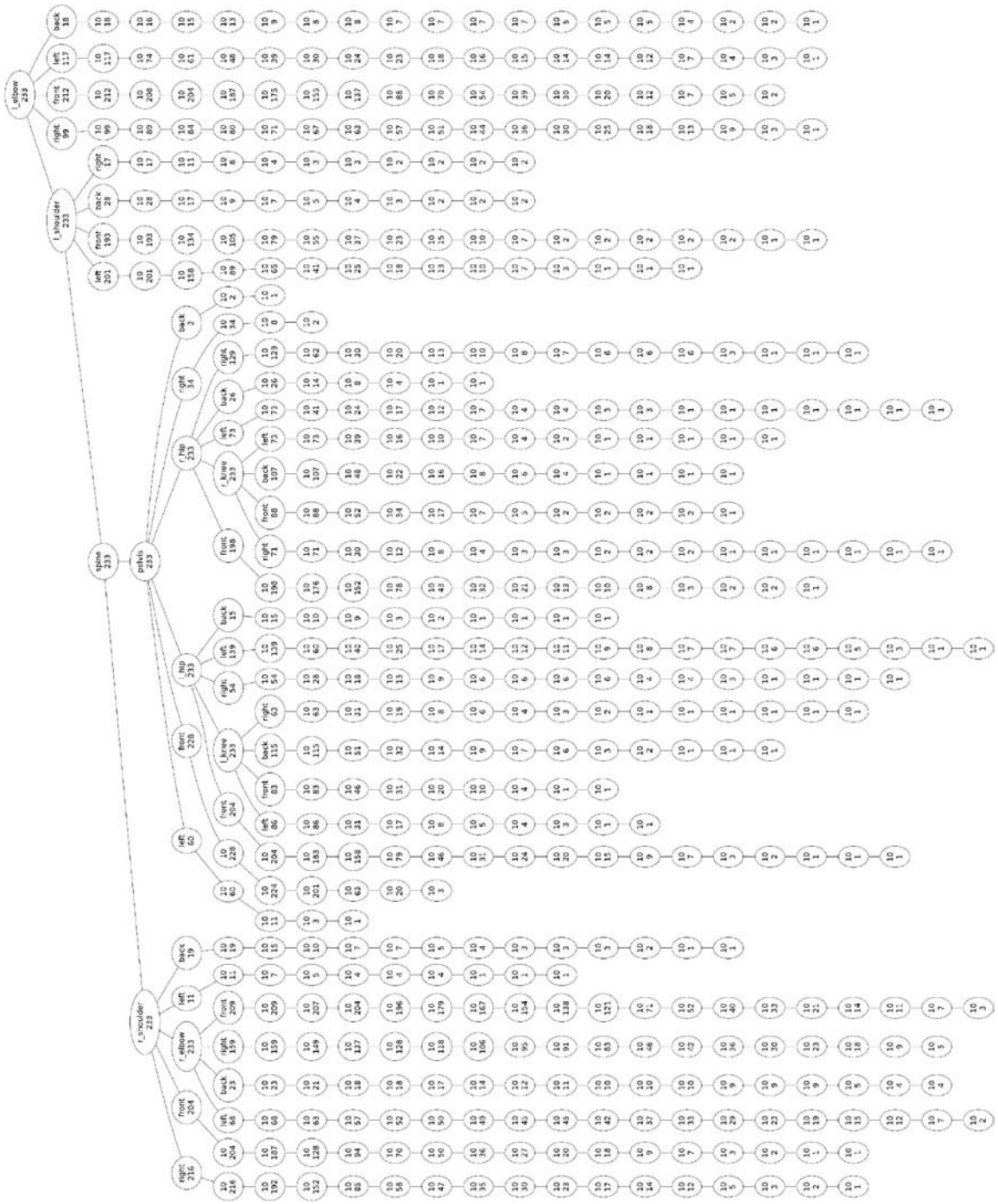


Figure 21: A super-composite of Walking class with *left elbow* as the progenitor. This super-composite has 511 vertices, 510 edges, and 233 layers. In each vertex, the first line displays primitive label, and the second line displays number of layers this primitive feature projects through. This super-composite is equivalent to other super-composites with other progenitors of the Walking class. Moreover, this super-composite is also equivalent to super-composites of other classes: Running, Jumping, and Riding Bike due to distribution of training datasets that reuses examples of other classes as negative training examples.

5.5. Performance

We compare our system's performance with performance of other systems on Pascal VOC 2011's action classification task. We emphasize that the comparison is relative, since our system trains on about 1/3 of training set, and test on about 1/3 of validation set, while other systems train on the entire training set plus validation set, and test on the entire test set. Furthermore, our system use semi-annotated data (the 2D joints are annotated) for training and testing, while other systems use VOC's 2D bounding box annotation. Nevertheless, this comparison gives a relative benchmark on how our system performs.

We run all training and testing phases on a single-core CPU clocking at 3GHz. Given the same dataset is recycled for training different classes, training time is heavily affected by the choice of measurements comprising the objective function. On average, training a model with complexity measure takes 2 to 4 hours, a model with typicality measure takes 9 to 11 hours, and a model with probability measure takes 8 to 10 hours. Combining

complexity with the other fitness measures cuts down training time to 2 to 6 hours. On the other hand, test time depends on quality of the learnt model and complexity of the test composite graph. Reconstruction time for a test example averages over less than 2 seconds at the low end, and 5 to 30 minutes at the high end. Our experiment shows that models with too many big graph transformations, or too many small graph transformations tends to take longer time to reconstruct a test composite graph. Explanation for the earlier case is complexity of graph matching, while reason for the latter case is exponential space of transformation combination.

Table 7 and Table 8 contains performance comparison between our methods and others. While Table 7 uses *average precision*, a modern performance measure based on precision at intervals enhanced with interpolation [220], Table 8 uses the traditional prevision and F-measure. Formulas of these performance measures are provided in Appendix B.

In the average precision test (Table 7), the best results of our system out-perform the state-of-the-art method on Jumping class, and close to the 2.5D graph method on Walking class. Our system falls behind in Running and Riding Bike classes. Between different types of measures, typicality usually outperforms probability measure, and probability outperforms complexity measure. Contrary to our expectation, regularization has not shown its effect yet. It might be due to the correlation between complexity with the two fitness measures.

In the complete history test (Table 8), our system gives high precision and F-measure on Walking class, and insignificant results on other classes.

Table 7: Average precision of state-of-the-art (SOTA) methods on Pascal VOC 2011 challenge [221], Yao's 2.5D graph matching method [215], and our system on probability (Prob.) measure, typicality (Typ.) measure, complexity (Clx.) measure, and combination measure. Only the best results are listed.

	SOTA	2.5D Pose	2.5D App.	2.5D Full	Prob.	Typ.	Clx.	Prob. − Clx.	Typ. − Clx.
Walking	65.9	52.8	59.7	62.1	49.9	**51.6**	39.6	48.6	51.1
Running	87.9	79.4	83.0	86.8	**41.9**	41.7	41.1	41.9	41.9
Jumping	71.6	64.6	68.9	72.4	60.2	**77.5**	30.0	60.0	77.5
Riding Bike	90.0	81.4	86.6	89.0	29.4	31.4	19.3	29.6	**34.1**

Table 8: Precision and F-measure of *complete* history test. Precision = N/A when TP = FP = 0. F-measure = N/A when precision = N/A or precision = recall = 0.

	Precision					F-measure				
	Prob.	Typ.	Clx.	Prob. − Clx.	Typ. − Clx.	Prob.	Typ.	Clx.	Prob. − Clx.	Typ. − Clx.
Walking	66.7	66.7	N/A	63.6	69.6	0.89	0.89	N/A	0.83	0.89
Running	29.2	31.6	N/A	31.8	31.6	0.35	0.3	N/A	0.35	0.3
Jumping	N/A	0	N/A	N/A	0	N/A	N/A	N/A	N/A	N/A
Riding Bike	4.2	4.7	N/A	4.5	4.7	0.08	0.08	N/A	0.08	0.08

CONCLUSION

In this work, we have presented a structured induction algorithm that is able to learn and feature graphs and an accompanying prediction algorithm that ranks test input in restricted time complexity. We demonstrated that the system produces on par performance with both graph-based and state-of-the-art methods on a difficult task, a real dataset, and with noisy input. We test the formulation on a variety of measures, including family resemblance typicality, likelihood probability, and minimum description length. To the best of our knowledge, this is the first comprehensive system dealing with graph-based pattern recognition. The system is comprised of three elements: a graphical encoder, a class induction algorithm, and a rank-based predictor. The system has a wide range of application but its specialty is best utilized in areas where statistical learning is required on graphical data, such as: social networks, biological networks, traffic networks, image segmentation, image understanding, structured information retrieval, textual analysis, etc.

The horizon is wide open for future research with gETSM. Regarding the machine, investigation on how different ways to build the super-composites affect induction and prediction outcomes is significantly important. A dynamic programming style predictor would help increase prediction quality substantially. Optimization techniques that guarantee global convergence are also of interest, and closed-form optimal solutions are

ideal. Regarding the application, one possibility is incorporating an object detector for contextual features. Another smart encoding of feature graphs that reduces representation complexity would also reduce time complexity.

Appendix A

Induction Performance

This section provides response of objective functions as well as statistics of induced models. All models are trained with a single-threaded program, on a single CPU at 3GHz and 4GB of RAM memory.

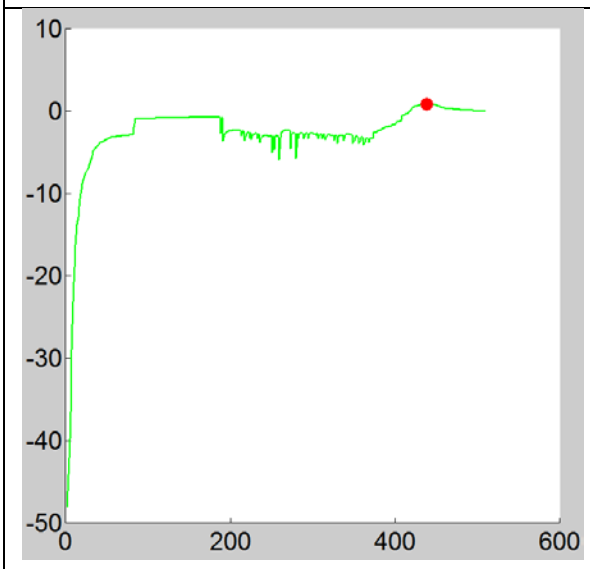
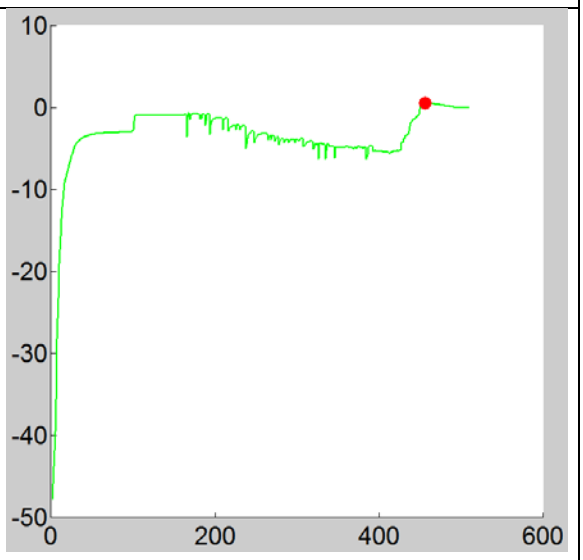
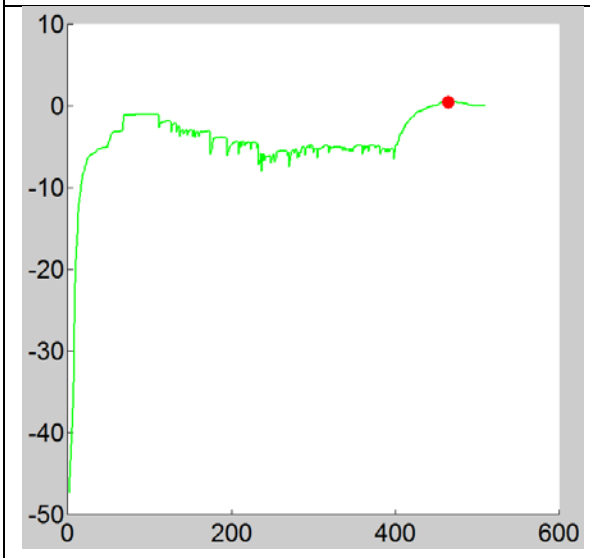
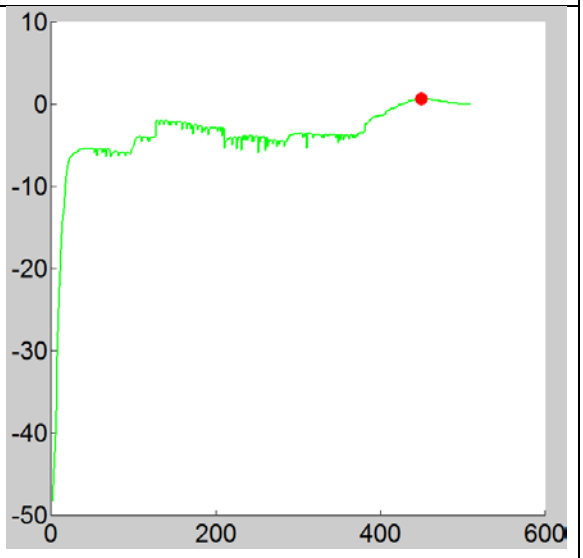
A.1. Measurement Ranges

