Algorithms for the Compression of Graphical Traces

This section defines the formal problems of compressing a dataset of graphs, assumed to have been generated from some underlying 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-oriented fields. However, the induction of graphical structure from data encompasses many classical formal problems in machine learning, planning, and artificial intelligence.

Consider a planning problem of a robot navigating two-dimensional space, using elementary reinforcement-learning formalisms. Assume the robot has four discrete actions {move: up, down, left, or right}, the environment is deterministic, 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 algorithms in a class that models the one-step dynamics and discrete action sets, for relatively modestly sized action sets, often rooted in Bellman equation formalisms [CITE Sutton].

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 required by the algorithms often used in 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 intractably large space even for this modest activity set, and without any assumptions about the complexity of the state space either.

Such problems are the domain of classical planning problems, such as the classical 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 huge for traditional learning formulations.

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, in special cases, planning problems can be reduced to the problem of learning graphical representations of processes (tasks). 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 structural features provides, essentially, 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 all 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.

[diagram]

Discovering the maximally compressing subgraph within this data representation resolves to finding the largest subset of columns containing all 1’s encompassing the greatest number of rows. Such a column set (if connected) represents a subgraph . 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, which 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 equals one. Additionally, there are (n choose i) 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 is illustrative of the problem’s structure, but 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 subsets of columns representing connected components. Enumerating the set of connected components for a graph can be done using basic, elementary graph search procedures.

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.

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]

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.