In-Depth Measurement and Analysis on Densification Power Law of Software Execution

Yu Qu
yqu@sei.xjtu.edu.cn

Qinghua Zheng
qhzheng@mail.xjtu.edu.cn

Ting Liu*
tliu@sei.xjtu.edu.cn

Jian Li
jianlee@stu.xjtu.edu.cn

Xiaohong Guan
xhguan@sei.xjtu.edu.cn

MOE Key Lab for Intelligent Networks and Network Security, Xi'an Jiaotong University
Xi'an, China

ABSTRACT

Measuring software execution is important for many software engineering tasks. In this paper, *Densification Power Law* (*DPL*) of software execution is measured and studied as a feature of growing software complexity. *Densification* means that during a networked system's evolution, it usually becomes denser and the number of edges and nodes grows with a consistent super linear relation. This feature was discovered and reported in 2005. In this paper, based on a measurement of 15 open-source Java programs, we show that when software systems are modeled as a series of dynamic *Call Graph*s during their executions, they always obey *DPL* with very close correlation. Then a comparison between static *Call Graph* and *DPL* is presented, showing that *DPL*'s properties cannot be derived statically. An explanation for *DPL* of software execution is given and verified. We believe the universality of *DPL* makes it an appropriate metric for software execution process.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Complexity measures, Process metrics*

General Terms

Theory, Measurement

Keywords

Software metrics, software execution, Densification Power Law

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

WETSoM'14, June 3, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2854-8/14/06...$15.00
http://dx.doi.org/10.1145/2593868.2593878

1. INTRODUCTION

Complex Network theory and graph algorithms have been successfully applied to software measurement and modeling [2, 3, 9]. But only a few of them [3] concentrate on dynamic execution process of software. Although many network growing models [1, 7] have been proposed in *Complex Network* theory, none of the existing research has investigated whether or not the dynamic processes of software also obey such models. Is there any common law among different software systems' execution processes? Can we discover new metrics for software execution from a growing network point of view? This paper aims to partially answer these questions and pave new research directions for software metrics.

It has been discovered in recent years that real world networks often exhibit a consistent tendency during their growing processes: they become denser over time. This phenomenon is not in accordance with traditional models, e.g., *preferential attachment model* [1, 7]. Moreover, the densifying processes exhibit a consistent relation, expressed as: $e(t) \propto n(t)^a$, where $e(t)$ and $n(t)$ are the number of edges and nodes of the network at time t, a is an exponent between 1 and 2. This feature is named as *Densification Power Law* [5, 6] (*DPL*).

In our previous research [8], we have discovered *DPL* feature in software's *Calling Network* (*CN*). *CN* is a software behavior model which is consisted of a series of *Calling Graph* (*CG*), a dynamic version of *Call Graph* [4].

As shown in Figure 1, during the execution process of a software, its *CG*s are constructed based on dynamic method/function call records. The sequence of these *CG*s (CG_1,CG_2,...,CG_n) presents typical *DPL*.

This paper makes the following contributions: 1) a much larger scale measurement is performed based on 15 widely-used open-source Java programs to study and show the universality of *DPL*; 2) A comparison between static *Call Graph* and the *DPL* equation is presented, showing that *DPL*'s properties cannot be derived statically; 3) An explanation for this finding is given. It is proved that the densification of software is mainly caused by the intensive reuse of software methods. We believe these results will provide new angles for software metrics research, and can be used in software dynamic execution and growth measurement and modeling.

2. CALLING NETWORK MODEL

The *Calling Network* (*CN*) model is formalized as:

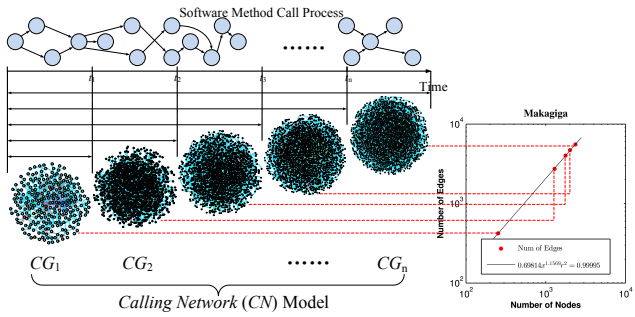


Figure 1: Densification Power Law feature.

