Universal distribution of component frequencies in biological and technological systems

Tin Yau Pang[a,b] and Sergei Maslov[a,1]

[a]Department of Biosciences, Brookhaven National Laboratory, Upton, NY 11973; and [b]Department of Physics and Astronomy, Stony Brook University, Stony Brook, NY 11794

Edited* by Ken A. Dill, Stony Brook University, Stony Brook, NY, and approved February 19, 2013 (received for review October 16, 2012)

Bacterial genomes and large-scale computer software projects both consist of a large number of components (genes or software packages) connected via a network of mutual dependencies. Components can be easily added or removed from individual systems, and their use frequencies vary over many orders of magnitude. We study this frequency distribution in genomes of ∼500 bacterial species and in over 2 million Linux computers and find that in both cases it is described by the same scale-free power-law distribution with an additional peak near the tail of the distribution corresponding to nearly universal components. We argue that the existence of a power law distribution of frequencies of components is a general property of any modular system with a multilayered dependency network. We demonstrate that the frequency of a component is positively correlated with its dependency degree given by the total number of upstream components whose operation directly or indirectly depends on the selected component. The observed frequency/dependency degree distributions are reproduced in a simple mathematically tractable model introduced and analyzed in this study.

gene frequency | metabolic network | software dependency

Individual components of complex interconnected systems are used with vastly different frequencies. Examples include the frequency with which individual genes and their orthologs are encoded in genomes of different species (1); the frequency of local installations of individual software packages in multicomponent software projects (2); broad power-law distributions of the frequency of citations, visitations, or other measures of popularity of individual publications, Web pages, YouTube videos, Facebook, and Twitter pages, etc. (3–5); and power-law distribution of word use frequencies in text (6).

The explanations of the observed broad distribution of use frequency (or popularity) of individual components generally fall into two broad categories. The first category invokes random multiplicative processes (7, 8) recently exemplified by the preferential attachment model of growing networks (9, 10). These models, recently invoked to explain frequency distribution of genes in pan-genomes of bacterial species (11), largely ignore functional differences between components so that the ultimate popularity of a component is determined mostly by its age as well as random events in early phases of growth of the system. The second category of models invokes heterogeneity of functional roles of individual components (12, 13). It is reasonable to assume that the frequency of a component is mainly determined by the breadth of its functional role in the system. This explanation is especially applicable to biological and technological systems subject to natural and artificial selection, respectively. Indeed, the frequency of genes whose "popularity" is not matched by their functional importance will be quickly corrected by the evolution. In agreement with this explanation, genes encoding certain core enzymes of central metabolism or ribosomal components are present in genomes of virtually all species (figure 4 in ref. 14). However, genes encoding peripheral enzymes tend to have much lower frequency of appearance in genomes (14). The same rule applies to multicomponent software projects such as Linux, where the most frequently installed components (e.g., Python and gzip) are also among the most functionally important and reusable software libraries. Most other packages either directly or indirectly depend on these low-level components for their operation. As a result, these packages end up being installed on the vast majority (if not all) of individual Linux computers. In what follows, we present empirical results supporting this second, functional explanation of the power-law distribution of frequency of components in complex biological and technological systems.

Results

Empirical Distribution of Component Frequencies. The eggNOG database (15) provided us with information about the presence or absence of genes from 45,000 orthologous gene families in genomes of more than 500 bacterial species. The Ubuntu popularity contest project quantified the frequencies of installation of about 200,000 Linux packages on more than 2 million individual computers (2) (see *Materials and Methods* for details). We found the distributions of components' frequencies f_i in both biological and technological systems to share multiple common features, including a power-law scaling regime $P(f) \sim f^{-\gamma}$ with $\gamma \sim 1.5$ (see Fig. 1*B* for genomes and Fig. 1*E* for Linux) terminating with a peak at the maximal frequency $f \sim 1$ (Fig. 1 *A* and *D*). This peak, formed by components present in the vast majority of systems, also manifests itself as a broad plateau at $f \sim 1$ in Zipf's rank-frequency plots (Fig. 1 *C* and *F*). A broad distribution of gene frequencies has been previously reported in biological literature (1, 16–19); however, this study reports and explains its scaling exponent.