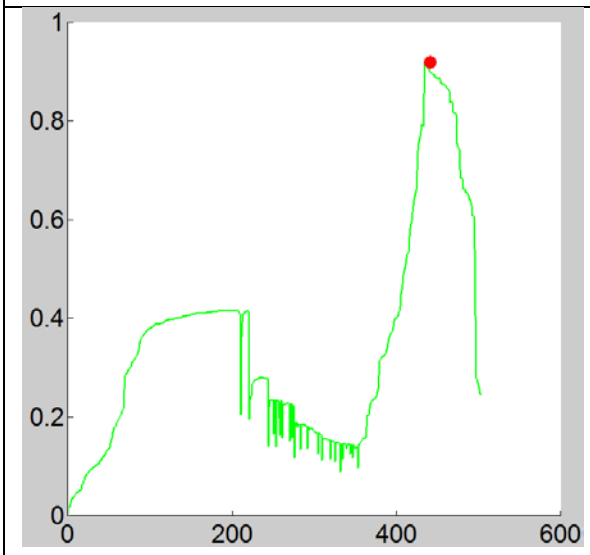
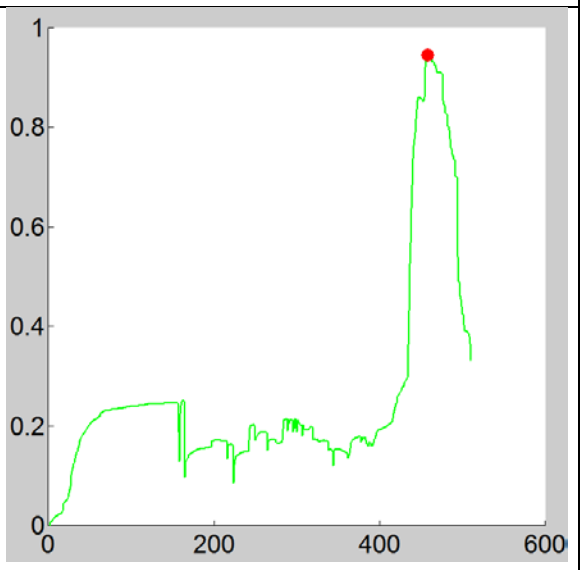
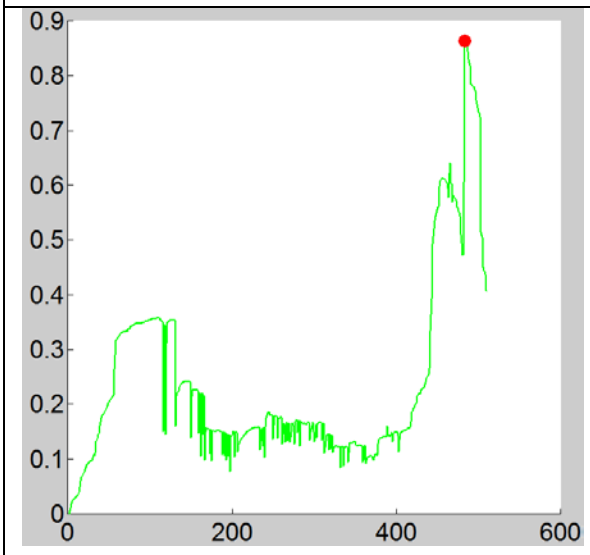
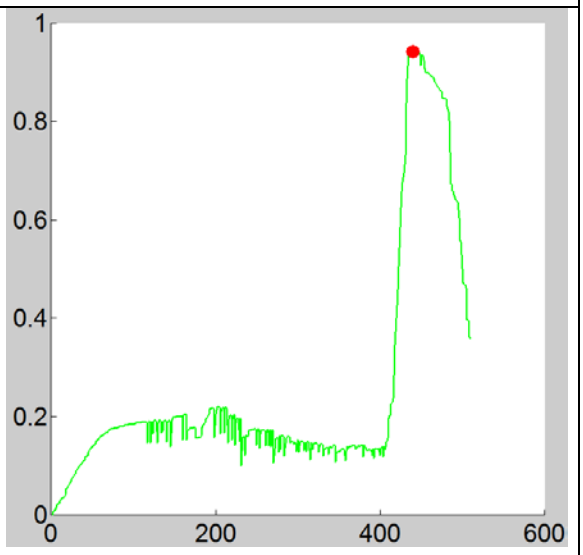
Based on empirical training on individual measure, we collect empirical ranges of measurements as follow:

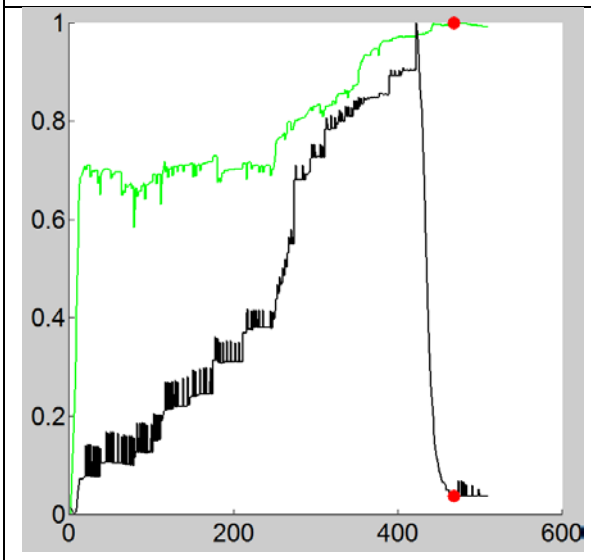
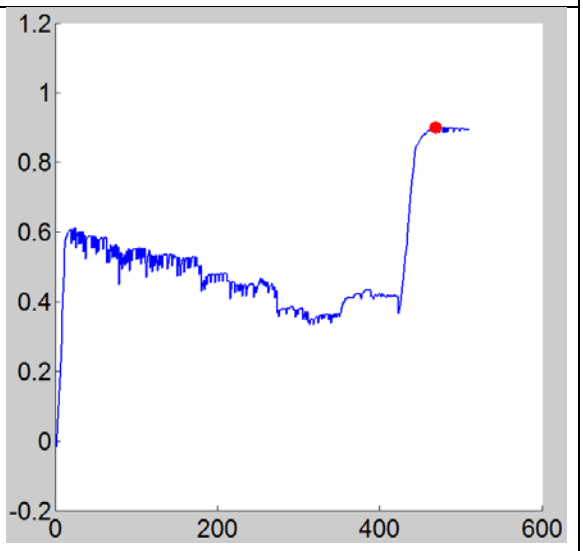
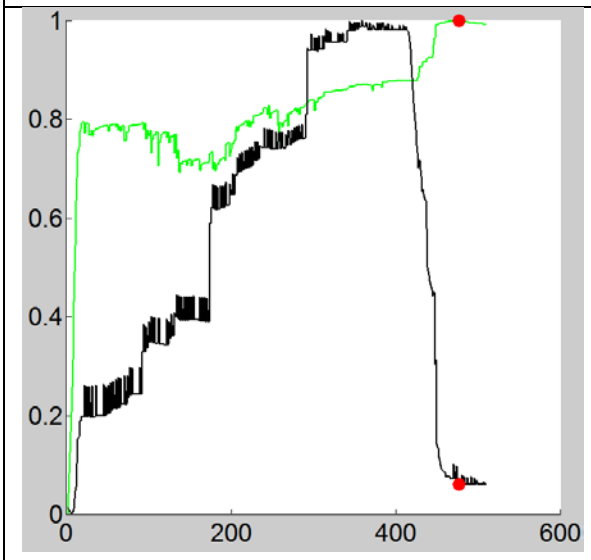
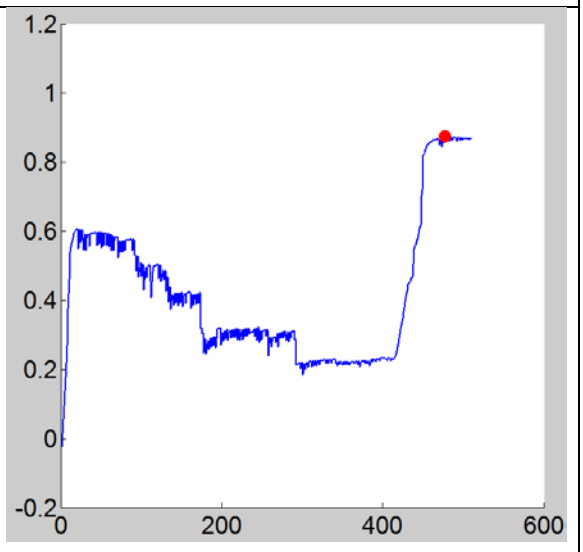
Measure	Action class	Min	Max
System Description length (bits)	Walking	668.946	3,477.430
	Running	789.98	5,934.93
	Jumping	1,333.36	10,040
	Riding bike	567.651	4,930.42
Model Description length (bits)	Walking	800	784,908
	Running	800	1,109,580
	Jumping	800	1,239,000
	Riding bike	800	1,097,460
Family resemblance typicality w/ Contrast model	Walking	-48.1476	1.31949
	Running	-47.8441	0.729305
	Jumping	-47.4041	0.505132
	Riding bike	-48.348	0.699714
Family resemblance typicality w/ Ratio model	Walking	0.000003	0.919603
	Running	0.000224	0.945360
	Jumping	0	0.864135
	Riding bike	0.000002	0.942852
Likelihood probability	Walking	0	0.016385
	Running	0	0.004467
	Jumping	-0.019825	0.000214

