Unravelling Graph Interchange File Formats

Matthew Roughan

ABSTRACT

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*graphs, file format*

1. INTRODUCTION

A short search of the Internet revealed that there are well over 60 formats used for storage and exchange of graph data: that is networks of vertices (nodes, switches, routers, ...) connected by edges (links, arcs, ...).

Every new tool for working with graphs seems to come with its own new graph format. There are reasons for this: new tools are often aimed at providing a new capability. Sometimes this capability is not supported by existing formats. And inventing your own new format isn't hard.

More fundamentally exchange of graph information just hasn't been that important. Standardised formats for images (and other consumer data) are crucial for the functioning of digital society. Standardised graph formats affect a small community of researchers and tool builders. But this community is growing, and the need for interchange of information is likewise growing, particularly where the data represent some real measurements which are hard to collect when and as needed, and so scientists need to be able to share.

So the current state of affairs is ridiculous. The existing formats do include many of the features one might need, and some are quite extensible, so the bottleneck is not the existing formats so much as information about those formats. This is the gap this monograph aims to fill.

Many of the formats presented may seem obsolete. Some are quite old (in computer science years). Some have clearly not survived beyond the needs of the authors' own pet project. However, we have listed as many as we could properly document, partially for historical reference, and partially to show the degree of reinvention in this area. But more importantly, because old and obscure isn't bad. For instance NetML, a format that doesn't seem to be used at all by any current toolkits, incorporates some of the most advanced ideas of any format presented. A good deal could be learnt by current tool builders if they were to reread the old documentation on this format.

It is important to note that this paper does not present yet-another format of our own. It is common, in this and other domains, for the discussion of previous works to be coloured by the need to justify the authors' own proposal. Here we aim to be unbiased by the need to motivate our own toolkit, and so (despite temptation) do not provide any such.

We do not argue that new graph formats should never be developed. In some applications new features are needed that are not present in the existing formats. However, it is critical that those who wish to propose new ideas should understand whether they are *really* needed, or whether existing tools provide what they need. Moreover, in studying the existing formats, and their features, we learn what should be required in any new format to make it more than a one-shot, aimed at only one application.

2. BACKGROUND

A mathematical *graph* $\mathcal{G}$ is a set of nodes (or vertices) $\mathcal{N}$ and edges (or links or arcs) $\mathcal{E} \subset \mathcal{N} \times \mathcal{N}$.

Graphs (alternatively called networks) have been used for many years to represent relationships between objects or people.

There are many subtypes of graphs, and generalisations, some of which we shall mention below.

Additional information is often added to a graph: for instance

- node or link labels (names, types, ...);
- values (distances, capacity, size, ...); or
- routing (paths taken when traversing the graph).

It has been necessary for many years for researchers in sociology, biology, chemistry, computer science, mathematics, statistics and other areas to be able to store graphs, and share them. They have done so by sharing files. As a result portable file formats for describing graphs have been around for decades.

This document is concerned with providing information about these formats, specifically with the intention of moving towards a smaller number of standard formats (the current trend seems to be progressing in the other direction).

3. DESCRIPTORS AND DISCRIMINATORS

In order to describe the formats we will consider here, we need some simple means to compare and contrast. Of a necessity, these will oversimplify some of the issues. For instance, some features that exist in principle in the format may not be supported in any software.

What's more, many descriptions of file formats are imprecise. It is common to describe the format by reference to examples. Although useful for simple cases, these leave out important details: for instance: the character set supported, and even more surprisingly, the format of identifiers. It is often vaguely suggested that these are numbers, but without formal definition of what is allowed (presumably non-negative integers, but are numbers outside the 32 bit range supported?).

In the following, we make the best estimate of the capabilities of each format through reference to online documentation, and through a survey of the file format creators[1]. In many cases the results are inferences, so in this section we will outline the features we describe, and the assumptions made in compiling our data.

There are three main types of descriptors here:

file type : these are simple issues of the type of file storing the data: binary vs ASCII, etc.

graph types : this refers to the nature of the graph data that can be stored.

attributes : these are features related to supplemental data about nodes and edges, such as labels and values associated with these.

general : this is a grab bag for additional features that don't fit in either of the previous classes.