U-shaped $P(f)$ distributions are sometimes plotted on semi-logarithmic scale (1) with piecewise linear fit used to define three types of components dubbed "core" ($f > 0.95$), "character" ($0.95 \geq f > 0.1$), and "accessory" ($f \leq 0.1$) genes (16). In Fig. 1*A* we validate these previous observations and demonstrate them for Linux systems (Fig. 1*D*). We also confirm the existence and explain the origins of a sharp cross-over separating the core components with $f \sim 1$ from the rest of the distribution. We mathematically predict the number of core components to be $\sim\sqrt{N}$, where N is the total number of components with nonzero frequencies that are functionally connected to the core. The empirical data are in approximate agreement with this prediction. The separation between character and accessory genes is less well defined. Indeed, when plotted in log-log coordinates, the power-law scaling observed for $f \ll 1$ directly crosses over into the core region at $f \sim 1$ without an obvious intermediate region corresponding to character genes. In what follows, we argue that the power law is expected on purely theoretical grounds. Thus, "fractal organization of the gene Universe" (20, pp. 71–75)

Author contributions: S.M. designed research; T.Y.P. and S.M. performed research; T.Y.P. and S.M. analyzed data; and T.Y.P. and S.M. wrote the paper.

The authors declare no conflict of interest.

*This Direct Submission article had a prearranged editor.

[1]To whom correspondence should be addressed. E-mail: maslov@bnl.gov.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1217795110/-/DCSupplemental.