$$\begin{cases} CN = \{CG_i \,|i \in \mathbb{N}\}, \\ CG_i = f_{CG-Gen}\,(CB_i)\,, CB_i \subseteq CB \text{ and} \\ CB_i = \{cb_k \,|(i-1)\cdot N_{Itv} \leq k \leq (i-1)\cdot N_{Itv} \;+ N_{CG}\}, \\ CG = (V, E)\,, w : E \to \mathbb{N}, \\ CB = \{cb_k \,|k \in \mathbb{N}\}, \\ cb_k = (t_k, Caller_k, Callee_k, Param_k)\,. \end{cases}$$

For more details of this model, please refer to our previous paper [8]. Briefly, setting $N_{Itv} = 0$ and $N_{CG} = i \cdot N_{Const}$, where N_{CG} is the step size to generate CG, N_{Const} is a constant non-zero value, will generate a CN model contains a series of CGs which represent the growing process of dynamic *Call Graph* over time.

3. MEASUREMENTS & DISCUSSIONS

3.1 Data Set and Experiment Implementation

A data set containing 15 real-world open-source Java programs are collected, as shown in Table 1. Most of the programs are widely used in practice and have millions of downloads on software repository sites. The 2nd column shows the program's architecture (**Arch**) (Web or desktop), the 3rd column provides the version number. The 4th to 5th columns give the value of **S**tatic **L**ines **o**f **C**ode, and the number of methods ("N/A"s in Table 1 and Table 2 represent that source codes of the corresponding programs are not publicly available). The 6th column gives the cardinality of CB, which is the quantity of method call records.

The Kieker framework[1], which is an open-source software dynamic monitoring framework based on AspectJ, is used as the instrumentation tool. Specific usage scenarios, user inputs and test cases were designed according to these programs' functionalities and documents. Then method call behaviors data of these programs are collected.

3.2 Densification Power Law

In our previous research [8], *DPL* feature was reported for the first time in software domain. In this section, a much larger scale measurement study is performed, followed by the comparison between static *Call Graph* and the dynamically derived *DPL* equation.

For the 15 subject programs, their CNs have been investigated. Figure 2 shows the results of CNs of 8 programs. In each subfigure, the number of edges versus the number of nodes are plotted in log-log scale. The straight line is the linear regression fit results, and the correlation coefficient of the regression is also shown. The first four subfigures show the results of 4 web programs, followed by the results of 4 desktop programs. It can be noticed that no matter what the architecture, size and functionality is, the *DPL* feature exists for all the subjects.

Table 2 gives the details of *DPL* results of the 15 programs. The 2nd column gives the regression equation. The 3rd and 4th columns give the corresponding N_{Const} (see Section 2) and the regression correlation coefficient r. It can be concluded that although the values of N_{Const} vary drastically, the real growing process of CN always fits to *DPL* with very high correlation coefficient (the smallest value is 0.98775 for LogicalDOC). The 5th column shows the number of nodes of the highest point in *DPL* figures ($N_{highest}$).

The 6th to 10th columns in Table 2 show the comparisons between the *DPL* equations and the static *Call Graph*s which are generated based on source codes of the corresponding programs. The 6th column emphasizes the total number of methods. The 7th and 8th columns show the number of nodes (N_{static}) and the number of edges (E_{static}) of the static *Call Graph*s. It should be noticed that for all the programs, N_{static} is much smaller than the total number of methods. This is mainly caused by the problem which has been discussed in [8]. The $E_{equation}$ in the 9th column is derived by substituting N_{static} into the equation in the 2nd column. The last column gives the difference between E_{static} and $E_{equation}$:

$$\Delta E = \frac{E_{equation} - E_{static}}{E_{static}}$$

The last column shows that the difference between static *Call Graph* and the equation of *DPL* is significant (ranging from -3.06% to 801.82%). Based on these results, it can be concluded that *DPL*'s properties (the exponent a and the adjustment parameter) can only be derived based on dynamic execution process and cannot be derived statically. The *DPL*'s properties reflect execution process of software from a certain viewpoint. What does the difference between dynamic and static (ΔE) imply? This question needs further investigation.

