Algorithms for the Compression of Graphical Traces

This section defines the formal problem of compressing a dataset of graphical traces, assumed to have been generated from some underlying graphical process model, M. The ability to induce the graphical structure of a process from trace data may seem of limited interest outside of process mining, and business process management, or similar operations fields. However, the induction of graphical structure from data encompasses many classical formal problems in machine learning, planning, and artificial intelligence, as will be discussed.

Consider a planning problem of a robot navigating two-dimensional space, using elementary reinforcement-learning formalisms. Assume the robot has four discrete actions {up, down, left, or right}, the environment is deterministic and known, the transition model may or may not be known, and the robot’s task is to navigate to some positive reward location while avoiding obstacles with negative rewards. These problem formulations are amenable to a wide variety of instances in a class of algorithms modeling the sequential one-step dynamics of discrete action sets, for relatively modestly sized action sets. Many canonical solutions are rooted in Bellman equation formalisms [CITE Sutton] or Monte Carlo methods.

Now instead consider that the rewards and system dynamics are determined by *k*-step bounded dynamics, where the robot achieves goals only by executing specific sequences of actions of length i, where : {up,up,down}, {up,up,down,left}, et cetera. These representations mimic real-life tasks, wherein actions possess long-range interdependencies, and tasks recursively decompose to subsets of actions, or “subtasks”. Moreover, such tasks often can only be described graphically, by process model formalisms such as Petri Nets [CITE]. These models also violate the clean, Markovian one-step dynamics often required by classical reinforcement learning formulations, and often result in exponential complexity. For instance, the long-range activity set of action sequences in the above example is exponential in the number of activities and the bound on sequence lengths, or 4^k, an intractable space even for this modest activity set, and without any assumptions about the complexity of the state space.

Such problems are the domain of classical planning problems, such as the block world domain [CITE]. More accessible examples are the games of Go and Chess, with their high branching factor and multiple game strategies composed of long-range dependencies between actions. In these problems, the action space, action sequence space, state space, or combinations of these are intractably large for traditional, sequential learning formulations. Current approaches often implement approaches such as Monte Carlo Tree Search [CITE], which still entail rather intensive search behavior before upper confidence bounds on action-value estimates yield satisfactory performance.

Graph compression addresses these problems heuristically by estimating the subgraphs characterizing advantageous behavior. For instance, in a highly complex combinatorial space such as chess, sub-graphs of useful actions can be learned from user examples. Likewise, non-adversarial everyday tasks such as performing home chores or driving may be learned from examples which characterize the underlying graphical model of the task. Given lots of such user data for this or similar planning problems, repeated subgraphs can be extracted and used to bootstrap learning algorithms to bias their activities toward these structures.

Thus, many planning problems can be reduced to the problem of learning graphical representations of processes and subtasks. These representations can then be used to more efficiently learn complex behavior from compositions of subgraphs representing subtasks within some domain. In this manner, methods for compressing and extracting structural patterns from graphical data, have general application in machine learning, planning, and artificial intelligence. [CITE SOME STUFF: Deep convolutional nets, planning, etc.] The ability to learn graphical structures provides the ability to learn combinatorial tasks of arbitrary complexity, a general task-learning and problem-solving goal.

The Optimization Problem of Graphical Data Compression

In this work, we wish to compress a set of graphical data representing directed, possibly-cyclic traces generated from some underlying graphical model. In the domain of process mining, these models typically have regular, compression-favorable properties, such as having defined *begin* and *end* points for traces, modest overall average degree, few or zero cycles, and generally, if not always, behaving like directed, acyclic graphs (DAGs).

The goal of graphical trace compression is to reduce a set of graphical traces to a minimal collection of prototype subgraphs , edge subsets, maximally compressing a log, , of such traces:

Once the set is found, each trace can be encoded as a binary vector indicating its subgraphs, and thus the trace can be encoded and decoded via and . Thus, a lossless method of compression can be devised by finding the minimal , converting each trace to a bit vector indicating its subsets in , and transmitting these vectors along with the edge-subsets by which to decode them.

A Naïve, Illustrative Problem Formulation:

The formal problem of finding the set reduces to the process of iteratively selecting columns from the unfolded adjacency matrix of all vertices in the directed graphical data. For a dataset in the form of a trace log, , of size , each trace is a subgraph of the super-graph composed of the union of all traces. Put simply, represents the behavioral model including all traces, which is simply the union of all edges in . Each trace can be represented as an adjacency matrix whose rows and columns are composed of the set of vertices in . Concatenating the rows of this matrix gives a binary vector whose non-zero components indicate the directed relations (edges) present in the trace. Thus the entire set of traces can be converted to a matrix, of such vectors.

[diagram]

This data representation is illustrative because it demonstrates the dimensionality of the input space, which is quadratic in the number of vertices.

Discovering the maximally compressing subgraph within this data representation resolves to finding the largest subset of columns containing all 1’s that encompasses the greatest number of rows. Such a column set of edges represents a subgraph , if connected. The size of the set represents the size of the subgraph in edges; likewise, the number of rows containing is its frequency. This introduces a compression tradeoff between size and frequency: it would be easy to find a very large and infrequent subgraph, or conversely a very small but frequent subgraph (such as a single edge). Consider two candidate prototypes, and : has and , whereas has and . Which of or should be chosen for optimal compression? How do criterions of and affect the optimality of the resulting set ?