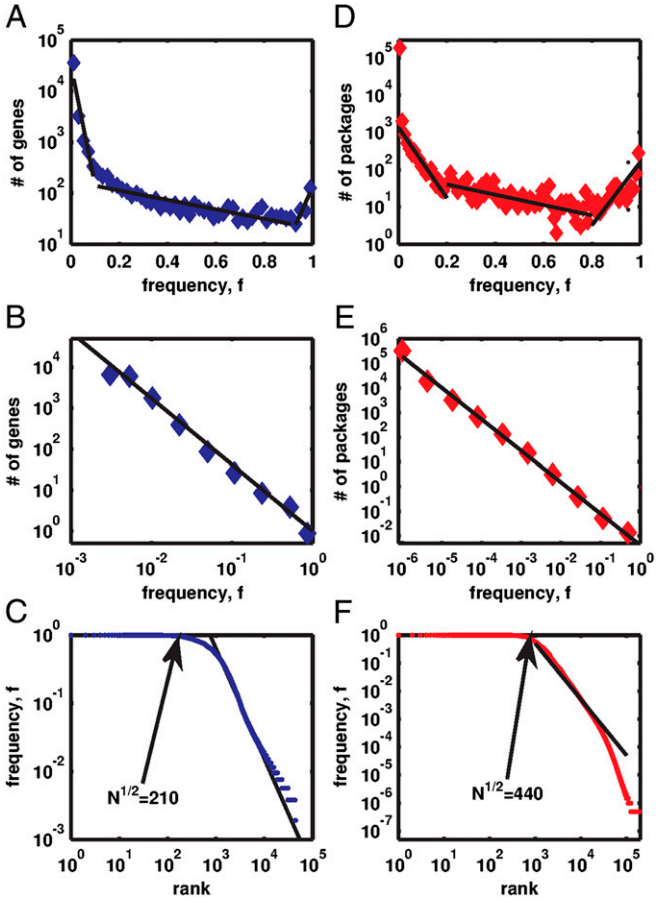

Fig. 1. The histogram $P(f)$ of the frequency f of bacterial genes present in genomes (*A*) or Linux software packages installed on computers (*D*) in semilogarithmic coordinates. Dashed lines show a piecewise linear fit used to define core ($f > 0.95$), character ($0.95 \geq f > 0.1$), and accessory ($f \leq 0.1$) components (1, 16). When plotted in log-log coordinates (*B* for genes and *E* for Linux), the histogram is consistent with the power law $P(f) \sim f^{-\gamma}$ with the exponents $\gamma_{\text{Genomes}} = 1.62$, and $\gamma_{\text{Linux}} = 1.42$ (solid lines in *B* and *E*). In rank-frequency Zipf's plots (*C* for genes and *F* for Linux), core components manifest themselves as plateaus at $f \sim 1$. Straight lines in *C* and *F* are the best power-law fits used to determine $\gamma_{\text{Genomes}}, \gamma_{\text{Linux}}$, and the arrows point to $\sqrt{N}$—the mathematically predicted number of core components.

manifests itself both in the scale-free distribution of component frequencies (as reported in this study) as well as in qualitatively similar shapes of $P(f)$ at different evolutionary timescales (as demonstrated in ref. 1).

Component's Frequency Is Positively Correlated with Its Dependency Degree. It is reasonable to expect the frequency of a component (a gene or a software package) to be influenced by its importance or the breadth of its functional role in the system. For a given component, we quantify the latter by the number of other components whose operation critically depends on it either directly (referred to as the direct dependency degree k_{dep}) or directly + indirectly (referred to as the total dependency degree K_{dep}). The difference between k_{dep} and K_{dep} can be easily understood in the dependency network of Linux packages (21). Edges of this directed network connect a given package to packages it requests to install during its own installation process. Some of these packages have direct dependencies of their own. For example, Fig. S1 visualizes direct and indirect dependencies of the Firefox browser. This cascade of sequential installations continues until all downstream packages required for the operation of the chosen package are installed. So, though $k_{dep}(i)$ counts the packages that require installation of the package i at the first step of this multistep process, $K_{dep}(i)$ counts the packages that do so at any step.

Though a similar interdependence of individual genes on each other certainly exists in biological systems, it is more difficult to quantify. Using the algorithm described in ref. 13, we calculated the dependency network for a subset of all gene families corresponding to metabolic enzymes (see *Materials and Methods* for details). Briefly, our algorithm derives upstream/downstream relations of enzymes reflecting their relative positions in metabolic pathways. The functioning of an anabolic enzyme requires the presence of enzymes in the smallest pathway necessary to synthesize all of its substrates from the minimal set of core metabolites (see *Materials and Methods* for our algorithm searching for such minimal pathway). The total dependency degree $K_{dep}(i)$ of the enzyme i is given by the total number of enzymes in this minimal pathway located downstream from it for anabolic enzymes (or upstream from it for catabolic enzymes). However, the direct dependency degree, $k_{dep}(i)$, counts enzymes located one step below (or above) it in this hierarchy. The direct dependency degree of an enzyme is closely related to its degree in the adjacency matrix of the metabolic network previously studied in refs. 10 and 22. Fig. S2 visualizes dependencies among enzymes in a particular metabolic pathway.

The scatter plot of the frequency of a component vs. its total (direct + indirect) dependency degree K_{dep} clearly shows positive correlation between the two variables. The Spearman rank correlation 0.3 (metabolic enzymes) and 0.47 (Linux packages) is highly statistically significant ($P < 10^{-16}$). A somewhat weaker correlation for metabolic enzymes can be attributed to an important difference between dependency networks in biological and computer systems. The dependencies of software packages in Linux are explicitly specified by their designers and thus totally unambiguous. The biological systems are designed in a more robust fashion and allow some flexibility in dependencies among their components. For example, in metabolic networks there is often more than one enzyme synthesizing a metabolite used by