We'll describe each of these in detail below, and then provide a table of the features vs file formats.

One last point, this is not intended as a pejorative list. We do not mean to imply that having a feature is good or bad (though vague descriptions do seem fairly unhelpful). The aim is to provide potential users with the background to choose the right format for their purposes.

3.1 File Type

binary/text : this is, in principle, a simple distinction in file type. However, text files today can use multiple different character sets, and this is important because some graphs will be labelled with non-English character sets. However, the majority of file-format definitions leave unspecified the character set to be used. We assume here that the character set is ASCII, unless there is some indication otherwise, either an explicit statement, or in the case of applications of XML it is assumed that the character set supported is Unicode.

representation : there are quite a few methods to represent a graph:

matrix : This is simply the graph's full adjacency matrix.

edge : This is a list of the graph's edges.

smatrix : The matrix representation is poor for sparse graphs, which are common in real situations. However, some tools actually store a sparse matrix, which is almost equivalent to an edge list. There is a subtle difference in that a matrix view of the edges in a network cannot contain much detail about the edges (only one number), and so we have a separate name, *smatrix*, for formats that use this type of representation.

neighbour This is a list of the graph's nodes, each giving a list of neighbours for each node.

path : One can also implicitly represent a graph as a series of *path* descriptions (essentially a path is a list of consecutive edges). This could be useful, for instance, with a tree or ring.

Moreover, graph data is often derived from path data, i.e., a series of paths are analysed, and the edges on these become the graph. In other cases, one might like to store path information, for instance related to routes used on a graph, along with the graph.

constructive : Graphs can often be described in terms of mathematical operations used to construct the graphs: for instance graph products on smaller graphs [1]. See [2] for a description of "levels" of graph formats.

Apart from simple incremental construction, the only format that seems to allow this is NetML [2].

procedural : Many graphs can be concisely defined by a set of procedures, rather than explicit definition of the nodes and links. This type of graph format could be very concise, but verges on creating another programming language. In fact, many graph libraries for particular programming languages essentially provide this, but obviously in a non-portable (between languages) way.

The only *generic* (language independent) format that seems to allow this is NetML [2].

Any procedural approach admits the possibility of defining a method for constructive graph description, but we do not automatically count any procedural approach as constructive, unless it provides explicit graph-related operations as part of the toolkit.

These representations are given varying names in the literature, but we use the names above to be clear.

The representation is important: for a graph with N vertices and E edges, the adjacency matrix requires $O(N^2)$ terms, the edge list $O(E)$ terms, and the neighbour list $O(N+E)$ terms. However, the terms in a matrix are $\{0, 1\}$ whereas the terms in the edge and neighbour lists are node identifiers (consider they might be 64 bit integers), so the size of a resulting file based on each representation depends on many issues, including the way the data is stored in the file. No approach is universally superior.

Moreover, some may be easier to read and write: for instance a neighbour listing may be slightly more compact than an edge list, but it has the same number of

[1]For which purpose this current document is being used.

elements per line, potentially making it easier to perform IO in some languages.

Some graph file formats allow alternative representations, and so we list all that are possible. It seems rare (for obvious reasons) to allow a mixed representation.

structure: we use this field to describe how the file format's structure is defined. The cases are:

simple : the typical approach to create a graph format is to use one line per data item (a node, an edge, or a neighbourhood), with the components on a line separated by a standard delineator (a comma, tab, or whitespace). There are many variations on this theme, some more complex than others, for instance including labels, comments or other information. These formats are usually specified by a very brief description and one or two examples. They rarely specify details such as integer range or character set.

intermediate : this is a slight advance on a *simple* file format, in that it includes some grammatical elements. For instance, the file may allow definition of new types of labels for objects. However, in common with simple files, these are usually only specified by a very brief description and one or two examples, not a complete grammar.

BNF : means that the file format is described using a grammar, loosely equivalent to a Backus-Naur Form (BNF). This is perhaps the most concise, precise description. When done properly it precisely spells out the details of the file in a relatively short form.

