Scale-Free Geometry in OO Programs

Though conventional OO design suggests programs should be built from many small objects, like Lego bricks, they are instead built from objects that are scale-free, like fractals, and unlike Lego bricks.

When executed, OO programs produce a complex web of objects that can be thought of as a graph, with objects as nodes and references as edges. From physicists to biologists to computer scientists, interest has been increasing in the geometry of networks, particularly those of human origin. Many such networks show a rather striking property: scale-free geometry. In the case of the Web, for example, the number of Web pages with two incoming links is half the number of pages with one incoming link. Then there are half as many pages with four links as there are with two links, and so on, all the way down to Google and other massively referenced sites [2]. We say the Web has a "scale-free" network geometry (the phrase reflects the fact that if we double the number of links n, the number of pages is always halved—or some other fixed ratio—regardless of the value of n. Scale-free geometry is very different from the geometry of a graph in

By Alex Potanin, James Noble, Marcus Frean, and Robert Biddle

which links are randomly distributed among nodes. In a random graph geometry, nearly all the nodes have approximately the same number of links. Thus, every random graph has as its "typical scale" the average number of links per node. By contrast, the Web has no typical scale to its connectivity—a remarkable and somewhat counterintuitive property closely related to fractals.

Other scale-free graphs include the network formed by co-authors of articles in scientific journals, the physical connections forming the Internet, the network of airports connected by airline flights, networks of personal contacts, and even the patterns of connectivity between neurons in the human brain [1]. Well before being noticed by mathematicians in real-world graph structures, scale-free distributions were found in other contexts, including the relative frequencies of English words, the distribution of personal wealth, the size of cities, and the number of earthquakes of given strength [11].

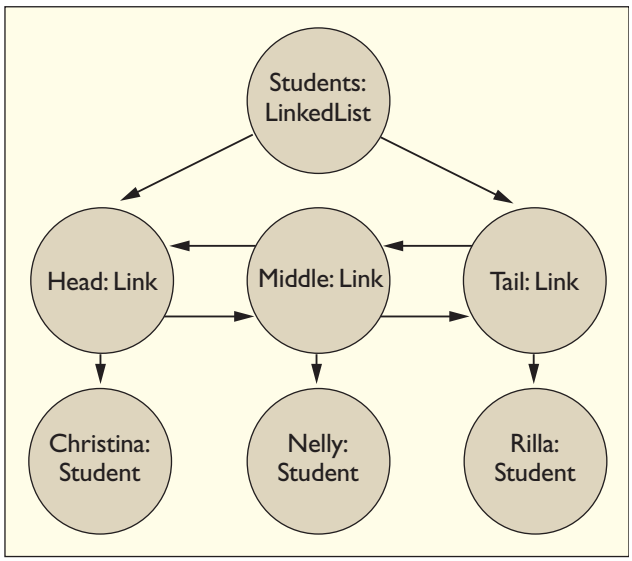

Figure 1. A simple object graph of a linked list. Each link object has two references to other link objects, except for the head and tail of the list. The student objects stored in the list are referred to by the link objects that store them.

Here, we examine the graphs formed by OO programs written in a variety of languages, showing they turn out to be scale-free networks as well. Apart from its considerable intrinsic interest, this unexpected facet of the geometry of real programs may help us optimize language runtime systems, improve the design of future OO languages, and reexamine modern approaches to software design.

Power Laws

The way to detect a scale-free phenomenon is to see if it shows up statistically in the form of a power law. In power law distributions, the number of occurrences N_k of some event of size k is proportional to k raised to some power. One drawback is that very rare events are by their nature noisy; there may be one node with, say, 1,000 connections and another with 1,005, but none with 1,002. For this reason, statisticians often adopt an alternative approach in which they first rank the event sizes by how often they occur, then look for a power law in the relationship between the number of occurrences N_k and the rank R_k of the form:

$$N_k \propto R_k^s$$

The easiest way to see a power law is to take logarithms of both sides or plot N vs. R on logarithmic scales; if the distribution follows a power law, we would expect to see a straight line with slope s.

Consider how often a particular word appears in any English novel. Common words like the, of, and or can be found many orders more times than most other words, while at the other extreme are a huge number of words that are used only rarely. In 1925, George Kingsley Zipf, a Harvard linguistics professor, conducted empirical studies [12] of word occurrences, observing that if we rank the words by the number of times they are found in the text of a particular novel, their rank will be proportional to their number of occurrences. Hence, if you draw a logarithmic plot of the number of times you find each word against the rank of such a word in your favorite novel, you will see a straight line.