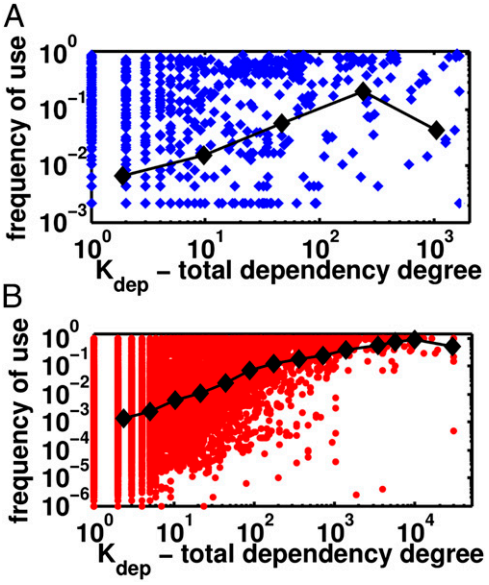

Fig. 2. Components' frequencies f (y axis) are positively correlated with their total (direct + indirect) dependency degrees K_{dep} (x axis) for both metabolic enzymes (*A*) (Spearman's $r_s = 0.30$) and Linux packages (*B*) (Spearman's $r_s = 0.47$). The black lines and symbols show the geometric averages of f in each logarithmic bin of K_{dep}.

another enzyme, which makes the definition of dependency degree of an enzyme more ambiguous and weakens its correlation with its frequency. To verify this hypothesis, we constructed a dependency network of metabolites instead of metabolic enzymes and recomputed their use frequencies in metabolic networks of different organisms (Fig. S3). The correlation coefficient (0.45) was considerably better than for metabolic enzymes (0.3) and just slightly lower than that observed for Linux packages (0.47) (Fig. 2).

Dependency Degrees Follow Power-Law Distributions. The distributions of direct (k_{dep}) and total (K_{dep}) dependency degrees for the metabolic enzymes as well as Linux packages are shown in Fig. 3; both have a power-law scaling region with exponents around −2 (k_{dep} shown in Fig. 3*A*) and −1.5 (K_{dep} shown in Fig. 3*B*), correspondingly. In addition to the power-law region, Zipf's rank degree plots of K_{dep} (Fig. 4 *A* and *B*), but not of k_{dep}, have plateaus formed by the core components with the largest K_{dep} (compare with frequency plateaus Fig. 1 *C* and *E*). Direct dependency degrees in a variety of large software projects have been previously reported to have scale-free distribution with exponents around −2 (see ref. 23 for Linux as well as in-degree exponents in table 1 of ref. 24).