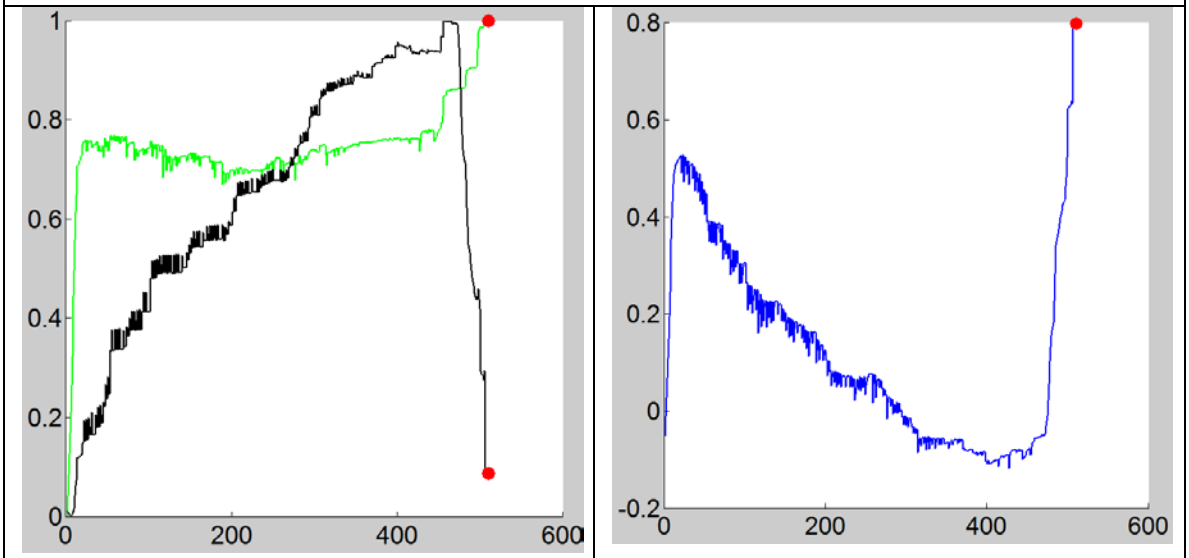
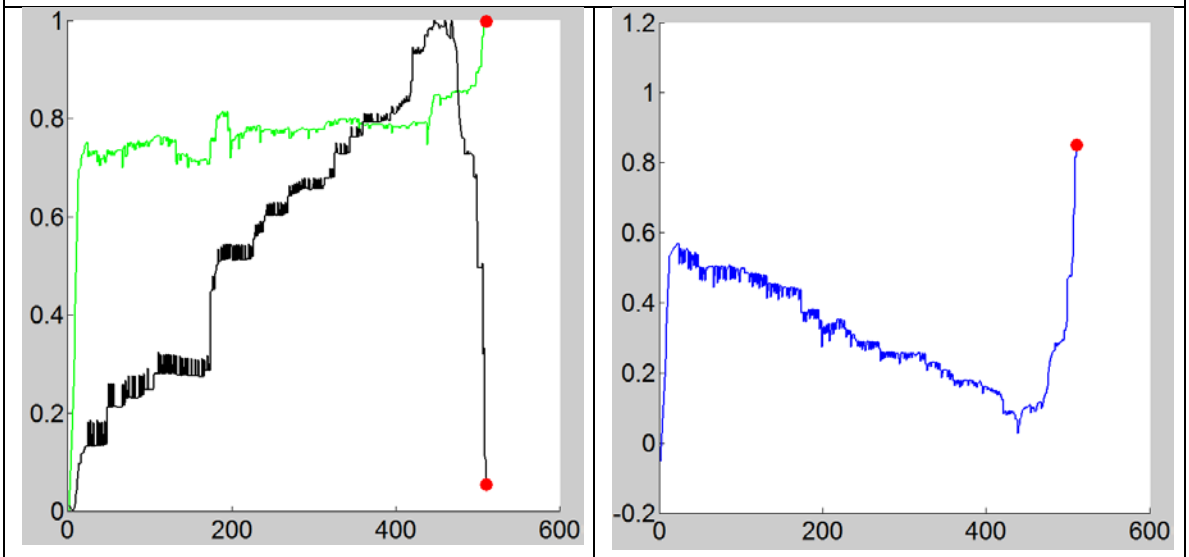
	Riding bike	0	0.007066

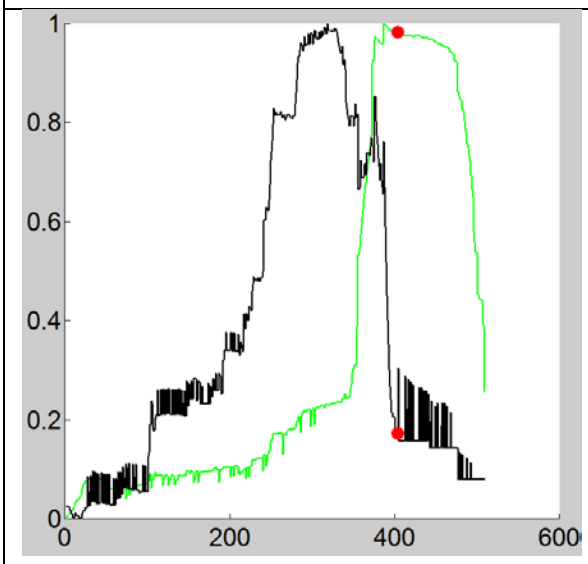
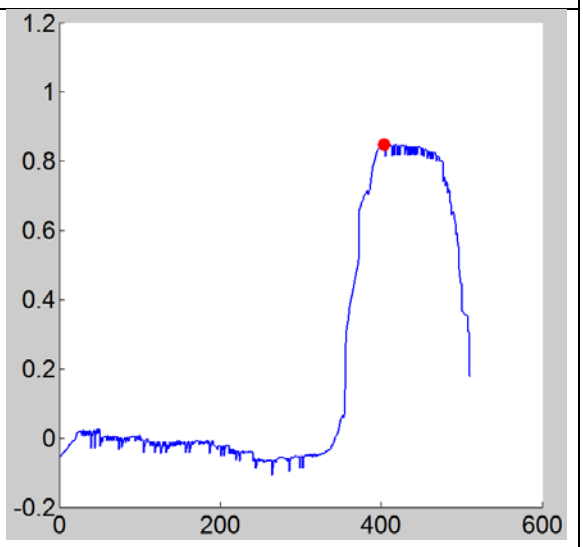
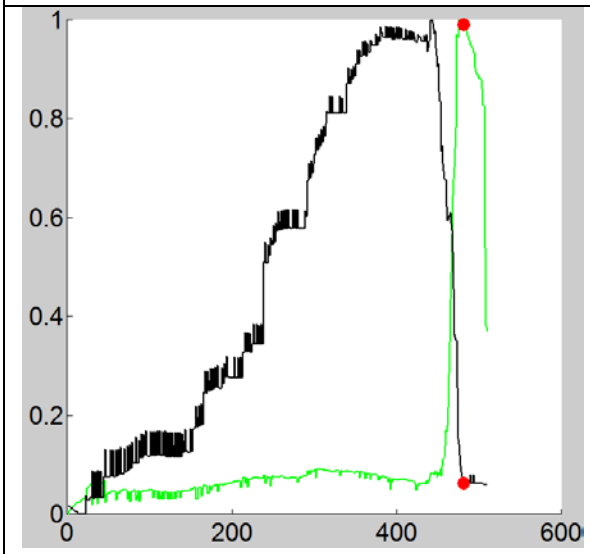
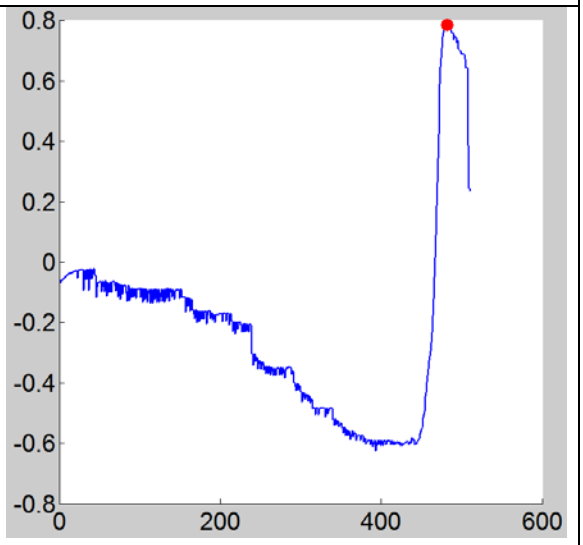
A.2. Family Resemblance

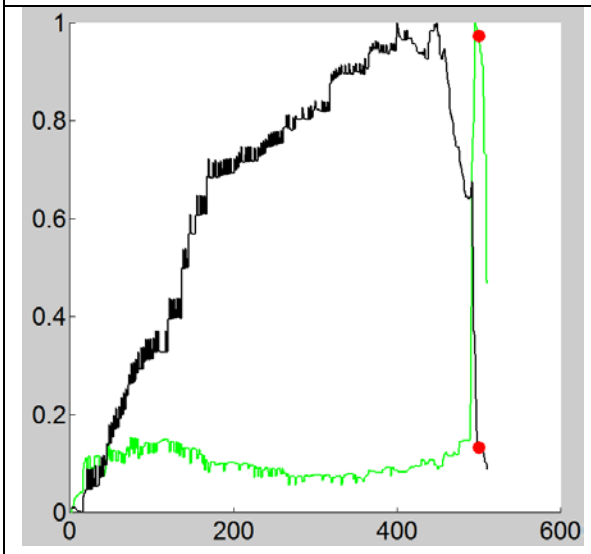
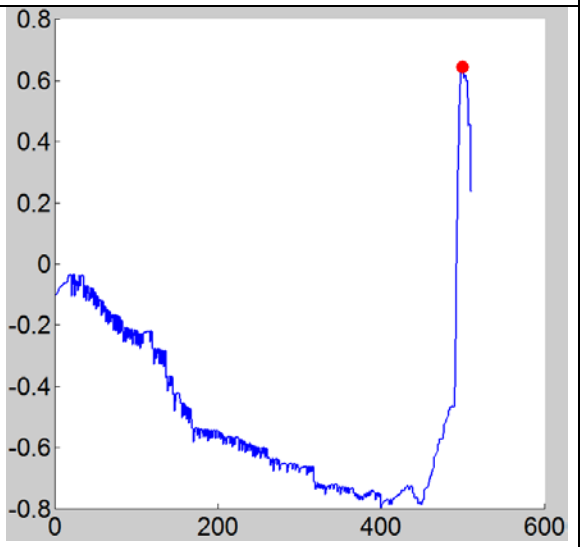
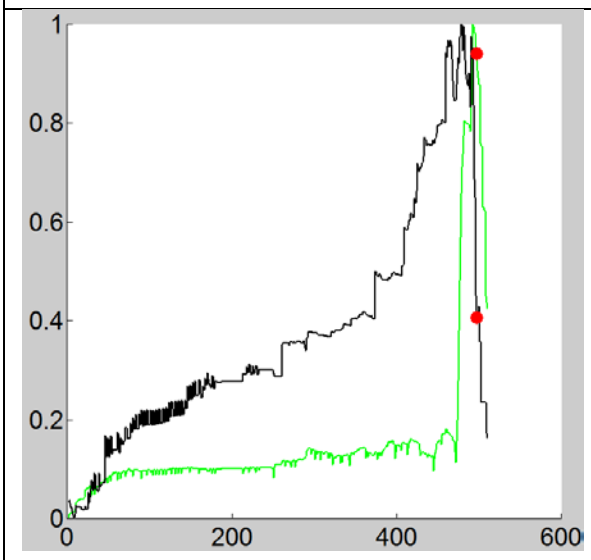
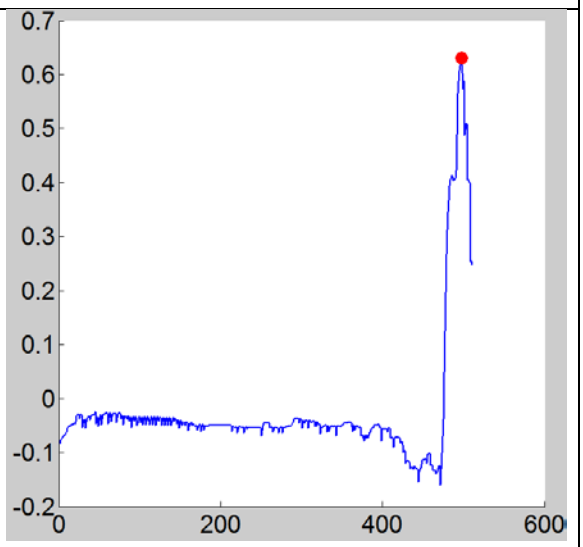
$\alpha = 1, \beta = 1, \gamma = 1$, model = Contrast, $\lambda = 0$	
Walking	Running
	