Worse yet, since candidate subgraphs are not disjoint, dependencies exist between the selection of compressing subgraphs such that the selection of a prototype at the *t*-th iteration can affect which candidates are be available in subsequent iterations, and may affect compression. Optimal compression is defined as minimizing the description length of the trace-graph codes sufficient to losslessly reproduce (decode) all trace-graphs from their encodings via . Due to the tradeoffs between prototype subgraph size and frequency, maximally encoding the subgraphs requires making the correct sequence of decisions per the size and frequency of each prototype subgraph. This problem is akin to bin-packing (Korte, 2008), an NP-hard combinatorial problem, but harder due to the dependencies between prototype selection. Loosely, each prototype’s size and frequency define the object’s abstract dimension, while the objective is to fit as many of these objects as possible into the smallest bin. Fortunately, the optimal formulation of this problem is not the subject of this work.

The max-size compressing subgraph can be found at each iteration by searching the columns of this matrix for the largest number of contiguous rows of all 1’s. For any fixed collection of columns, all traces must be traversed to quantify the number of columns for which the conjunction of all columns evaluates to true. Additionally, there are possible subsets of columns. Defining the set of all edges in graph as , the set of all vertices and using the identity (Cormen et al, 2001), the complexity of this procedure for is given by:

Thus, this column selection procedure is exponential in the number of edges, which in the worst-case, for a fully-connected and non-reflexive graph, is:

Hence,

This search procedure illustrates the problem’s structure, and brute-force complexity. But it is flawed since it needlessly searches over all combinations of subsets of edges of size *i*, whereas we are only interested in connected components of the graph. Graphical problems frequently involve sparse graphs, so we can expect to reduce complexity by restricting iterations to the subsets of columns representing connected components, which can be enumerated using basic, elementary graph-search procedures. Lastly, real graphical data typically has high redundancy, such that can be reduced significantly by a de-duplication strategy of storing each unique row with its frequency.

The Heuristic View

In a relaxed version of the trace-graph compression problem the encoding need not be optimal, but may instead use a heuristic to generate an approximately optimal encoding much faster than the optimal problem. Such procedures can still be lossless, where any trace can be completely reconstructed from its encoding and the set to decode it. Many such heuristics are possible, since the task is relaxed to that of iteratively finding frequent subgraphs under some encoding criterion, such as favoring the size or frequency of graphs. Many graph problems involve graphs with node attributes or other additional information, which may also be incorporated into the information theoretic definition of their encoding.

Notably, the data representation in the prior section is amenable to a wide range of supervised and unsupervised learning approaches. Unsupervised approaches, such as neural autoencoders, provide great promise in terms of automating the entire process of hidden pattern discovery. This framework trains neural network using the input as the target output. Using a variety of training and architectural strategies, these networks learn the hidden structural patterns of the data, by which normative and anomalous patterns can be determined. A recent example is given by (Nolle et al, 2016), in which the authors used denoising autoencoder model for both anomaly detection and normative pattern discovery, though the authors presented the traces to their networks as linear activity sequences, rather than sparse adjacency matrices. The authors report their method perfectly split the trace log into normal and anomalous traces. Similar work is possible by using recursive/RECURRENT neural networks by presenting the traces to the network as linear activity sequences (CITE RNN’s).

Supervised learning models can also be adapted to unsupervised pattern discovery. By appending a +1 to each binary input vector as a dummy target “output” for a learning model, the unsupervised data can be adapted to a supervised-learning data representation. Each vector can likewise be replicated by its negation (possibly with additive noise), to generate a semi-synthetic supervised-learning dataset that divides the input space into two classes: positive examples, and synthetic examples sampled outside the set via some distribution that facilitates or improves a particular learning model. Some distant examples of such data extension/generation strategies are the negative sampling used by some implementations of the word2vec algorithm (Mikolov et al, 2013), various structured learning algorithms like the DAgger algorithm (Bagnell, 2015), or (very distantly) the generative adversarial networks of (Goodfellow et al, 2014).

The benefit of such a representation is that many supervised learning models have been developed, especially generative ones, by which normative patterns or other model parameters can be learned to perform secondary tasks like anomaly detection or normative pattern extraction. The simplest example i to run linear regression on the preceding semi-synthetic data. The result output is a weight vector whose non-zero components correspond to the collection of edges which maximally “compress” the data the most, by minimizing the mean-squared error (MSE) loss. The corresponding columns would then be removed from the input data, and the procedure would be re-run on the remaining examples to find the next set of such edges, and so on, until the data is completely compressed. Notably, the edge collection found on any iteration might not represent a connected subgraph, but regularization strategies might be devised to bias the learning algorithm toward connected components, rather than disconnected subsets of edges.

SUBDUE

While many data representations and strategies are left to be explored, this work’s primary focus is on the SUBDUE method for discovering the maximally compressing components of graphical input data. In contrast to matrix-based graph-data representations, SUBDUE is search-based and focuses on the vertex perspective to search for compressing substructures. In this manner, SUBDUE proceeds by “growing” candidate substructures within some search beam of size k, and maintaining only the most-highly compressing components in the beam at any time. Compression is measured by the reduction in the minimum-description length of the data with respect to compressing components.

From this perspective, compressing components are found not by solving a brute-force global search over edges, but rather by growing compressing components from the neighborhood surrounding promising looking nodes.

SUBDUE fits neatly into these purposes, since the algorithm compresses not just based on the frequency of a subgraph, but also some metric of its encoded length ????

[read and cite stuff]

Bagnell, J. A. (2015). *An invitation to imitation* (No. CMU-RI-TR-15-08). CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to algorithms second edition. p.102.

Korte, B., & Vygen, J. (2008). Bin-packing. *Combinatorial Optimization: Theory and Algorithms*, 449-465.

Nolle, Timo Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders (2016)

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).