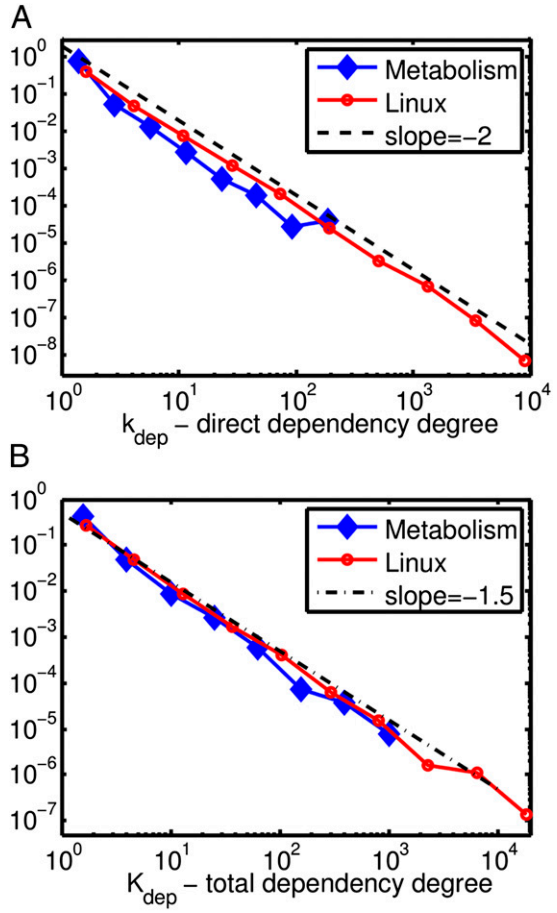

Fig. 3. Probability distributions of direct (k_{dep}; *A*) and total (K_{dep}; *B*) dependency degrees for metabolic enzymes (blue diamonds) and Linux packages (red circles). Power-law fits to direct degree cumulative distribution give −2.08 for metabolic enzymes and −1.91 for Linux packages, and are both consistent with the −2.0 scaling law (solid line in *A*). Power-law fits to direct degree cumulative distribution give −1.5 for metabolic enzymes and −1.56 for Linux packages, consistent with the mathematically derived −1.5 scaling (solid line in *B*).

Discussion

One of the intriguing results presented above is a remarkable similarity of distributions of frequencies (Fig. 1) as well as topological properties of dependency networks in biological (Fig. 3, red circles) and technological (Fig. 3, blue diamonds) systems. It is rather surprising to see near-perfect overlap of distributions in these two systems of very different origins: one is optimized by nature over billions of years of evolution, whereas the other is designed by a distributed population of human software engineers over the past several decades. In fact, we argue below that the functional form of $P(K_{dep})$ and $P(f)$ observed in this study is a universal property of any multicomponent and multilayered complex system. Such systems grow by gradually acquiring new components whose operation extends the functions performed by previously acquired components. Dependency networks connecting components to each other in such systems tend to be multilayered as a direct consequence of the long history of growth and evolution (25). Metabolic and software dependency networks used in this study with 34 and >40 layers, respectively, are indeed multilayered. A slightly different version of the universal metabolic network has been estimated (25) to have up to 60 layers of enzymes gradually acquired over billions of years of biological evolution (see figures 6 and 7 in ref. 25).

One mathematically tractable example of a multilayered dependency network is provided by a critical random branching tree (26)—namely, a tree with the branching ratio b close to 1. Here the branching ratio $b \leq \langle k_{dep} \rangle$ counts nodes that directly depend on a given node and are located one layer above it. Indeed, in a branching tree with b significantly larger or smaller than 1, either the number of layers is logarithmically small ($b \gg 1$) or the branches terminate prematurely ($b \ll 1$), rendering a multilayered network impossible.

For a critical branching tree, one can show that $P(K_{dep}) \sim K_{dep}^{-\gamma}$ with $\gamma = 1.5$. Indeed, the part of the tree located upstream of a given node itself constitutes an instance of a critical branching process that is independent from the rest of the tree; therefore, its size is distributed with the Galton–Watson exponent $\gamma = 1.5$ (see ref. 26 for the mathematical derivation). Because no subtree can be larger than the parent tree, in a tree of size N, one expects to find $N \cdot P(K_{dep} \geq N) = N \cdot N^{1-\gamma} = \sqrt{N}$ subtrees with sizes about N. Therefore, about $\sqrt{N}$ nodes located at the lowest layers of the dependency network will have the largest possible dependency degree $K_{dep} \sim N$ (see *Materials and Methods* for more details). These nodes constitute the plateau in Zipf's plots (Figs. 1 *C* and *E* and 4 *A* and *B*) and the large-X peak in the U-shaped distribution of dependency degrees or frequencies of system's components (Fig. 1 *A* and *D*).

Though the distribution of dependency degrees in a critical branching tree is in excellent agreement with the empirically observed data, there is a conceptual difference between real-life dependency networks and trees. In a tree, each component directly depends on one, and only one, downstream component. However, in real-life networks this number, D, is certainly larger than one; it varies from component to component, but averages ~2 for both metabolic networks and Linux packages. To describe real-life dependency networks with $D > 1$, we introduced and studied the following simple model. In our model, dependency networks start to grow from a few seed components. At each evolutionary time-step, one adds a new component depending on D_i randomly selected existing components. For simplicity we assume D_i to have a Poisson distribution with average $\langle D_i \rangle = D$. However, as shown in *SI Materials and Methods*, our results depend only on the average value of D_i. We mathematically derive (see *SI Materials and Methods* for step-by-step calculations) that the total dependency degree K_{dep} in a dependency network of size N generated by this model has a power-law tail $P(K_{dep}) \sim K_{dep}^{-(1+1/D)}$ as well as a plateau in the Zipf's plot composed of $N^{(D-1)/D}$