Jumping	Riding bike
	

α = 1, β = 1, γ = 1, model = Ratio, λ = 0	
Walking	Running
	
Jumping	Riding bike
	

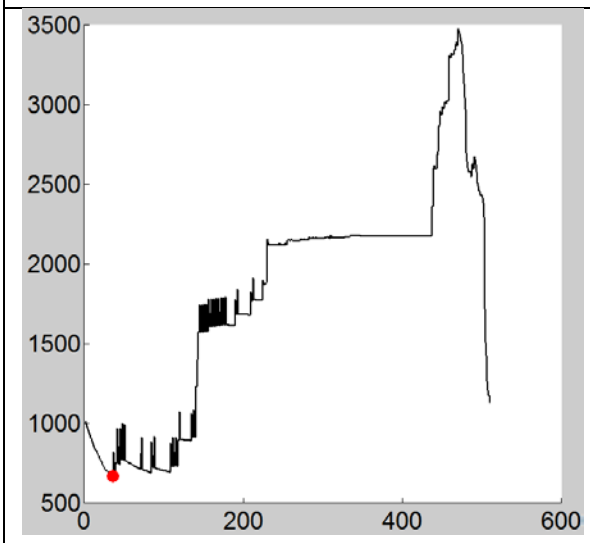
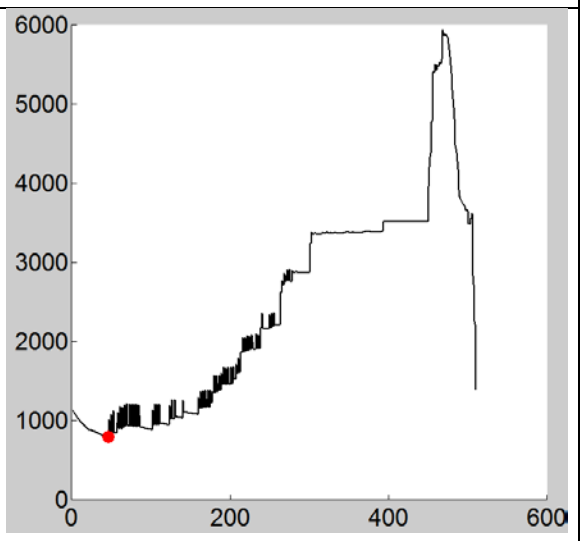
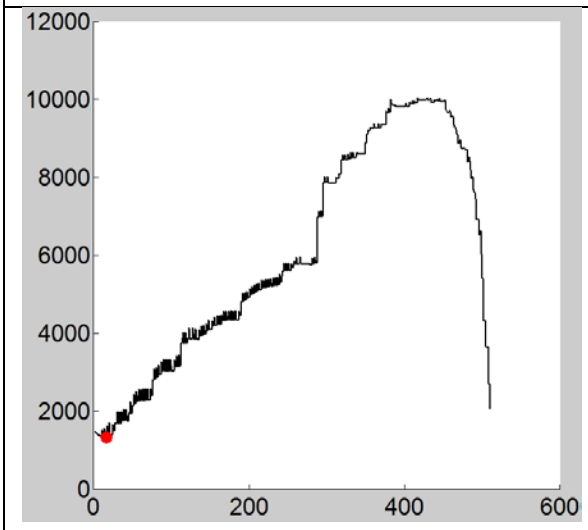
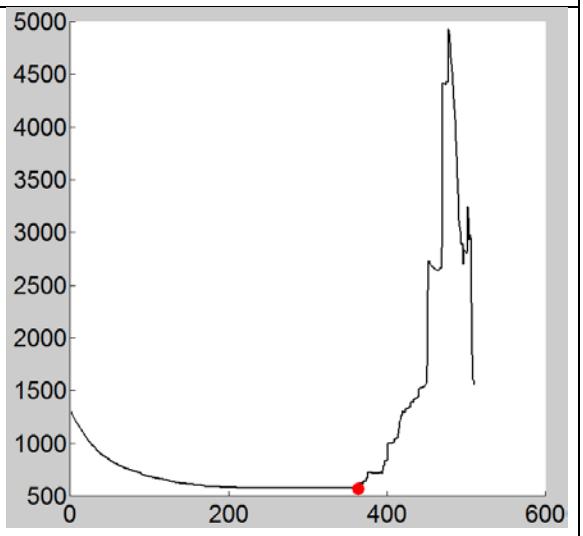
α = 1, β = 1, γ = 1, model = Contrast, λ = 1	
Walking	
	
Running	
	

Jumping

Riding bike


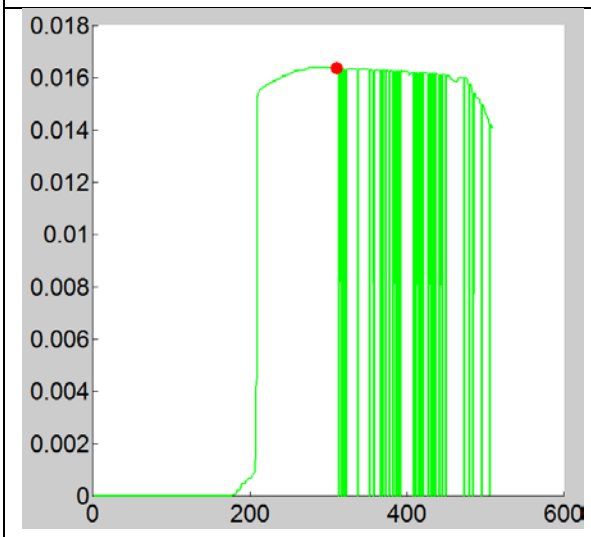
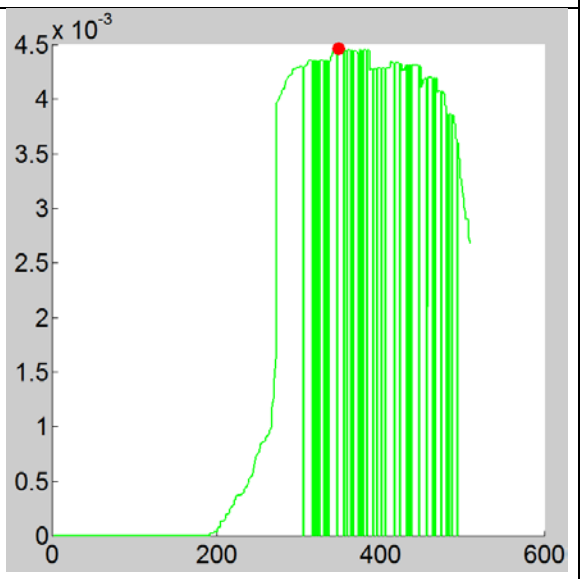
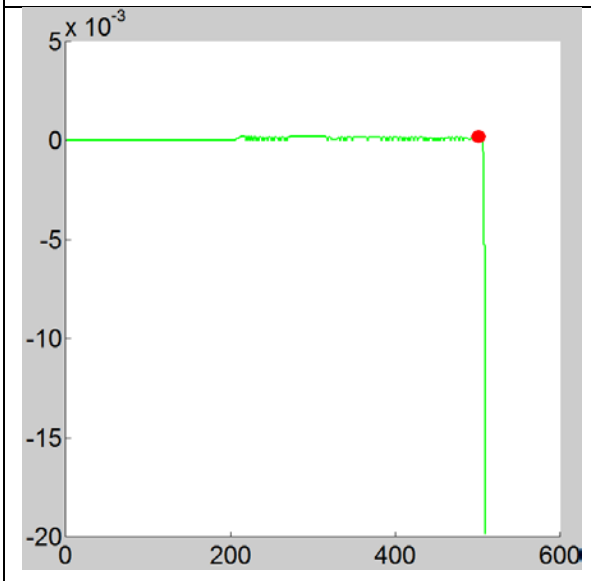
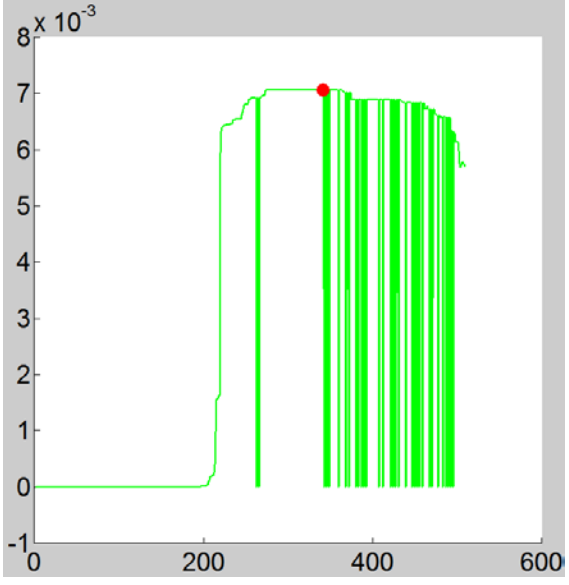
α = 1, β = 1, γ = 1, model = Ratio, λ = 1	
Walking	
	