XML, JSON, SGML, ... : many file formats extend XML, JSON, SGML, or similar generic, extensible file formats. This is a natural approach to the problem, and allows a specification as precise as BNF, though only through reference to the format being extended. Thus it is precise, but sometimes rather difficult to ascertain all of the details, unless one is an expert in XML, etc.

On the other hand, these approaches draw on the wealth of tools and knowledge about these data formats. On the other hand again, to use those tools the model of your graph object has to map to the XML model (or at least be easily transformed into that form).

Tcl, Lisp, ... : as noted above one approach to defining a graph is procedural. Most of the approaches that allow this are extensions or libraries for common programming languages.

We will not list every programming language and library as a data format though because, generically, such approaches are not portable between programming languages. We do mention a few formats though (cypher, ns-2 and S-Dot), because translators exist from/or to these from other data formats.

single or multiple files : most data files are a single file, but some formats require multiple files, for instance, a separate files for the lists of nodes and edges. Other formats allow supplementary information in additional files, so multiple files aren't mandatory. We have only classified the files by whether multiple files are allowed, not whether they are mandatory (because the later requires a distinction about what mandatory would mean: does it mean they are required to support basic features or advanced features?)

must be ordered : most files have some requirement for ordering, for instance a header, or tags around data, so we are not concerned by that aspect of ordering. We are concerned with whether elements of the actual data must be presented in some type of ordering. For instance, do nodes have to be defined before we can create an edge joining them? This is often unspecified, particularly in simple formats, so we draw conclusions from the examples presented.

The decision to list a particular file as ordered can seem a little arbitrary, but the importance of it is really whether the file can be read in one (simple) pass or not. Ordering (e.g., defining nodes before using them) can potentially help in a one pass read. This is the information we are interested in expressing.

integral meta-data : meta-data is data about the graph: for example, its name, its author, the date created, and so on. This is very important data, but many formats provide no means to include it in the file, and instead rely on external records. We refer to meta-data as *integral* if it is contained in the file itself.

Some formats allow meta-data through unstructured comments. This is better than nothing, but lack of structure of the comments means these are not machine readable, in general.

Some file formats provide only a limited range of meta-data fields, whereas others are arbitrarily extensible. At the moment we don't distinguish between these as it is difficult to tell the difference in many cases.

built-in compression : it is easy enough to compress a graph-file using common utilities such as gzip, and typical compression ratio will be reasonably good as graph files often have many repeated strings. However, one format provides for compression of the graph as it is written, in much the way image file formats allow intrinsic compression of the image. Such an approach requires a graph-compression algorithm, though two other formats provided some crude mechanisms to reduce the size of the file.

3.2 Graph Types

directed/undirected : The two basic forms of graph are the directed and undirected graph. In the former edges (or arcs) imply a relation from one node to another. In the later an edge implies a relationship in both directions.

Some graph formats specify one or the other; others allow the user to specify which; and the most general allow the user to specify directed (or not) for each edge.

The difficulty with this is that many graph formats fail to specify anything. We assume that, in the absence of explicit statements to the contrary, a graph format is

directed if the edges/arcs are specified by source/target or from/to, or some other directional nomenclature. We also assume that matrix formats are directed unless there is specific mention of mechanism to represent the upper triangular part of the matrix alone.

Finally, in one case, the format is explicitly restricted to DAGs (Directed Acyclic Graphs).

multi-graph : a multi-graph is a graph generalisation that allows (i) self-loops, and (ii) more than one edge between a single pair of nodes.

Some formats specifically allow, or disallow multi-graphs. A few allow loops, but not multi-edges. Many, however, say nothing on the topic. We assume in this case that formats presenting either matrix or neighbour representations don't allow multi-graphs. It is technically possible to represent a multi-graph in these cases, but this would require special processing of the information, and unless we see an indication this is present we assume it is not. Edge lists, however, can easily cope with multi-graphs. We suspect it is left to the software supporting the data format to make a decision about how to deal with these cases, and the decision may be inconsistent between supporting software. Hence it seems important that when an edge-based representation leaves the question unspecified, we say this.

hyper-graphs : a hyper-graph allows edges that connect more than two nodes. These are useful for some problems: for instance indicating a multi-access medium in a computer network (such as a wireless network).