3.3 Microscopic Discussion

What makes the difference between *DPL* of software system and traditional *Complex Network* theory models? This section gives a microscopic discussion considering the characteristics of method calls in software.

In traditional *Complex Network* models [1, 7], when a new node arrives, it will connect to (or be connected by) old nodes following certain mechanisms. This process often leads to a constant average node degree over time. When we investigated the growing process of software's CN, the following differences have been observed:

Take the growing process of JForum's CN for instance, Figure 3 (a) shows the growth details from CG_1 to CG_2 in JForum's CN. The method `JForumBaseServlet.init` was in CG_1. When JForum's functionality executed, more methods were called, adding more nodes into CG_2. In CG_2, the method `JForumBaseServlet.init` called the method `getApplicationPath`. Then, to accomplish its functionality, `getApplicationPath` called the method `getValue`, which was an old node in CG_1. As the execution of `JForumBaseServlet.init` continued, it also called the `getValue` method.

[1]http://kieker-monitoring.net/

Table 1: Experiment subject programs

Programs	Arch	Version	SLoC	# Method	$\|CB\|$	Website
DrJava	Desktop	5.7.5	162,416	10,593	447,842	http://www.drjava.org/
Endeavour	Web	1.21	18,312	1,706	39,322	http://sourceforge.net/projects/endeavour-mgmt/
FreeMind	Desktop	0.9.0	53,669	5,974	192,694	http://sourceforge.net/projects/freemind/
JabRef	Desktop	2.9.2	144,406	6,449	185,134	http://jabref.sourceforge.net/
jEdit	Desktop	5.1.0	185,569	7844	438,321	http://www.jedit.org/
JForum	Web	2.1.9	65,040	2,991	42,516	http://jforum.net/
JPetStore	Web	6.0	1,893	289	2,099	http://code.google.com/p/mybatis/
Kunagi	Web	0.23	176,486	18,021	198,259	http://kunagi.org/
LogicalDOC	Web	6.7.0	131,888	8,692	160,685	http://www.logicaldoc.com/
Makagiga	Desktop	3.8.2	156,906	10,356	324,928	http://sourceforge.net/projects/makagiga/
OpenKM	Web	6.2.2	N/A	N/A	249,990	http://www.openkm.com/
OpenProj	Desktop	1.4	151,821	11,632	371,750	http://sourceforge.net/projects/openproj/
OpenSyncro	Web	2.2	54,163	3,276	137,433	http://www.opensyncro.org/
Sweet Home 3D	Desktop	4.2	109,090	6,346	381,586	http://www.sweethome3d.com/
Weka	Desktop	3.7.10	N/A	N/A	237,239	http://www.cs.waikato.ac.nz/ml/weka/