Object Graphs

An object graph—the object instances created by a program and the links between them—is the skeleton of the execution of an OO program. Because each node in the graph represents an object, the graph grows and changes as the program runs. It contains just a few objects when the program is launched, gains more objects as they are created, and loses objects when they are no longer required. The structure of the graph (the links between objects) changes, too, as every assignment statement to an object's field may create, modify, or remove an edge in the graph.

Figure 1 outlines the object graph of a simple part of a program—in this case a doubly linked list of `Student` objects. The list itself is represented by a `LinkedList` object with two references to `Link` objects representing the head and tail of the list. Each `Link` object has two references to other `Link` objects—the previous and the next links in the list,

It appears that the world of object graphs is indeed scale free, just like the Web, the Internet, and many other networks we routinely use in our everyday lives.

plus a third reference to one of the `Student` objects in the list.

Object graphs are the most fundamental structure in the OO domain. The primary aim of OO analysis is to model the real world in terms of communicating objects (that is, in terms of an object graph), while OO design produces a description of an object graph that will eventually be embodied in a program. OO artifacts and methods (such as classes, associations, interfaces, inheritance, packages, patterns, UML, and CRC Cards) are ultimately techniques for defining object graphs by describing the contents of the objects and the structure of the links between them.

Program	Description	Objects ≥1 in-refs	Objects ≥1 out-refs
Java ArgoUML	A popular CASE tool	203,875	153,106
Java BlueJ	Visual OO programming and learning environment	171,666	123,701
Java Forte	A Java integrated development environment from Sun Microsystems	358,279	267,755
Java Jinsight	A memory analysis tool from IBM	76,312	118,272
Java Satin	A pen-based user interface research tool from Stanford University	80,415	53,328
C++ GCC	GCC is a C++ compiler used by developers for Unix platforms	71,990	15,064
Self	Self is a prototype-based OO language and environment	120,748	1,259,668
Smalltalk	Smalltalk is one of the original OO languages, self-contained in an environment developed using Smalltalk	375,529	188,031

The object graphs from Figure 2, listing the number of objects with at least one incoming or outgoing reference. They were obtained when experienced users were using the programs. Each C++ and Java program was run separately, and all objects were obtained in the interactive environments for Self and Smalltalk.

Given that object graphs are so basic to OO programs, it is surprising that so little attention is paid to their global structure. Some temporal properties of object graphs (such as the time performance of garbage collection algorithms and the distributions of object life spans) have been analyzed to support garbage collection [7]. Visualization of object graphs is used to support debugging [10]. Designers of programming languages work to control object graph structures using type systems [8], and compiler developers analyze parts of the graph to find ways to improve program performance [6].

Regarding scale-free structure in programs, the class diagrams of the Java Development Kit 1.2 (not the actual instantiation of objects of these classes at runtime) are scale-free [9]. We can also observe a similar structure in the distribution of pointers to atoms in Lisp [3].

Power Laws in Object Graphs

To analyze the geometry of object graphs in Java programs we used the Heap Profiler Library and the Java Virtual Machine Profiler Interface to collect a corpus of 60 object graphs from 35 programs encoded as binary snapshots of the Java heap. These heap snapshots record all the objects in the program, along with the geometry of the references between them, at one instant of the program's execution—exactly the kind of information shown in Figure 1. To analyze this corpus, we extended the Java Heap Analysis Tool [5], which parses these snapshots, to determine the properties of the program's object graphs.

For each graph, we first counted the number of objects with k references, for k of 1 and up. If the object graph had been scale-free we should have seen a straight line when the number of objects was plotted against k on log scales or when plotted against their rank ordering. Without exception, all the object graphs in our corpus demonstrated this phenomenon, leading to the conclusion that object graphs are scale free. The same general effect applies to both incoming references (reflecting an object's popularity) and outgoing references (reflecting an object's size); this effect can be detected in multiple snapshots taken during a single run of some programs. It appears that the world of object graphs is indeed scale free, just like the Web, the Internet, and many other networks we routinely use in our everyday lives.

Figure 2 includes five large Java snapshots and three additional object graphs from programs in other OO languages (see the table). We chose them for their size,

The power law indicates the reverse of the Lego hypothesis: There is no evidence of a typical size (the equivalent of a standard Lego brick) to objects.