Support for hyper-graphs requires specialised data structures for hyper-edges, so unless a format explicitly states it can support these and presents the mechanism we assume it cannot.

hierarchy : it is common for graphs to have sub-structure, for instance nodes that themselves contain graphs.

Several formats provide mechanisms to record this sub-structure. Unfortunately, there does not seem to be a consistent definition of this type of structure, and so we see differences not just in the representation, but also what exactly is being represented.

We don't try to list this level of detail here though, we simply note whether the format provides this feature.

meta-graph : a meta-graph [3] is a generalisation of a graph, multi-graph, hyper-graph, and hierarchical graph. As far as we know, no format yet supports meta-graphs[2], but this is included as a feature as an indication of the type of feature that might require a new format, or extended version of an existing format.

3.3 Attributes

edge weights : a very common requirement is to store a numerical value associated with an edge. Generically, we call this a weight. Many formats provide the facility to keep one such value.

[2]Note the term "meta-graph" is somewhat overloaded, e.g., there is at least one package called *metagraph* that has nothing to do with the mathematical meta-graph.

multiple attributes : Some formats allow one to keep multiple labels (numerical or otherwise) for each node and/or edge.

For some formats these are fixed (e.g., they allow a name and a value), whereas others allow arbitrary lists of attributes. We don't yet distinguish these two cases.

default values : specifying the value of a weight or attributed for every edge or node can be laborious (if it has to be done by hand), and wasteful of space. Moreover, it makes it hard to see structure in the data. Simply providing a default value for the common case can improve the situation. We include here the case of simple inheritance of values through a tree of "class" structures on the objects. For instance, nodes can be given a type which conveys a default value to be overridden by a more specific type or particular value. Notice here we are no speaking of inheritance through the graph itself, but a structure on top of the graph.

multiple inheritance : a few formats allow values to be derived through multiple inheritance of values from multiple classes the belong to. Thus allowing a node to have, for instance, a type "router" which conveys that it is an Internet router, with appropriate characteristics for such a device, from vendor "Cisco" which appropriate characteristics for that vendor.

Once again, inheritance is not through the structure of the graph, but through a further structure defined on the graph objects.

visualisation data : files that allow multiple (extensible) attributes can always provide data to be used in visualising the graph, but here we refer to formats that explicitly provide such data. It needs to be explicit because it has a particular use in software, different from the use of more arbitrary associated data such as labels.

The level of visualisation data varies dramatically: some formats only allow position information for nodes, whereas others allow SVG definitions to be used in drawing the nodes. Still others provide guidance about which layout algorithms to use in displaying the graph.

There is not space here to document all of the variations possible, so we simply indicate whether any such data is defined or not.

ports : this is a specialised piece of layout information: often ports are specified by a compass direction, and indicate where on a node the link should join to it. We include it separate to the previous field because port-based information can also carry semantic information about the relationship between links on a complex node: e.g., the arrangement of links on a real device like an Internet router.

temporal data/dynamics : a topic of recent interest is analysis/visualisation of graphs as they change. One way to store this information is as a series of "snapshot" graphs, but storing it all together in the same file has some appeal. A few formats provide some variant on this: allowing links or nodes to be given a lifetime, or proving "edits" to the graph at specific epochs.

3.4 General

extensible : some formats allow extensibility in varying forms. We only consider them to have this facility, however, if they provide an explicit mechanism. For instance, we do not regard all XML derivatives as intrinsically extensible because they could, in principle, be extended using standard XML techniques. The format has to explain the explicit mechanism whereby it is extended.

Simply adding extra attributes is not considered extensibility.

schema checking : a format that provides an explicit mechanism to check that a file is in a valid format is useful. We only say it has this facility if a tool exists to perform the check (a schema-checking program, DTD, or other similar formal tool).

checksums : It is possible for large data files to become corrupted. A common preventative (or at least check for this problem) is to use a checksum. This is possible for all files, but we say that a given format has this capability if it includes it as a internal component (usually checking everything except the checksum itself). Only a few formats contain this check.

external data references : Some formats allow reference to external files. This could be for visualisation data, meta-data, or other purposes. Again, we look for an explicit explanation of the mechanism, not a generic belief that it is inherited from the parent file format.

multiple graphs : Some formats allow multiple graphs to be held in one file. Again, we only count this as a feature if the specification explains how explicitly.

incremental specification : A small number of formats that present multiple graphs allow these graphs to be specified incrementally. This is subtly different from including temporal dynamics, as there is no implication of time, and the different graphs could potentially be unrelated (for instance, this might be used to describe graph edit distance problems).