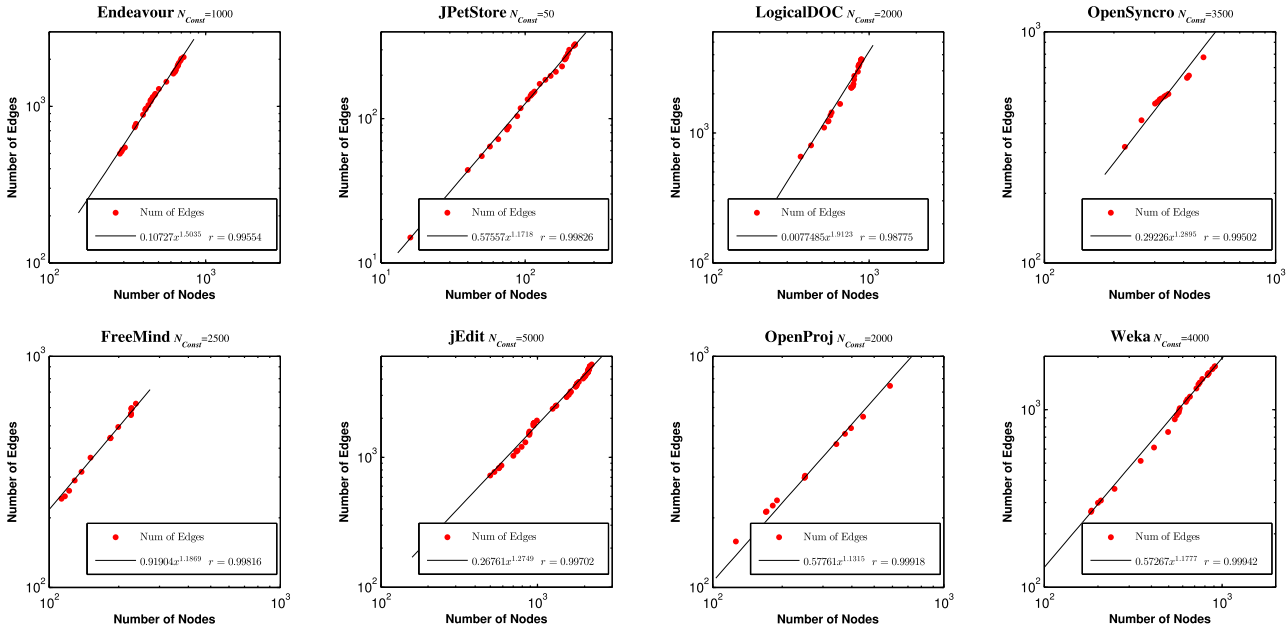
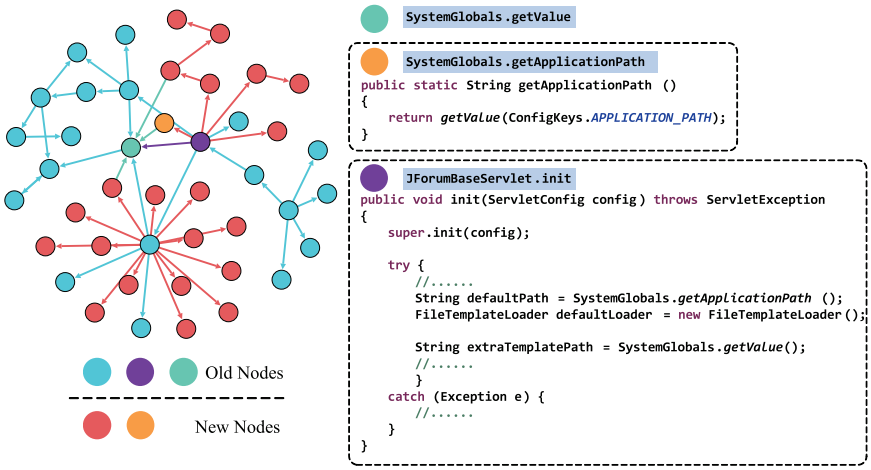


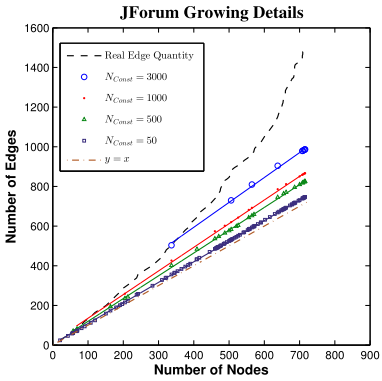
Figure 2: Densification Power Law phenomenon of 8 open-source programs.

Table 2: Densification Power Law results and comparison with *Static Call Graphs*