Running	
	

Jumping	
	
Riding bike	
	

A.3. Description Length

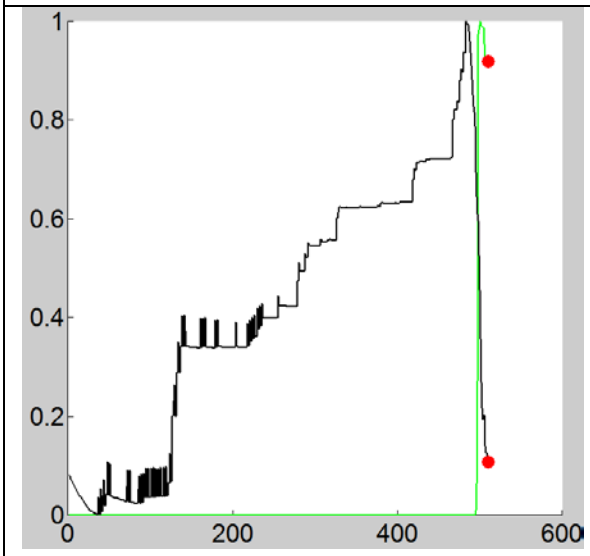
λ = 1, others = 0	
Walking	Running
	
Jumping	Riding bike
	

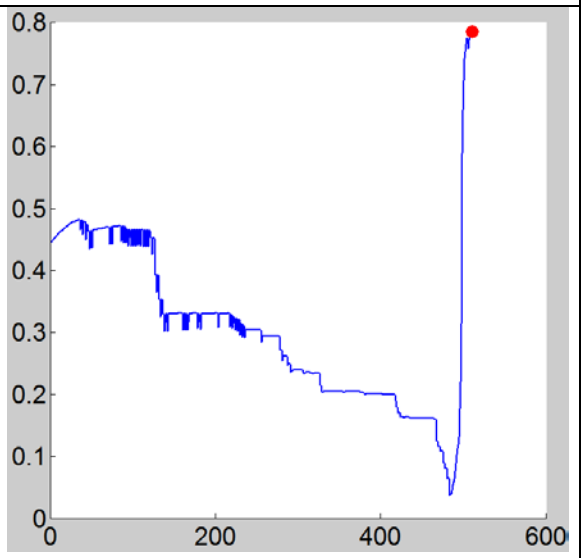
A.4. Likelihood

$\alpha = 1, \lambda = 0$	
Walking	Running
	
Jumping	Riding bike
	

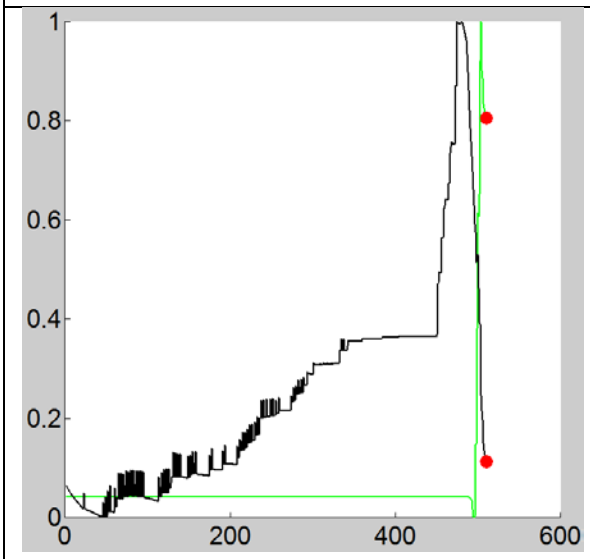
$\alpha = 1, \lambda = 1$

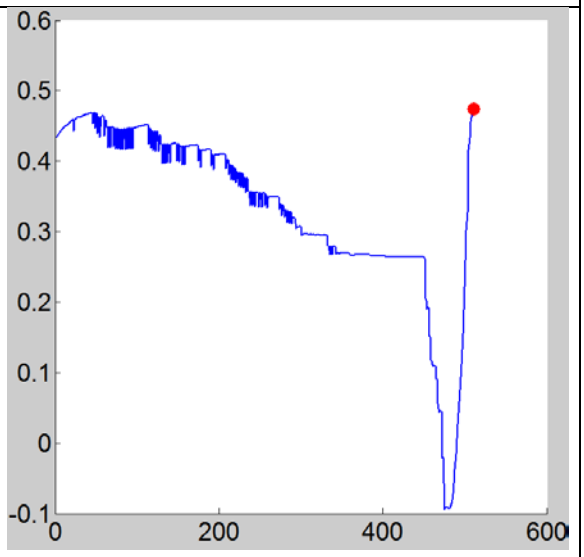
Walking

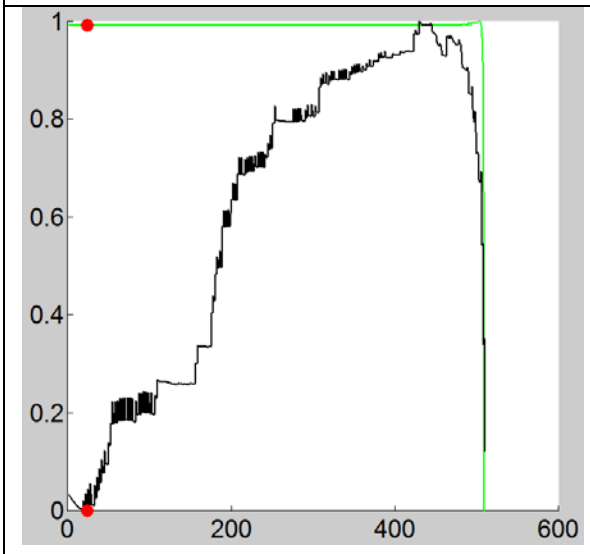
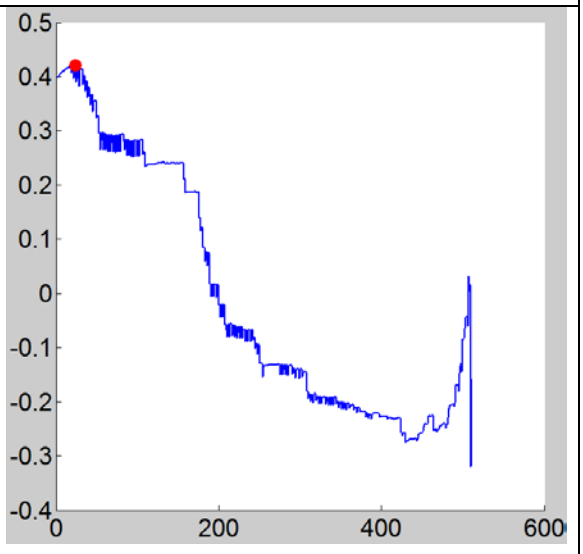
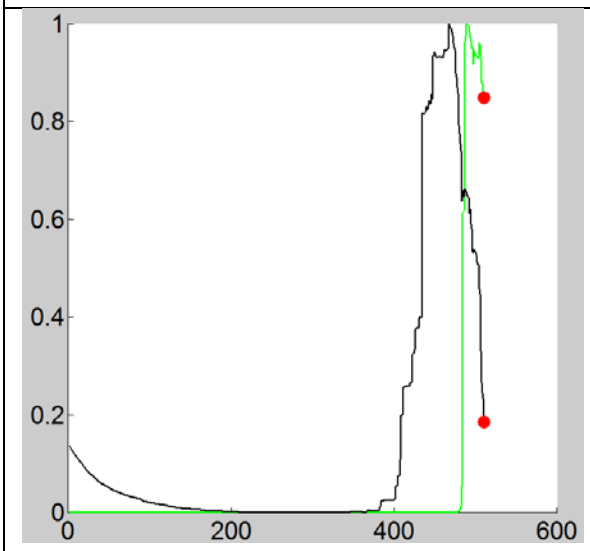
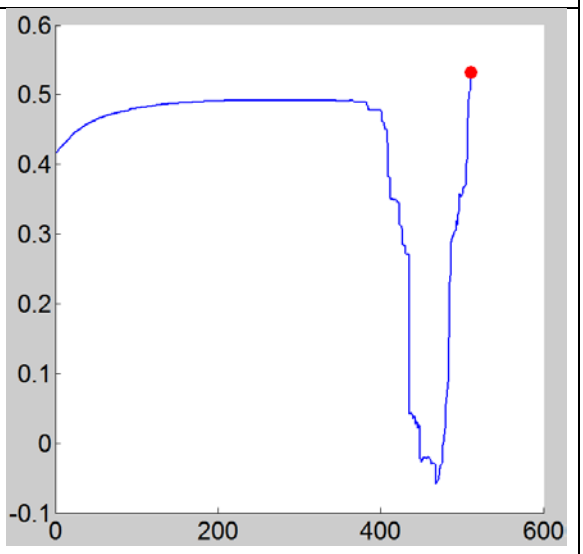




Running





Jumping	
	
Riding bike	
	

Appendix B

Prediction Performance

This section provides test results on the VOC 2011 action classification dataset. All test examples are reconstructed on a single-core CPU at 3GHz.

We report two types of results: rigid results, and soft results. Rigid results is graded for complete test examples only, which means a test example has to be fully reconstructed by a model to be counted in the system's performance. On the other hand, soft results output a non-negative number (i.e. in the range $[0, \infty)$) for each test example. The larger the number, the more likely the test example is classified as a positive one.

Rigid results are measured using recall, precision, specificity, and accuracy scores:

$$recall = \frac{TP}{TP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$specificity = \frac{TN}{FP + TN}$$

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Where:

- TP (true positive) is number of positive examples classified as positive.
- FN (false negative) is number of positive examples classified as negative.
- TN (true negative) is number of negative examples classified as negative.
- FP (false positive) is number of negative examples classified as positive.

Soft results are measured by average precision (AP) using the method provided by VOC 2011 action classification task [220]. Denote r is recall and $p(r)$ is the corresponding precision at a position in the ranked output list.

$$Average\ precision\ (AP) = \int_0^1 p(r) \times dr \approx \frac{1}{n+1} \sum_{r \in \{0, \frac{1}{n}, \frac{2}{n}, \ldots, 1\}} p(r)$$

B.1. Walking

Likelihood Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 0; test = Coverage							**0.499639**
16	38	151	8	0.784038	0.949686	0.296296	0.666667
α = 1, λ = 0; test = Probability × Coverage							**0.441902**
16	38	151	8	0.784038	0.949686	0.296296	0.666667

Likelihood − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 1; test = Coverage							**0.486036**
14	40	151	8	0.774648	0.949686	0.259259	0.636364
α = 1, λ = 1; test = Probability × Coverage							**0.416029**
14	40	151	8	0.774648	0.949686	0.259259	0.636364

Family resemblance Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Coverage							**0.502845**
12	42	152	7	0.769953	0.222222	0.955975	0.631579
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Typicality × Coverage							**0.418729**
12	42	152	7	0.769953	0.222222	0.955975	0.631579
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Coverage							**0.515816**
14	40	152	7	0.779343	0.259259	0.955975	0.666667
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Typicality × Coverage							**0.432200**
14	40	152	7	0.779343	0.259259	0.955975	0.666667

Family resemblance − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0.4; test = Coverage							**0.502845**
15	39	152	7	0.784038	0.277778	0.955975	0.681818
α = 1, β = 1, γ = 1, model = Contrast, λ = 1; test = Typicality × Coverage							**0.418729**
15	39	152	7	0.784038	0.277778	0.955975	0.681818
α = 1, β = 1, γ = 1, model = Ratio, λ = 0.1; test = Coverage							**0.511123**
16	38	152	7	0.788732	0.296296	0.955975	0.695652
α = 1, β = 1, γ = 1, model = Ratio, λ = 0.1; test = Typicality × Coverage							**0.432200**
16	38	152	7	0.788732	0.296296	0.955975	0.695652

Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
λ = 0, others = 0; test = Coverage							**0.395623**
0	54	159	0	0.746479	0	1	N/A

B.2. Running

Likelihood Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 0; test = Coverage							**0.419105**
7	53	136	17	0.671362	0.888889	0.116667	0.291667
α = 1, λ = 0; test = Probability × Coverage							**0.380263**
7	53	136	17	0.671362	0.888889	0.116667	0.291667

Likelihood − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 1; test = Coverage							**0.419105**
7	53	138	15	0.680751	0.901961	0.116667	0.318182
α = 1, λ = 1; test = Probability × Coverage							**0.380263**
7	53	138	15	0.680751	0.901961	0.116667	0.318182

Family resemblance Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Coverage							**0.417721**
6	54	140	13	0.685446	0.1	0.915033	0.315789
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Typicality × Coverage							**0.389847**
6	54	140	13	0.685446	0.1	0.915033	0.315789
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Coverage							**0.412685**
6	54	140	13	0.685446	0.1	0.915033	0.315789
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Typicality × Coverage							**0.389348**
6	54	140	13	0.685446	0.1	0.915033	0.315789

Family resemblance − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Coverage							**0.419861**
6	54	140	13	0.685446	0.915033	0.1	0.315789
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Typicality × Coverage							**0.391967**
6	54	140	13	0.685446	0.915033	0.1	0.315789
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Coverage							**0.419861**
6	54	140	13	0.685446	0.915033	0.1	0.315789
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Typicality × Coverage							**0.391967**
6	54	140	13	0.685446	0.915033	0.1	0.315789

Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
λ = 0, others = 0; test = Coverage							**0.410717**
0	60	153	0	0.71831	0	1	N/A

B.3. Jumping

Likelihood Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 0; test = Coverage							**0.298077**
0	62	151	0	0.70892	1	0	N/A
α = 1, λ = 0; test = Probability × Coverage							**0.601942**
0	62	151	0	0.70892	1	0	N/A

Likelihood − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 1; test = Coverage							**0.299517**
0	62	151	0	0.70892	1	0	N/A
α = 1, λ = 1; test = Probability × Coverage							**1**
0	62	151	0	0.70892	1	0	N/A

Family resemblance Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Coverage							**0.299517**
0	62	133	18	0.624413	0	0.880795	0
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Typicality × Coverage							**0.775000**
0	62	133	18	0.624413	0	0.880795	0
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Coverage							**0.299517**
0	62	133	18	0.624413	0	0.880795	0
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Typicality × Coverage							**0.775000**
0	62	133	18	0.624413	0	0.880795	0

Family resemblance − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Coverage							**0.299517**
0	62	133	18	0.624413	0.880795	0	0
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Typicality × Coverage							**0.775000**
0	62	133	18	0.624413	0.880795	0	0
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Coverage							**0.299517**
0	62	133	18	0.624413	0.880795	0	0
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Typicality × Coverage							**0.775000**
0	62	133	18	0.624413	0.880795	0	0

Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
λ = 0, others = 0; test = Coverage							**0.299517**
0	62	151	0	0.70892	0	1	N/A

B.4. Riding Bike

Likelihood Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 0; test = Coverage							**0.209463**
1	36	153	23	0.723005	0.869318	0.027027	0.041667
α = 1, λ = 0; test = Probability × Coverage							**0.293694**
1	36	153	23	0.723005	0.869318	0.027027	0.041667

Likelihood − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Specificity**	**Recall**	**Precision**
α = 1, λ = 1; test = Coverage							**0.205400**
1	36	155	21	0.732394	0.880682	0.027027	0.045455
α = 1, λ = 1; test = Probability × Coverage							**0.295684**
1	36	155	21	0.732394	0.880682	0.027027	0.045455

Family resemblance Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Coverage							**0.216459**
1	36	156	20	0.737089	0.027027	0.886364	0.047619
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Typicality × Coverage							**0.313912**
1	36	156	20	0.737089	0.027027	0.886364	0.047619
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Coverage							**0.230589**
1	36	156	20	0.737089	0.027027	0.886364	0.047619
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Typicality × Coverage							**0.296978**
1	36	156	20	0.737089	0.027027	0.886364	0.047619

Family resemblance − Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Coverage							**0.216459**
1	36	156	20	0.737089	0.886364	0.027027	0.047619
α = 1, β = 1, γ = 1, model = Contrast, λ = 0; test = Typicality × Coverage							**0.322109**
1	36	156	20	0.737089	0.886364	0.027027	0.047619
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Coverage							**0.216459**
1	36	156	20	0.737089	0.886364	0.027027	0.047619
α = 1, β = 1, γ = 1, model = Ratio, λ = 0; test = Typicality × Coverage							**0.340939**
1	36	156	20	0.737089	0.886364	0.027027	0.047619

Description length Parameters							Average Precision
TP	**FN**	**TN**	**FP**	**Accuracy**	**Recall**	**Specificity**	**Precision**
λ = 0, others = 0; test = Coverage							**0.193368**
0	37	176	0	0.826291	0	1	N/A

BIBLIOGRAPHY

1. Sleator, D. and D. Temperley, *Parsing English with a Link Grammar*, in *Proc. 1993 3rd International Workshop on Parsing Technologies*. 1991.
2. Samudrala, R. and J. Moult, *Handling context-sensitivity in protein structures using graph theory: bona fide prediction.* Proteins, 1997. **Suppl 1**: p. 43-9.
3. Felzenszwalb, P.F. and D.P. Huttenlocher, *Efficient Graph-Based Image Segmentation.* Int. J. Comput. Vision, 2004. **59**(2): p. 167-181.
4. Bakir, G.H., et al., *Predicting Structured Data (Neural Information Processing)*. 2007: The MIT Press.
5. Kinderman, R. and S.L. Snell, *Markov random fields and their applications*. Vol. 6. 1980: American mathematical society. %&.
6. Tsochantaridis, I., et al., *Support vector machine learning for interdependent and structured output spaces*, in *Proceedings of the twenty-first international conference on Machine learning*. 2004, ACM: Banff, Alberta, Canada. p. 104.
7. Blei, D.M., *Probabilistic topic models.* Commun. ACM, 2012. **55**(4): p. 77-84.
8. Blei, D.M., A.Y. Ng, and M.I. Jordan, *Latent dirichlet allocation.* J. Mach. Learn. Res., 2003. **3**: p. 993-1022.
9. Deerwester, S., et al., *Indexing by latent semantic analysis.* Journal of the American Society for Information Science, 1990. **41**(6): p. 391-407.
10. Hofmann, T., *Probabilistic latent semantic indexing*, in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. 1999, ACM: Berkeley, California, USA. p. 50-57.
11. Hoffman, M.D., D.M. Blei, and F.R. Bach, *Online Learning for Latent Dirichlet Allocation*, in *NIPS*, J.D. Lafferty, et al., Editors. 2010, Curran Associates, Inc. p. 856-864.
12. Pritchard, J.K., M. Stephens, and P. Donnelly, *Inference of population structure using multilocus genotype data.* Genetics, 2000. **155**(2): p. 945-59.
13. Fei-Fei, L. and P. Perona. *A Bayesian hierarchical model for learning natural scene categories*. in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. 2005.