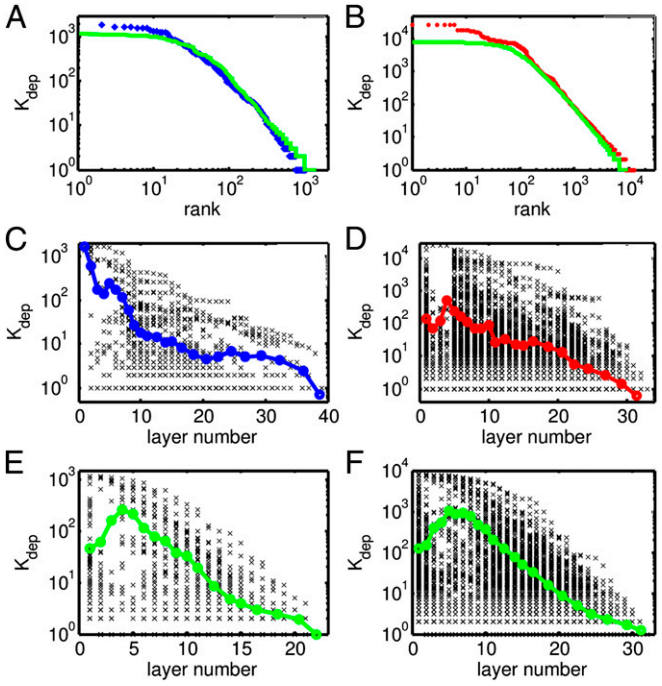

Fig. 4. Zipf's plots of total dependency degree in real metabolic (blue symbols in *A*) and Linux (red symbols in *B*) systems fitted with a random dependency model with $D = 2$ and $n = 1{,}500$ (green symbols in *A*) or $n = 10{,}000$ (green symbols in *B*), respectively. *C* and *E* show K_{dep} vs. the layer number in the metabolic network and the best-fitting random model, respectively. *D* and *F* do the same for Linux dependency network and its best approximation with random model. Black dots show scatter plots of individual nodes, and color lines are binned averages.

nearly universal components. Simulations of the model with $D = 2$ and $n = 1{,}500$ (green line in Fig. 4*A*) provides a reasonable fit to the metabolic dependency degree distribution (blue diamonds in Fig. 4*A*), whereas $D = 2$ and $n = 10{,}000$ (green line in Fig. 4*B*) is an excellent fit to the Linux dependency degree distribution (red circles in Fig. 4*B*).

We see that the model with $D \sim 2$ provides a rather good fit to dependency networks in both biological and technological systems. Metabolic enzymes usually have two substrates, and rarely one or three and more substrates. Hence in this case there is a good biophysical explanation for the observed value of $D_{met} = 1.7 \sim 2$. The situation is more complicated for the Linux dependency network where there are no geometrical limitations on the number of direct dependencies of a software package. This network is characterized by a large number of direct links between packages already indirectly connected on each other. Such shortcuts (known as feed-forward loops in the network jargon) do not change the overall (direct + indirect) network of package dependencies. Because our model does not contain feed-forward loops beyond those created by pure chance, we pruned them from the Linux direct dependency network as well. After removing all direct links short-circuiting any chain of direct links in the Linux dependency network, we were left with a direct dependency network with $D_{Linux} = 2.4 \sim 2$ and the same set of direct + indirect package dependency links as the original network. Admittedly in the case of Linux packages we have no ready explanation for this particular value of D beyond a vague notion that the easiest way to make a new package is to combine the outputs of two already existing ones.