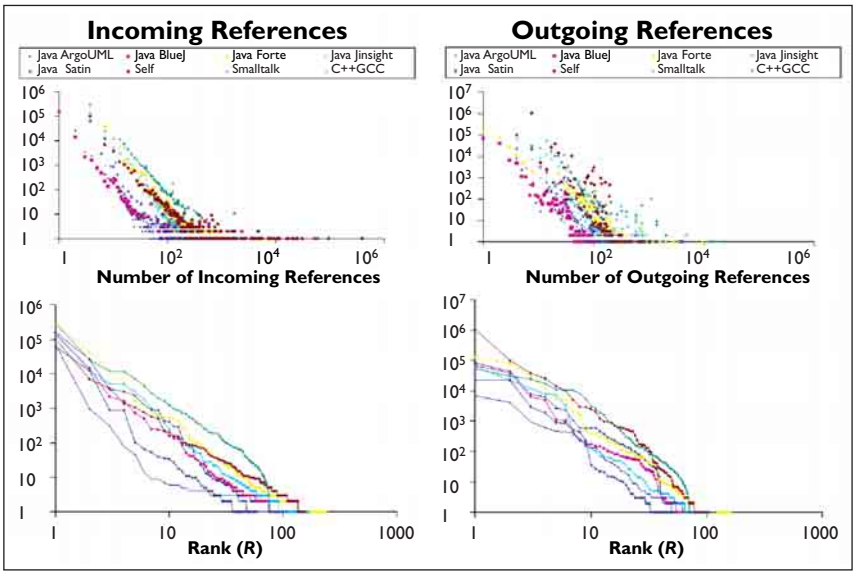

Figure 2. Power laws in object graphs. The upper two figures plot the number of objects with k references vs. the number of references k for incoming and outgoing references, respectively. The lower two figures plot the number of occurrences of each number of incoming (and outgoing) references vs. their rank, from highest to lowest number of references. All exhibit clear linearity on log-log scales, the characteristic feature of scale-free networks.

popularity, and diversity. The plots show all objects inside a program's memory at the moment the snapshot was taken. Although pointers from local variables and other references from the stack are included in the figure, excluding them improves the scale-free structure.

Perhaps the most intriguing aspect of the ranked graphs in the figure is that all the plots have similar slopes. This similarity is surprising because the plots all come from runtime snapshots of separate programs written for entirely different purposes in different languages. For incoming references the slope of the line is close to -2.5 while for outgoing references the slope is close to -3.

Figure 3 shows the number of objects with a given combination of incoming and outgoing references for the Forte data (the largest Java heap snapshot in our study). Notably, no objects have both high in-degree and high out-degree; on the contrary, the objects with many incoming references have few outgoing references, and vice versa. This effect may be a consequence of widely shared data structures with many outgoing references (such as arrays) having a proxy object that hides the actual reference to the array from the other objects that use it.

No Typical Size for Objects

If OO programs were constructed from completely independent components, like Lego bricks, then we would expect the distribution of the size and popularity of objects to stay the same, no matter how large the program—just as fixed-size Lego bricks can be assembled into structures of any size. The rhetoric of OO design is that large programs can be constructed the same way as small programs—by encapsulating complexity within objects at one level of abstraction and then composing these objects together at the next. Thus all objects should appear to be the same size and complexity; larger programs merely use more objects and more levels of abstraction. We have found the exact opposite in our corpus of OO snapshots. The power law indicates the reverse of the Lego hypothesis, that there is no evidence of a typical size (the equivalent of a standard Lego brick) to objects.

The relative steepness of the slope we obtained reflects the fact that there is an exceedingly large number of objects with few references among the programs. We might take this to imply that programmers prefer simple objects over complex objects, avoiding complexity just as software engineering guidelines would suggest. The power law distribution indicates that this adage is not followed; instead, large programs contain objects that are much more connected than one might expect. For example, for the unranked power law, the Java programs in our

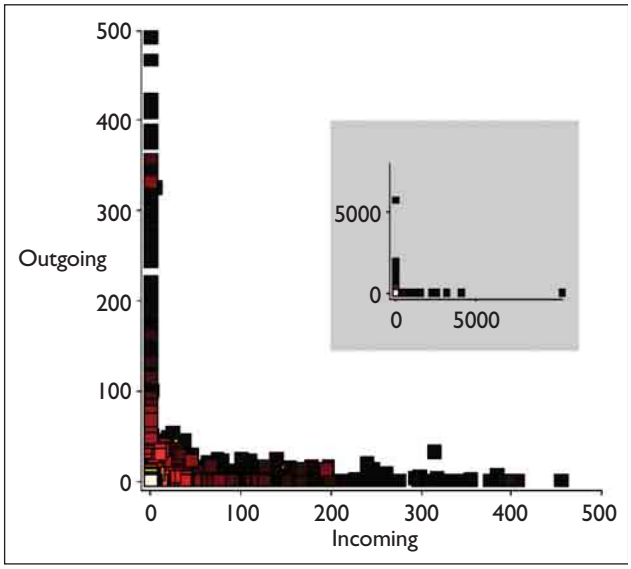

Figure 3. Distribution of incoming vs. outgoing references in the Forte snapshot. Lighter squares correspond to a greater number of objects having that combination of references. The objects contain up to 6,000 outgoing and 10,000 incoming references.