14. Blei, D.M. and M.I. Jordan, *Modeling annotated data*, in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. 2003, ACM: Toronto, Canada. p. 127-134.

15. Bart, E., M. Welling, and P. Perona, *Unsupervised Organization of Image Collections: Taxonomies and Beyond.* IEEE Trans. Pattern Anal. Mach. Intell., 2011. **33**(11): p. 2302-2315.

16. Wallach, H.M., *Topic modeling: beyond bag-of-words*, in *Proceedings of the 23rd international conference on Machine learning*. 2006, ACM: Pittsburgh, Pennsylvania. p. 977-984.

17. Griffiths, T., et al., *Integrating topics and syntax*, in *Adv. in Neural Information Processing Systems*. 2004. p. 537-544.

18. Fayyad, U.M., G. Piatetsky-Shapiro, and P. Smyth, *From data mining to knowledge discovery: an overview*, in *Advances in knowledge discovery and data mining*, M.F. Usama, et al., Editors. 1996, American Association for Artificial Intelligence. p. 1-34.

19. You, J., et al., *Towards Graph Summary and Aggregation: A Survey*, in *Social Media Retrieval and Mining*, S. Zhou and Z. Wu, Editors. 2013, Springer Berlin Heidelberg. p. 3-12.

20. Tian, Y., R.A. Hankins, and J.M. Patel, *Efficient aggregation for graph summarization*, in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008, ACM: Vancouver, Canada. p. 567-580.

21. Zou, L., et al., *Summarization graph indexing: Beyond frequent structure-based approach*, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2008: New Delhi. p. 141-155.

22. Pearl, J., *Probabilistic reasoning in intelligent systems: networks of plausible inference*. 1988: Morgan Kaufmann Publishers Inc. 552.

23. Koller, D. and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. 2009: The MIT Press. 1208.

24. Kimmig, A., L. Mihalkova, and L. Getoor, *Lifted graphical models: a survey.* Machine Learning, 2014: p. 1-45.

25. Poole, D., *First-order probabilistic inference*, in *Proceedings of the 18th international joint conference on Artificial intelligence*. 2003, Morgan Kaufmann Publishers Inc.: Acapulco, Mexico. p. 985-991.

26. Della Vigna, P. and C. Ghezzi, *Context-free graph grammars.* Information and Control, 1978. **37**(2): p. 207-233.

27. Fahmy, H. and D. Blostein. *A survey of graph grammars: theory and applications*. in *Pattern Recognition, 1992. Vol.II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on*. 1992.

28. Vento, M., *A long trip in the charming world of graphs for Pattern Recognition.* Pattern Recognition, 2014(0).

29. Flasi, M., et al., *Fundamental methodological issues of syntactic pattern recognition.* Pattern Anal. Appl., 2014. **17**(3): p. 465-480.

30. FOGGIA, P., G. PERCANNELLA, and M. VENTO, *Graph Matching and Learning in Pattern Recognition in the last 10 years.* International Journal of Pattern Recognition and Artificial Intelligence, 2014. **28**(01): p. 1450001.

31. Vapnik, V.N. and V. Vapnik, *Statistical learning theory*. Vol. 2. 1998: Wiley New York.

32. Burges, C.J., *A tutorial on support vector machines for pattern recognition.* Data mining and knowledge discovery, 1998. **2**(2): p. 121-167.

33. Cristianini, N. and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. 2000: Cambridge university press.

34. Herbrich, R., *Learning kernel classifiers*. 2002: MIT Press, Cambridge.

35. Schölkopf, B. and A.J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. 2002: MIT press.

36. Kimeldorf, G. and G. Wahba, *Some results on Tchebycheffian spline functions.* Journal of Mathematical Analysis and Applications, 1971. **33**(1): p. 82-95.

37. Cox, D.D. and F. O'Sullivan, *Asymptotic analysis of penalized likelihood and related estimators.* The Annals of Statistics, 1990: p. 1676-1695.

38. Schölkopf, B., A. Smola, and K.-R. Müller, *Kernel principal component analysis*, in *Artificial Neural Networks—ICANN'97*. 1997, Springer. p. 583-588.

39. Haussler, D., *Convolution kernels on discrete structures*. 1999, Technical report, Department of Computer Science, University of California at Santa Cruz.

40. Collins, M. and N. Duffy. *Convolution kernels for natural language*. in *Advances in neural information processing systems*. 2001.

41. Borgwardt, K.M. and H.-P. Kriegel. *Shortest-path kernels on graphs*. in *Data Mining, Fifth IEEE International Conference on*. 2005. IEEE.

42. Kashima, H., K. Tsuda, and A. Inokuchi. *Marginalized kernels between labeled graphs*. in *ICML*. 2003.

43. Jaakkola, T. and D. Haussler. *Probabilistic kernel regression models*. in *Proceedings of the 1999 Conference on AI and Statistics*. 1999. San Mateo, CA.

44. Weston, J., et al. *Kernel dependency estimation*. in *Advances in neural information processing systems*. 2002.

45. Hammersley, J.M. and P. Clifford, *Markov fields on finite graphs and lattices.* 1971.

46. Besag, J., *Spatial interaction and the statistical analysis of lattice systems.* Journal of the Royal Statistical Society. Series B (Methodological), 1974: p. 192-236.

47. Heckerman, D., *A tutorial on learning with Bayesian networks*. 1998: Springer.

48. Cowell, R.G., *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. 2006: Springer Science & Business.

49. Murphy, K.P., Y. Weiss, and M.I. Jordan. *Loopy belief propagation for approximate inference: An empirical study*. in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. 1999. Morgan Kaufmann Publishers Inc.

50. Kschischang, F.R., B.J. Frey, and H.-A. Loeliger, *Factor graphs and the sum-product algorithm.* Information Theory, IEEE Transactions on, 2001. **47**(2): p. 498-519.

51. Taskar, B., P. Abbeel, and D. Koller. *Discriminative probabilistic models for relational data*. in *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*. 2002. Morgan Kaufmann Publishers Inc.

52. Richardson, M. and P. Domingos, *Markov logic networks.* Machine learning, 2006. **62**(1-2): p. 107-136.

53. McCallum, A., K. Schultz, and S. Singh. *Factorie: Probabilistic programming via imperatively defined factor graphs*. in *Advances in Neural Information Processing Systems*. 2009.

54. Kersting, K. and L. De Raedt, *Towards combining inductive logic programming with Bayesian networks*, in *Inductive Logic Programming*. 2001, Springer. p. 118-131.

55. Koller, D. and A. Pfeffer. *Probabilistic frame-based systems*. in *AAAI/IAAI*. 1998.

56. Milch, B., et al., *1 BLOG: Probabilistic Models with Unknown Objects.* Statistical relational learning, 2007: p. 373.

57. Koller, D. and A. Pfeffer. *Object-oriented Bayesian networks*. in *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*. 1997. Morgan Kaufmann Publishers Inc.

58. Pfeffer, A., et al. *SPOOK: A system for probabilistic object-oriented knowledge representation*. in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. 1999. Morgan Kaufmann Publishers Inc.

59. Kersting, K. *Lifted Probabilistic Inference*. in *ECAI*. 2012.

60. de Salvo Braz, R., *Lifted first-order probabilistic inference*. 2007: ProQuest.

61. Taghipour, N., et al. *Lifted variable elimination with arbitrary constraints*. in *Proceedings of the fifteenth international conference on Artificial Intelligence and Statistics, JMLR workshop and conference proceedings*. 2012.

62. Jaimovich, A., O. Meshi, and N. Friedman, *Template based inference in symmetric relational Markov random fields.* arXiv preprint arXiv:1206.5276, 2012.

63. Singla, P. and P. Domingos. *Lifted First-Order Belief Propagation*. in *AAAI*. 2008.

64. Wellman, M.P., J.S. Breese, and R.P. Goldman, *From knowledge bases to decision models.* The Knowledge Engineering Review, 1992. **7**(01): p. 35-53.

65. Bishop, C.M., *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2006: Springer-Verlag New York, Inc.

66. Milch, B. and S. Russell, *General-purpose MCMC inference over relational structures.* arXiv preprint arXiv:1206.6849, 2012.

67. Broecheler, M. and L. Getoor. *Computing marginal distributions over continuous Markov networks for statistical relational learning*. in *Advances in Neural Information Processing Systems*. 2010.

68. Niepert, M. *Symmetry-Aware Marginal Density Estimation*. in *AAAI*. 2013.

69. Getoor, L. and L. Mihalkova. *Learning statistical models from relational data*. in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011. ACM.

70. *ICTV dB*. Available from: http://www.ncbi.nlm.nih.gov/ICTVdb/.

71. Lowd, D. and P. Domingos, *Efficient weight learning for Markov logic networks*, in *Knowledge Discovery in Databases: PKDD 2007*. 2007, Springer. p. 200-211.

72. Singla, P. and P. Domingos. *Discriminative training of Markov logic networks*. in *AAAI*. 2005.

73. Huynh, T.N. and R.J. Mooney, *Max-margin weight learning for Markov logic networks*, in *Machine Learning and Knowledge Discovery in Databases*. 2009, Springer. p. 564-579.

74. Ahmadi, B., et al., *Exploiting symmetries for scaling loopy belief propagation and relational training.* Machine learning, 2013. **92**(1): p. 91-132.

75. De Raedt, L. and K. Kersting, *Statistical relational learning*, in *Encyclopedia of Machine Learning*. 2010, Springer. p. 916-924.

76. Friedman, N., et al. *Learning probabilistic relational models*. in *IJCAI*. 1999.

77. Kok, S. and P. Domingos. *Learning the structure of Markov logic networks*. in *Proceedings of the 22nd international conference on Machine learning*. 2005. ACM.

78. Biba, M., S. Ferilli, and F. Esposito, *Discriminative structure learning of Markov logic networks*, in *Inductive Logic Programming*. 2008, Springer. p. 59-76.

79. Khosravi, H., et al. *Structure Learning for Markov Logic Networks with Many Descriptive Attributes*. in *AAAI*. 2010.

80. Kok, S. and P. Domingos. *Learning Markov logic networks using structural motifs*. in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010.

81. CONTE, D., et al., *Thirty Years Of Graph Matching In Pattern Recognition.* International Journal of Pattern Recognition and Artificial Intelligence, 2004. **18**(03): p. 265-298.

82. Konc, J. and D. Janezic, *An improved branch and bound algorithm for the maximum clique problem.* proteins, 2007. **4**: p. 5.

83. Gori, M., M. Maggini, and L. Sarti, *Exact and approximate graph matching using random walks.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2005. **27**(7): p. 1100-1111.

84. Dahm, N., et al. *Topological features and iterative node elimination for speeding up subgraph isomorphism detection*. in *Pattern Recognition (ICPR), 2012 21st International Conference on*. 2012. IEEE.