In a sense incremental specification is a simple case of constructive graph definition, but it is a very limited case, with specific application, so we list it separately.

4. THE FILE FORMATS

This list is incomplete. There are some formats that we have seen mentioned, but been unable to find documentation for (e.g., Gem2Ddraw). There are undoubtedly others that we have missed, and there are a few that we have lumped together under the general heading of "Adj" because they provide a simple delimited edge list.

Moreover, we have deliberately omitted generic file formats that could, in principle, contain a graph: e.g., XML, JSON, SGML, RDF, Avro, YAML, and so on, unless there is a specific extension of these designed to provide support for graphs, in which case we list the specific not the generic. For instance, several software tools say that they can read/write JSON or other generic serialisations of data, but without details of exactly what is being serialised, then these are not useful interchange formats.

We also aim to avoid, for simple practicality, formats that represent data that has a graph structure, but whose main content is not the graph. For instance the HTML WWW: the graph structure of this is vastly smaller than the content and HTML is intended to store both in a distributed fashion. If one wished to represent the graph of the WWW, then another format seems indicated. Other examples include SBML (the Systems Biology Markup Language), and FOAF [4].

The attached spreadsheet contains the currently known formats, and a first draft of features for these formats. It is to be considered unverified and potentially highly inaccurate at this point.

Additional formats are welcome, but we plan to add only those which are aimed at portable data exchange, not internal data formats for use only within one tool (unless the internal format illustrates a particular issue very well).

5. DECISIONS

The list above is not intended to be pejorative. However, it is inevitable that potential users need to make decisions about which format to use. There are several issues that need be considered in such a decision, and although the first is the feature list required, there are others:

data size : the size of the graph data to be recorded and used is an important factor in file format decisions. This is sometimes glossed over when XML-style formats are considered: these are very redundant formats, and hence much larger than needed, but they compress well. Hence, the compressed version may be no longer than a tighter initial specification. However, the issue of read/write time (and indeed compression/decompression time) still depends greatly on the format's wordiness. Large graphs need tighter formats: either binary formats, or at least those that avoid unnecessary bloat.

On the far end of the spectrum is the possibility of graph-specific compression being part of the storage process (much as many image formats provide image compression as an integral features). Only one format we found provides true graph-based compression: BV-Graph.

edge density : edge density affects the choice of best representation of a graph. Very sparse graphs are best represented by edge lists, moderately sparse graphs are (perhaps) slightly better stored as neighbour lists, and dense graphs may be better stored as a full adjacency matrix.

access method : most graph formats are designed to be read serially directly into memory in their entirety. None we saw provided support for random (or indexed subgraph) access to part of a graph. Few formats seem to specifically address this, though some public databases of graph data provide interfaces for querying components of the graph.

In other cases, a single graph might be part of a larger database, and this seems only to have been addressed by ad-hoc mechanisms.

human readability : Implicitly we need the file to be machine readable, but a file that is more easily compre-

hended by humans is potentially better because it is easier to enter and check. This might seem strange to consider, but many of graph examples were entered at least in part by hand. Human readability requires a text file in a logical format, but it also needs to avoid: (i) bloat, which distracts the reader with unnecessary text, and (ii) the file to be organised neatly. XML formats often fail on these: the first because of the volume of tags, and the second because they allow organisations which are unreadable, e.g., with all the text on one line.

Ultimately, human readability is a highly subjective criteria. Some people may find XML easy to read, and others get distracted by the tags. As such, we won't comment on it further here.

documentation : Through compiling the information used in this paper it has become obvious that a key limitation of many formats is incomplete documentation. Hidden assumptions, specification by (limited) examples, and/or documentation by source code are all common. Ideally, any truly portable format should have a complete, highly-specific schema; human readable documentation (with examples); and source code. All of these together provide the ideal documentation.

support : Finally, the support for the format in a variety of tools is a crucial requirement for exchange of data. Likewise, support for formats in a variety of public databases makes it more useful. We shall consider this issue in more detail below.