study all had a slope of approximately -2; it follows that for a given number of objects of size k there are about one quarter that number of size $2k$. Thus a program generating 10,000 objects of size one will also involve about 2,500 objects of size two, 625 of size four, 156 of size eight, and so on, leading to an expectation of one object of size roughly 100. In programs with twice as many objects altogether, we expect the number of very popular objects and the size of the largest object to increase by a factor of $\sqrt{2}$.

One especially useful aspect of scale-free networks is their robustness to damage. Because the vast majority of nodes are poorly connected to the rest of the graph, deleting them has a negligible effect on the connectivity of the remaining ones [2]. On the other hand, a small number of hub objects is highly connected, and deleting them is far more destructive. An implication of having a small number of hubs is that by concentrating our debugging methodologies on such popular well-connected objects, rather than on the unpopular ones, we may be able to improve the reliability of code—first by eliminating bugs from the hubs, then by dealing with other objects.

Aside from their scale-free character, power laws are also notable for having much longer tails than, say, exponential distributions. Larger programs thus contain considerably larger and more popular objects than simpler models would predict. Having many large and many popular objects may have consequences for both the design and implementation of OO programming languages. For example, it is well known that garbage collectors can improve their performance by assuming that most objects have only one or two outgoing references. The scale-free nature of object graphs explains why making this assumption is worthwhile.

Conclusion

We have found that distributions of incoming and outgoing references in object graphs follow a power law. This unexpected result raises theoretical questions about OO program design and has important implications for debugging costs, program understanding, and garbage collection. More generally, it challenges the perceived wisdom of OO design; unlike Lego bricks, objects within large programs have no characteristic scale. C

References

1. Barabasi, A.-L. *Linked: The New Science of Networks.* Perseus Press, New York, 2002.
2. Barabasi, A.-L., Albert, R., Jeong, H., and Bianconi, G. Powerlaw distribution of the World Wide Web. *Science 287* (Mar. 24, 2000), 2115.
3. Clark, D. and Green, C. An empirical study of list structures in Lisp. *Commun. ACM 20,* 2 (Feb. 1977), 78–87.
4. Erdos, P. and Renyi, A. On the strength of connectedness of random graphs. *Acta Mathematica Acadamiae Scientiarum Hungaricae 12* (1961), 261–267.
5. Foote, B. Heap Analysis Tool Project Web Page; https://hat.dev.java.net/.
6. Ghiya, R. and Hendren, L. Is it a tree, a DAG, or a cyclic graph? A shape analysis for heap-directed pointers in C. In *Proceedings of the 23rd ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages* (St. Petersburg Beach, FL, Jan. 21–24). ACM Press, New York, 1996, 1–15.
7. Jones, R. and Lins, R. *Garbage Collection.* John Wiley & Sons, Inc., New York, 1996.
8. Noble, J., Vitek, J., and Potter, J. Flexible alias protection. In *Proceedings of the 12th European Conference on Object-Oriented Programming* (Brussels, July). Springer-Verlag, Berlin, 1998, 158–211.
9. Valverde, S., Ferrer-Cancho, R., and Sole, R. Scale-free networks from optimal design. *Europhysics Letters 60*, 4 (Nov. 2002), 512–517.
10. Zimmermann, T. and Zeller, A. Visualizing memory graphs. In *Revised Lectures on Software Visualization, International Seminar,* S. Diehl, Ed. Springer-Verlag, Berlin, May 2001, 191–204.
11. Zipf, G. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology.* Hafner, New York, 1965 (facsimile of 1949 edition).
12. Zipf, G. *Psycho-Biology of Languages.* Houghton-Mifflin, Boston, 1935.

Alex Potanin (alex@mcs.vuw.ac.nz) is a Ph.D. student in the School of Mathematics, Statistics, and Computer Science at Victoria University of Wellington, New Zealand.

James Noble (kjx@mcs.vuw.ac.nz) is a professor in the School of Mathematics, Statistics, and Computer Science at Victoria University of Wellington, New Zealand.

Marcus Frean (marcus@mcs.vuw.ac.nz) is a senior lecturer in the School of Mathematics, Statistics, and Computer Science at Victoria University of Wellington, New Zealand.

Robert Biddle (robert_biddle@carleton.ca) is a professor in the Human-Oriented Technology Laboratory at Carleton University, Ottawa, Canada.