85. Sanfeliu, A., et al., *Graph-based representations and techniques for image processing and image analysis.* Pattern Recognition, 2002. **35**(3): p. 639-650.

86. Serratosa, F., R. Alquézar, and A. Sanfeliu, *Function-described graphs for modelling objects represented by sets of attributed graphs.* Pattern Recognition, 2003. **36**(3): p. 781-798.

87. Massaro, A. and M. Pelillo, *Matching graphs by pivoting.* Pattern Recognition Letters, 2003. **24**(8): p. 1099-1106.

88. Solé-Ribalta, A. and F. Serratosa, *Models and algorithms for computing the common labelling of a set of attributed graphs.* Computer Vision and Image Understanding, 2011. **115**(7): p. 929-945.

89. Caelli, T. and S. Kosinov, *An eigenspace projection clustering method for inexact graph matching.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2004. **26**(4): p. 515-519.

90. Duchenne, O., et al., *A tensor-based algorithm for high-order graph matching.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2011. **33**(12): p. 2383-2395.

91. Gao, X., et al., *A survey of graph edit distance.* Pattern Analysis and applications, 2010. **13**(1): p. 113-129.

92. Bonabeau, E., *Graph multidimensional scaling with self-organizing maps.* Information Sciences, 2002. **143**(1): p. 159-180.

93. de Mauro, C., et al., *Similarity learning for graph-based image representations.* Pattern Recognition Letters, 2003. **24**(8): p. 1115-1122.

94. Jouili, S. and S. Tabbone, *Graph embedding using constant shift embedding*, in *Recognizing Patterns in Signals, Speech, Images and Videos*. 2010, Springer. p. 83-92.

95. Luo, B., R. C Wilson, and E.R. Hancock, *Spectral embedding of graphs.* Pattern recognition, 2003. **36**(10): p. 2213-2230.

96. Xiao, B., S. Yi-Zhe, and P. Hall, *Learning invariant structure for object identification by using graph methods.* Computer Vision and Image Understanding, 2011. **115**(7): p. 1023-1031.

97. Torsello, A. and E.R. Hancock, *Graph embedding using tree edit-union.* Pattern Recognition, 2007. **40**(5): p. 1393-1405.

98. Czech, W., *Graph descriptors from b-matrix representation*, in *Graph-Based Representations in Pattern Recognition*. 2011, Springer. p. 12-21.

99. Riesen, K., M. Neuhaus, and H. Bunke, *Graph embedding in vector spaces by means of prototype selection*, in *Graph-Based Representations in Pattern Recognition*. 2007, Springer. p. 383-393.

100. Zare Borzeshi, E., et al., *Discriminative prototype selection methods for graph embedding.* Pattern Recognition, 2013. **46**(6): p. 1648-1657.

101. Neuhaus, M. and H. Bunke, *Edit distance-based kernel functions for structural pattern classification.* Pattern Recognition, 2006. **39**(10): p. 1852-1863.

102. Neuhaus, M., K. Riesen, and H. Bunke, *Novel kernels for error-tolerant graph classification.* Spatial vision, 2009. **22**(5): p. 425-441.

103. Gaüzere, B., L. Brun, and D. Villemin, *Two new graphs kernels in chemoinformatics.* Pattern Recognition Letters, 2012. **33**(15): p. 2038-2047.

104. Shervashidze, N., et al. *Efficient graphlet kernels for large graph comparison*. in *International Conference on Artificial Intelligence and Statistics*. 2009.

105. Bai, L. and E.R. Hancock, *Graph kernels from the jensen-shannon divergence.* Journal of mathematical imaging and vision, 2013. **47**(1-2): p. 60-69.

106. Günter, S. and H. Bunke, *Self-organizing map for clustering in the graph domain.* Pattern Recognition Letters, 2002. **23**(4): p. 405-417.

107. Serratosa, F., R. Alquézar, and A. Sanfeliu, *Synthesis of function-described graphs and clustering of attributed graphs.* International journal of pattern recognition and artificial intelligence, 2002. **16**(06): p. 621-655.

108. Jain, B.J. and K. Obermayer, *Graph quantization.* Computer Vision and Image Understanding, 2011. **115**(7): p. 946-961.

109. Guigues, L., H. Le Men, and J.-P. Cocquerez, *The hierarchy of the cocoons of a graph and its application to image segmentation.* Pattern Recognition Letters, 2003. **24**(8): p. 1059-1066.

110. Brás Silva, H., P. Brito, and J. Pinto da Costa, *A partitional clustering algorithm validated by a clustering tendency index based on graph theory.* Pattern Recognition, 2006. **39**(5): p. 776-788.

111. Bach, S., et al., *Hinge-loss Markov random fields: Convex inference for structured prediction.* arXiv preprint arXiv:1309.6813, 2013.

112. Foggia, P., et al., *A graph-based algorithm for cluster detection.* International Journal of Pattern Recognition and Artificial Intelligence, 2008. **22**(05): p. 843-860.

113. Zanghi, H., C. Ambroise, and V. Miele, *Fast online graph clustering via Erdős–Rényi mixture.* Pattern Recognition, 2008. **41**(12): p. 3592-3599.

114. Tabatabaei, S.S., M. Coates, and M. Rabbat, *GANC: Greedy agglomerative normalized cut for graph clustering.* Pattern Recognition, 2012. **45**(2): p. 831-843.

115. Ducournau, A., et al., *A reductive approach to hypergraph clustering: An application to image segmentation.* Pattern Recognition, 2012. **45**(7): p. 2788-2803.

116. Maulik, U., *Hierarchical Pattern Discovery in Graphs.* Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 2008. **38**(6): p. 867-872.

117. Neuhaus, M. and H. Bunke, *Automatic learning of cost functions for graph edit distance.* Information Sciences, 2007. **177**(1): p. 239-247.

118. Serratosa, F., A. Solé-Ribalta, and X. Cortés, *Automatic learning of edit costs based on interactive and adaptive graph recognition*, in *Graph-Based Representations in Pattern Recognition*. 2011, Springer. p. 152-163.

119. Ferrer, M., E. Valveny, and F. Serratosa, *Median graph: A new exact algorithm using a distance based on the maximum common subgraph.* Pattern Recognition Letters, 2009. **30**(5): p. 579-588.

120. Ferrer, M., et al., *Generalized median graph computation by means of graph embedding in vector spaces.* Pattern Recognition, 2010. **43**(4): p. 1642-1655.

121. Raveaux, R., et al., *Learning graph prototypes for shape recognition.* Computer Vision and Image Understanding, 2011. **115**(7): p. 905-918.

122. Wang, B., et al., *Manifold-ranking based retrieval using k-regular nearest neighbor graph.* Pattern Recognition, 2012. **45**(4): p. 1569-1577.

123. Hu, W., et al., *Unsupervised active learning based on hierarchical graph-theoretic clustering.* Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2009. **39**(5): p. 1147-1161.

124. Janssens, D. and G. Rozenberg, *On the structure of node-label-controlled graph languages.* Information Sciences, 1980. **20**(3): p. 191-216.

125. Drewes, F., H.-J. Kreowski, and A. Habel, *Hyperedge replacement graph grammars*, in *Handbook of graph grammars and computing by graph transformation*. 1997, World Scientific Publishing Co., Inc. p. 95-162.

126. Pfaltz, J.L. and A. Rosenfeld, *Web grammars*, in *Proceedings of the 1st international joint conference on Artificial intelligence*. 1969, Morgan Kaufmann Publishers Inc.: Washington, DC. p. 609-619.

127. Pfaltz, J.L., *Web grammars and picture description.* Computer Graphics and Image Processing, 1972. **1**(2): p. 193-220.

128. Milgram, D.L., *Web automata.* Information and Control, 1975. **29**(2): p. 162-184.

129. Shi, Q.Y. and K.-S. Fu, *Parsing and Translation of (Attributed) Expansive Graph Languages for Scene Analysis.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1983. **PAMI-5**(5): p. 472-485.

130. Bunke, H. and B. Haller, *A parser for context free plex grammars*, in *Graph-Theoretic Concepts in Computer Science*, M. Nagl, Editor. 1990, Springer Berlin Heidelberg. p. 136-150.

131. Ke Juan, P., T. Yamamoto, and Y. Aoki, *A new parsing scheme for plex grammars.* Pattern Recognition, 1990. **23**(3–4): p. 393-402.

132. Earley, J., *An efficient context-free parsing algorithm.* Commun. ACM, 1970. **13**(2): p. 94-102.

133. Wittenburg, K. *Earley-style parsing for relational grammars*. in *Visual Languages, 1992. Proceedings., 1992 IEEE Workshop on*. 1992.

134. Ferrucci, F., et al. *A predictive parser for visual languages specified by relation grammars*. in *Visual Languages, 1994. Proceedings., IEEE Symposium on*. 1994.

135. Flasiński, M., *On the parsing of deterministic graph languages for syntactic pattern recognition.* Pattern Recognition, 1993. **26**(1): p. 1-16.

136. Flasiński, M., *Power properties of NLC graph grammars with a polynomial membership problem.* Theoretical Computer Science, 1998. **201**(1–2): p. 189-231.

137. Chiang, D., et al., *Parsing graphs with hyperedge replacement grammars*, in *In Proc. ACL*. 2013.

138. Lautemann, C., *The complexity of graph languages generated by hyperedge replacement.* Acta Informatica, 1990. **27**(5): p. 399-421.

139. Skomorowski, M., *Syntactic recognition of distorted patterns by means of random graph parsing.* Pattern Recogn. Lett., 2007. **28**(5): p. 572-581.

140. Zhang, D.-Q., K. Zhang, and J. Cao, *A Context-sensitive Graph Grammar Formalism for the Specification of Visual Languages.* The Computer Journal, 2001. **44**(3): p. 186-200.

141. Rekers, J. and A. SchÜRr, *Defining and Parsing Visual Languages with Layered Graph Grammars.* Journal of Visual Languages & Computing, 1997. **8**(1): p. 27-55.

142. Cook, D.J. and L.B. Holder, *Substructure discovery using minimum description length and background knowledge.* J. Artif. Int. Res., 1994. **1**(1): p. 231-255.

143. Rissanen, J., *Stochastic Complexity in Statistical Inquiry Theory*. 1989: World Scientific Publishing Co., Inc. 177.

144. JONYER, I., L.B. HOLDER, and D.J. COOK, *MDL-BASED CONTEXT-FREE GRAPH GRAMMAR INDUCTION AND APPLICATIONS.* International Journal on Artificial Intelligence Tools, 2004. **13**(01): p. 65-79.

145. Doshi, S., F. Huang, and T. Oates. *Inferring the structure of graph grammars from data*. in *Proceedings of the International Conference on Knowledge-Based Computer Systems*. 2002. Citeseer.

146. Oates, T., S. Doshi, and F. Huang, *Estimating Maximum Likelihood Parameters for Stochastic Context-Free Graph Grammars*, in *Inductive Logic Programming*, T. Horváth and A. Yamamoto, Editors. 2003, Springer Berlin Heidelberg. p. 281-298.

147. Dempster, A.P., N.M. Laird, and D.B. Rubin, *Maximum Likelihood from Incomplete Data via the EM Algorithm.* Journal of the Royal Statistical Society. Series B (Methodological), 1977. **39**(1): p. 1-38.

148. Lari, K. and S.J. Young, *The estimation of stochastic context-free grammars using the Inside-Outside algorithm.* Computer Speech & Language, 1990. **4**(1): p. 35-56.

149. Ates, K., et al., *Graph Grammar Induction on Structural Data for Visual Programming*, in *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*. 2006, IEEE Computer Society. p. 232-242.

150. Lev, G., *A unified approach to pattern recognition.* Pattern Recognition, 1984. **17**(5): p. 575-582.

151. Goldfarb, L., *Metric data models and associated search strategies.* ACM SIGIR Forum, 1986. **20**(1-4): p. 7-11.

152. Goldfarb, L., *On the foundations of intelligent processes;I. an evolving model for pattern learning.* Pattern Recognition, 1990. **23**(6): p. 595-616.

153. Chan, T.Y.T. and L. Goldfarb, *Primitive pattern learning.* Pattern Recognition, 1992. **25**(8): p. 883-889.

154. Lev, G., *What is distance and why do we need the metric model for pattern learning?* Pattern Recognition, 1992. **25**(4): p. 431-438.