Programs	Equation	N_{Const}	r	$N_{highest}$	# Methods	N_{static}	E_{static}	$E_{equation}$	$\Delta E\%$
DrJava	$0.68431x^{1.1276}$	7,000	0.99956	1,701	10,593	6,204	13,346	12,938	-3.06%
Endeavour	$0.10727x^{1.5035}$	1,000	0.99544	723	1,706	1,468	3,331	6,189	85.8%
FreeMind	$0.91904x^{1.1869}$	2,500	0.99816	236	5,974	3,724	7,804	15,914	103.91%
JabRef	$0.67521x^{1.1477}$	4,000	0.99894	867	6,449	4,885	9,805	11,565	17.95%
jEdit	$0.26761x^{1.2749}$	5,000	0.99702	2,221	7,844	5,606	13,845	16,094	16.24%
JForum	$0.31046x^{1.2845}$	1,000	0.99341	715	2,991	2,051	5,749	5,579	-3.03%
JPetStore	$0.57557x^{1.1718}$	50	0.99826	221	289	97	124	122	-1.2%
Kunagi	$0.73925x^{1.1287}$	4,500	0.99858	780	18,021	12,583	24,699	31,349	26.92%
LogicalDOC	$0.0077485x^{1.9123}$	2,000	0.98775	891	8,692	5,932	14,112	127,270	801.82%
Makagiga	$0.47128x^{1.2075}$	1,500	0.99887	1,776	10,356	7,078	17,781	20,992	18.06%
OpenKM	$0.46685x^{1.1842}$	6,000	0.99963	1,389	N/A	N/A	N/A	N/A	N/A
OpenProj	$0.57761x^{1.1315}$	2,000	0.99918	2,823	11,632	7,258	13,752	13,494	-1.87%
OpenSyncro	$0.29226x^{1.2895}$	3,500	0.99502	657	3,276	1,865	2,937	4,823	64.21%
Sweet Home 3D	$0.7917x^{1.1404}$	5,500	0.9995	1,117	6,346	4,547	11,219	11,744	4.68%
Weka	$0.57267x^{1.1777}$	4,000	0.99942	910	N/A	N/A	N/A	N/A	N/A



(a) Growing details from CG_1 to CG_2.



(b) Edge removal results.

Figure 3: Growing details of JForum's *CN* and edge removal results.

The aforementioned process added 3 new edges to CG_2. Among them, one edge was between 2 old nodes in CG_1 (the **purple** edge), one edge (the **green** edge) was a "passive" method call to accomplish the functionality of a new node (`getApplicationPath`)

These 2 edges are different from the edge which links `JForumBaseServlet.init` and `getApplicationPath`, the latter is the same with all the **red** edges in Figure 3 (a). These 2 situations (new edge between 2 old nodes and "passive" edge) have not been considered in existing models. What if these edges are excluded in *CN*? Whether we could derive a similar result with traditional models?

Figure 3 (b) shows the results of JForum, all the points are the number of nodes versus the number of edges after removing the 2 aforementioned kinds of edges with different N_{Const}. These results are consistent with the predictions made by existing *Complex Network* models [1, 7] (constant average node degree over time). Similar results have been obtained on other subject programs.

Thus, the densification process is mainly caused by the pervasive dynamic reuse of software methods. But why the densification process follows a consistent exponent? This question should be answered in future research.

4. CONCLUSIONS

Based on 15 widely-used software systems, we have shown the universality of *Densification Power Law* (*DPL*) during software's execution. We believe this interesting finding will provide new angles for software runtime measurement. The difference between static *Call Graph* and *DPL* has been presented. An explanation for the densification process has been given. It is believed that the major cause of this finding is the reuse of software methods. After removing these reusing method calls, the growth of software's *CN* is in accordance with predictions made by traditional *Complex Network* models. These measurements and findings will pave new research directions for software metrics.

5. ACKNOWLEDGMENTS

This paper was supported by the National Natural Science Foundation of China (91118005, 91218301, 61221063, 61203174 and 61202392), Key Projects in the National Science & Technology Pillar Program (2011BAK08B02), Doctoral Fund of Ministry of Education of China (20110201120010) and the Fundamental Research Funds for the Central Universities.

6. REFERENCES

[1] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[2] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos. Graph-based analysis and prediction for software evolution. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 419–429. IEEE Press, 2012.

[3] K.-Y. Cai and B.-B. Yin. Software execution processes as an evolving complex network. *Information Sciences*, 179(12):1903–1928, 2009.

[4] S. L. Graham, P. B. Kessler, and M. K. Mckusick. Gprof: A call graph execution profiler. *ACM Sigplan Notices*, 17(6):120–126, 1982.

[5] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[6] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[7] M. E. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[8] Y. Qu, X. Guan, Q. Zheng, T. Liu, J. Zhou, and J. Li. Calling network: A new method for modeling software runtime behaviors. In *Proceedings of the Second International Workshop on Software Mining*, Palo Alto, CA, USA, 2013. (in press, available at: http://nskeylab.xjtu.edu.cn/people/yuqu/files/2014/01/1.pdf).

[9] H. Zhang. Discovering power laws in computer programs. *Information Processing & Management*, 45(4):477–483, 2009.