5.1 Software Support

The most difficult issue surrounding software support is that with vague specifications, a piece of software may notionally support a file format, and yet still be incompatible with other software notionally supporting the same format.

For instance, software might

- be able or unable to cope with multi-graphs;
- fail to accept integers outside a particular range;
- have varying case sensitivity;
- be unable to read the right character set;
- be unable to read strings beyond a particular length (very few formats specify buffer or string lengths); or
- fail to cope with files larger than some size.

Size is interesting, because almost no documentation exists for size limits for any data formats. However, it should be reasonably obvious that if 32 bit integers are used, then the largest number of (integer) identifiers is around 4 billion. In the past this was large enough that the need to specify it may have seemed small. With today's graphs, this could be an important limitation.

Even more pernicious is partial support for a format. Even when documented this makes our job hard, but partial support is not often documented. Instances include:

- hyper-graphs supported in the format, but not in software; or
- some small number of formats make mention of allowing complex numbers; or
- partial support for hierarchy (i.e., the file can be read, but the subgraph structure is not retained).

Even more complex is the fact that some features may be supported on read or write, but not both.

Generally, software is designed for a particular purpose, and when a feature falls outside that purpose, the software often fails to support it.

Our goal here is not to criticise software, only to provide some guidance over the level of support for a format, and the visibility this format has in the general graph-research community. So instead of determining the exact details of support for each format, in each piece of software, we will report the software authors' claims of support only. The goal is not to report technical detail so much as the level of awareness and interest a format engenders.

See the attached spreadsheet for a list of software and the supported formats.

The list of software is long, but is still a tiny cross-section of the graph toolkits provided today. We have sampled primarily on the results of simple web searches, but also when a format was created for a particular piece of software, we have included that software.

5.2 Public DB support

The other type of support we might wish to see is general support amongst those who provide data publicly. There are many public databases that provide example networks for benchmarking or research. We provide a list of some of the better known of these with their format choices.

6. TO DO

Evaluate criteria given, and expand on a couple which could use more detail (e.g., extensible, vs static meta-data).

Check details again.

Other features/descriptors we might like to include:

self-describing : this format data isn't filled in yet (as it is yet another rather rubbery term when dealing with imprecise formats), but it refers to whether a file provides its own definition of its format.

7. CONCLUSION

The science of graphs and networks needs portable, well-documented, precisely defined, exchange formats. There are many existing formats, and this paper seeks to unravel this mess, most notably with the aim of reducing the number of new formats developed.

One size probably does not fit all though. There is a clear need for at least three major types of file format:

- a general, flexible, extensible approach such as GraphML;
- a quick and dirty approach that satisfies the least common denominator for the exchange of information to/from the simplest software; and
- a very efficient (compressed) format for very large graphs.

Its not clear that any format at present has a complete enough list of features to take the roll of the first format. No

doubt this will continue to evolve as well, as new features are required.

The second is easy, but there are very many contenders, and settling on one will be hard.

The final one should be seen as an interesting research topic.

Maybe what is needed is actually a container format: allowing specification of parts of a graph in alternative formats. Or allowing specification of meta-data and labels in an XML-like format, but the edge data in a more compact form.

8. REFERENCES

[1] E. Parsonage, H. X. Nguyen, R. Bowden, S. Knight, N. J. Falkner, and M. Roughan, "Generalized graph products for network design and analysis," in *19th IEEE International Conference on Network Protocols (ICNP)*, Vancouver, CA, October 2011.

[2] V. Batagelj and A. Mrvar, "Towards NetML: Networks markup language," in *International Social Network Conference*, London, July 1995, vlado.fmf.uni-lj.si/pub/networks/netml/snetml.pdf.

[3] A. Basu and R. W. Blanning, *Metagraphs and their Applications*. Springer, 2007, http://link.springer.com/book/10.1007%2F978-0-387-37234-1.

[4] "FOAF: friend of a friend," http://www.foaf-project.org/.