155. Goldfarb, L. and S. Nigam, *The unified learning paradigm: A foundation for AI*, in *In V. Honavar and L. Uhr (Eds.), Artificial Intelligence and Neural Networks: Steps towards Principled Integration*. 1994, Academic Press.

156. Goldfarb, L., et al., *Can a vector space based learning model discover inductive class generalization in a symbolic environment?* Pattern Recognition Letters, 1995. **16**(7): p. 719-726.

157. Goldfarb, L. and S. Deshpande. *What is a symbolic measurement process?* in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*. 1997.

158. Goldfarb, L. and J. Hook, *Why Classical Models for Pattern Recognition are Not Pattern Recognition Models*, in *International Conference on Advances in Pattern Recognition*, S. Singh, Editor. 1999, Springer London. p. 405-414.

159. Korkin, D. and L. Goldfarb, *Multiple genome rearrangement: a general approach via the evolutionary genome graph.* Bioinformatics, 2002. **18**(suppl 1): p. S303-S311.

160. Goldfarb, L., *Representational formalisms: why we haven't had one*, in *Proc. ICPR 2004 Satellite Workshop on Pattern Representation and the Future of Pattern Recognition*. 2004.

161. Goldfarb, L., *Representation before computation.* Natural Computing, 2010. **9**(2): p. 365-379.

162. Lev Goldfarb, D.G., Oleg Golubitsky, Dmitry Korkin, Ian Scrimger, *What is a structural representation? A proposal for an event-based representational formalism.* 2008. **Sixth edition**.

163. Theobald, D.L., *A formal test of the theory of universal common ancestry.* Nature, 2010. **465**(7295): p. 219-22.

164. Hopcroft, J.E., R. Motwani, and J.D. Ullman, *Context-Free Grammars*, in *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. 2006, Addison-Wesley Longman Publishing Co., Inc. p. 77–106.

165. Korkin, D., *A new model for molecular representation and classification: Formal approach based on the ETS framework*, in *Computer Science*. 2003, The University of New Brunswick.

166. Fu, K.S., *Syntactic methods in pattern recognition*. 1974: Elsevier.

167. Wen-Hsiang, T. and F. King-Sun, *Error-Correcting Isomorphisms of Attributed Relational Graphs for Pattern Analysis.* Systems, Man and Cybernetics, IEEE Transactions on, 1979. **9**(12): p. 757-768.

168. Wen-Hsiang, T. and F. King-Sun, *Subgraph error-correcting isomorphisms for syntactic pattern recognition.* Systems, Man and Cybernetics, IEEE Transactions on, 1983. **SMC-13**(1): p. 48-62.

169. Freeman, H., *On the Encoding of Arbitrary Geometric Configurations.* Electronic Computers, IRE Transactions on, 1961. **EC-10**(2): p. 260-268.

170. Fischler, M.A. and R.A. Elschlager, *The Representation and Matching of Pictorial Structures.* Computers, IEEE Transactions on, 1973. **C-22**(1): p. 67-92.

171. Cordella, L.P., et al. *An efficient algorithm for the inexact matching of ARG graphs using a contextual transformational model*. in *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*. 1996.

172. Cordella, L.P., et al., *Subgraph Transformations for the Inexact Matching of Attributed Relational Graphs*, in *Graph Based Representations in Pattern Recognition*, J.-M. Jolion and W. Kropatsch, Editors. 1998, Springer Vienna. p. 43-52.

173. Sanfeliu, A. and K.-S. Fu, *A distance measure between attributed relational graphs for pattern recognition.* Systems, Man and Cybernetics, IEEE Transactions on, 1983. **SMC-13**(3): p. 353-362.

174. Sriram, R. and H. Garcia-Molina. *Representing Web graphs*. in *Data Engineering, 2003. Proceedings. 19th International Conference on*. 2003.

175. Bunke, H., *On the generative power of sequential and parallel programmed graph grammars.* Computing, 1982. **29**(2): p. 89-112.

176. Joshi, A.K. and Y. Schabes, *Tree-adjoining grammars*, in *Handbook of formal languages, vol. 3*, R. Grzegorz and S. Arto, Editors. 1997, Springer-Verlag New York, Inc. p. 69-123.

177. Joshi, A.K., L.S. Levy, and M. Takahashi, *Tree adjunct grammars.* Journal of Computer and System Sciences, 1975. **10**(1): p. 136-163.

178. Chi, Z. and S. Geman, *Estimation of probabilistic context-free grammars.* Comput. Linguist., 1998. **24**(2): p. 299-305.

179. Booth, T.L. and R.A. Thompson, *Applying Probability Measures to Abstract Languages.* IEEE Trans. Comput., 1973. **22**(5): p. 442-450.

180. Charniak, E., *Tree-bank Grammars*. 1996, Brown University.

181. Charniak, E., *Statistical parsing with a context-free grammar and word statistics*, in *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*. 1997, AAAI Press: Providence, Rhode Island. p. 598-603.

182. Loken, B. and J. Ward, *Measure of the attribute structure underlying product typicality.* Advances in Consumer Research, 1987. **14**: p. 22-26.

183. Loken, B. and J.C. Ward, *Alternative Approaches to Understanding the Determinants of Typicality.* Journal of Consumer Research: An Interdisciplinary Quarterly, 1990. **17**(2): p. 111-26.

184. Rosch, E. and C.B. Mervis, *Family resemblances: Studies in the internal structure of categories.* Cognitive Psychology, 1975. **7**(4): p. 573-605.

185. Tversky, A., *Features of similarity.* Psychological Review, 1977. **84**(4): p. 327-352.

186. Agrawal, R. and R. Srikant, *Fast Algorithms for Mining Association Rules in Large Databases*, in *Proceedings of the 20th International Conference on Very Large Data Bases*. 1994, Morgan Kaufmann Publishers Inc. p. 487-499.

187. Pal, N.R., et al., *A Possibilistic Fuzzy c-Means Clustering Algorithm.* Fuzzy Systems, IEEE Transactions on, 2005. **13**(4): p. 517-530.

188. Zadeh, L.A., *Fuzzy sets as a basis for a theory of possibility.* Fuzzy Sets Syst., 1999. **100**: p. 9-34.

189. Dubois, D. and H. Prade, *Possibility Theory, Probability Theory and Multiple-Valued Logics: A Clarification.* Annals of Mathematics and Artificial Intelligence, 2001. **32**(1-4): p. 35-66.

190. Krishnapuram, R. and J.M. Keller, *A possibilistic approach to clustering.* Fuzzy Systems, IEEE Transactions on, 1993. **1**(2): p. 98-110.

191. Jukna, S., *On Graph Complexity*. Combinatorics, Probability and Computing, 2006. **15**(06): p. 855-876.

192. Neel, D.L. and M.E. Orrison, *The Linear Complexity of a Graph*, in *Advances in Network Complexity*. 2013, Wiley-VCH Verlag GmbH & Co. KGaA. p. 155-175.

193. Dehmer, M., S. Borgert, and F. Emmert-Streib. *Network Classes and Graph Complexity Measures*. in *Complexity and Intelligence of the Artificial and Natural Complex Systems, Medical Applications of the Complex Systems, Biomedical Computing, 2008. CANS '08. First International Conference on*. 2008.

194. Rissanen, J., *Modeling by shortest data description.* Automatica, 1978. **14**(5): p. 465-471.

195. Kuramochi, M. and G. Karypis, *An Efficient Algorithm for Discovering Frequent Subgraphs.* IEEE Trans. on Knowl. and Data Eng., 2004. **16**(9): p. 1038-1051.

196. Garey, M.R. and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979: W. H. Freeman \& Co. 338.

197. Duda, R.O., P.E. Hart, and D.G. Stork, *Pattern Classification (2nd Edition)*. 2000

198. Döpmann, C., *Survey on the Graph Alignment Problem and a Benchmark of Suitable Algorithms*, in *Institut für Informatik*. 2013, Humboldt-Universität Zu Berlin.

199. Kimelfeld, B. and P.G. Kolaitis, *The complexity of mining maximal frequent subgraphs*, in *Proceedings of the 32nd symposium on Principles of database systems*. 2013, ACM: New York, New York, USA. p. 13-24.

200. Johnson, M., *PCFG models of linguistic tree representations.* Comput. Linguist., 1998. **24**(4): p. 613-632.

201. Suzuki, J., *A construction of Bayesian networks from databases based on an MDL principle*, in *Proceedings of the Ninth international conference on Uncertainty in artificial intelligence*. 1993, Morgan Kaufmann Publishers Inc.: Washihgton, DC. p. 266-273.

202. Cooper, G.F. and E. Herskovits, *A Bayesian Method for the Induction of Probabilistic Networks from Data.* Mach. Learn., 1992. **9**(4): p. 309-347.

203. Heckerman, D., D. Geiger, and D.M. Chickering, *Learning Bayesian Networks: The Combination of Knowledge and Statistical Data.* Mach. Learn., 1995. **20**(3): p. 197-243.

204. Cordella, L.P., et al., *A (sub) graph isomorphism algorithm for matching large graphs.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2004. **26**(10): p. 1367-1372.

205. Cordella, L., et al. *An improved algorithm for matching large graphs*. in *In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen*. 2001. Citeseer.

206. Aggarwal, J.K. and M.S. Ryoo, *Human activity analysis: A review.* ACM Comput. Surv., 2011. **43**(3): p. 1-43.

207. Guo, G. and A. Lai, *A survey on still image based human action recognition.* Pattern Recognition, 2014. **47**(10): p. 3343-3361.

208. Yang, W., et al. *Unsupervised Discovery of Action Classes*. in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. 2006.

209. Weilong, Y., W. Yang, and G. Mori. *Recognizing human actions from still images with latent poses*. in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. 2010.

210. Desai, C. and D. Ramanan, *Detecting actions, poses, and objects with relational phraselets*, in *Proceedings of the 12th European conference on Computer Vision - Volume Part IV*. 2012, Springer-Verlag: Florence, Italy. p. 158-172.

211. Yin, Z., et al. *Action recognition in still images using a combination of human pose and context information*. in *Image Processing (ICIP), 2012 19th IEEE International Conference on*. 2012.

212. Li-Jia, L. and F.-F. Li. *What, where and who? Classifying events by scene and object recognition*. in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. 2007.

213. Yao, B., A. Khosla, and L. Fei-Fei, *Classifying Actions and Measuring Action Similarity by Modeling the Mutual Context of Objects and Human Poses*, in *Proc. Int'l Conf. Machine Learning*. 2011.

214. Gupta, A., A. Kembhavi, and L.S. Davis, *Observing Human-Object Interactions: Using Spatial and Functional Compatibility for Recognition.* Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2009. **31**(10): p. 1775-1789.

215. Yao, B. and L. Fei-Fei, *Action recognition with exemplar based 2.5d graph matching*, in *Proceedings of the 12th European conference on Computer Vision - Volume Part IV*. 2012, Springer-Verlag: Florence, Italy. p. 173-186.

216. Maji, S., *Large Scale Image Annotations on Amazon Mechanical Turk*. 2011, EECS Department, University of California, Berkeley.

217. Maji, S., L. Bourdev, and J. Malik. *Action recognition from a distributed representation of pose and appearance*. in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. 2011.

218. Ramakrishna, V., T. Kanade, and Y. Sheikh, *Reconstructing 3D Human Pose from 2D Image Landmarks*, in *Computer Vision – ECCV 2012*, A. Fitzgibbon, et al., Editors. 2012, Springer Berlin Heidelberg. p. 573-586.

219. Riesen, K. and H. Bunke, *IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning*, in *Structural, Syntactic, and Statistical Pattern Recognition*, N. da Vitoria Lobo, et al., Editors. 2008, Springer Berlin Heidelberg. p. 287-297.

220. Everingham, M., et al., *The Pascal Visual Object Classes Challenge: A Retrospective.* International Journal of Computer Vision, 2014: p. 1-39.

221. Everingham, M., et al. *The PASCAL Visual Object Classes Challenge 2011 (VOC 2011) Results (2011).* in *URL http://www. pascal-network. org/challenges/VOC/voc2011/workshop/index. html*. 2011.

VITA

Thanh Thieu earned his Master degree in Computer Science from University of Missouri - Columbia, USA in December 2011. He has publication in top Bioinformatics journal and contributed in a book chapter. Before that, he earned his Bachelor degree in Information Technology from Hanoi University of Science and Technology, Vietnam in July 2006. His academic endeavor has been focusing on advancement of computational intelligence and application in natural language processing and artificial vision.