The similarity between real-life dependency networks and those generated by our model with $D = 2$ extends beyond the shape of the total dependency degree distribution with the exponent $\gamma = 1 + 1/D = 1.5$ and $N_c = N^{(D-1)/D} = \sqrt{N}$ of the best-connected components forming the plateau in Zipf's plots. Our model makes very specific predictions about how the dependency degree of a component depends on the time when it was first added to the dependency network. Unfortunately, obtaining system-wide information about these "creation" times is not easy for Linux, and downright impossible for metabolic enzymes. As advocated in ref. 25, the time of appearance of a metabolic enzyme in the metabolic pan-network can be estimated from its layer number obtained by the scope expansion algorithm. Using the layer number of a node in a real-life dependency network as a proxy of its acquisition/creation time, we investigated its correlations with its total dependency degree. It stands to reason that older nodes located at bottom layers will tend to have a systematically larger dependency degree both in model and real networks; this is indeed what was observed and shown Fig. 4 *C–E*.

An important caveat in applying the $N_c = \sqrt{N}$ relationship is that N counts only those components that are directly or indirectly connected to the core by the functional dependency network; for biological systems, this allows us to reconcile the apparent paradox. Indeed, the pan-genome of all bacterial species is believed to be open (16). That is to say, N continues to increase without any hint at saturation as we sequence genomes of new bacterial species or even new strains of the same species (figure 1 in ref. 27). At the same time, the core bacterial genome remains relatively stable. Different methods result in somewhat different estimates of N_c, ranging from 250 in ref. 16 to 400 in refs. 17–19. To reconcile the apparent stability of N_c with unlimited growth of N, one recalls that continuing expansion of N is caused by either nonfunctional (prophages or transposable elements) or extremely niche-specific gene families—both are likely to be disconnected from the core and hence will not contribute to growth of N_c. Assuming $N_c \leq 500$, one gets the upper bound on the number of gene families connected to the core at ~250,000.

The frequency of a given component is expected to be strongly correlated with its total dependency degree. Indeed, the system using any of K_{dep} components located upstream of a given component is guaranteed to include this component itself. Hence, if every software package (metabolic enzyme) was equally likely to be initially selected (with probability $p_i = \frac{1}{N}$) for local installation on a computer (incorporation into a bacterial genome), one would have $f_i = \sum_{j=1}^{K_{dep}(i)} p_j = \sum_{j=1}^{K_{dep}(i)} \frac{1}{N} \sim K_{dep}(i)$. Deviations from this idealized linear relationship between f_i and $k_{dep}(i)$ in real data reflect among other things a nonuniform frequency of initial selection or installation of upstream components. Indeed, idiosyncratic differences in popularity p_i of higher-level components will be translated into differences in installation frequencies of lower-level components required for their operation. By adjusting the values of p_i—the initial popularity of components—we were able to increase the correlation coefficient between f_i and $\sum_{j=1}^{K_{dep}(i)} p_j \equiv \tilde{K}_{dep}(i)$ to 0.8 up from ~0.5.

Comparison between biological and technological networks has been previously performed by Yan et al. (28), and a number of similarities as well as significant differences were reported. However, the biological and technological systems studied were rather different from the ones we used in this study. The focus of the analysis performed in ref. 28 was on regulation and control represented by transcriptional regulatory network in *Escherichia coli* and the call graph between subroutines within the Linux kernel. However, in this article we compare biological and technological systems with independently installable components represented by metabolic enzymes encoded in bacterial genomes and software packages installed on top of the Linux kernel. A more systematic analysis of similarities and differences between

different versions of biological and technological complex systems will have to await future studies.

Materials and Methods

Our methods are briefly summarized here, and a more detailed description is provided in *SI Materials and Methods*.

Empirical Data for Frequencies of Use of Bacterial Genes. The eggNOG database v3.0 (15) contains the mapping of orthologous gene families to 630 species with fully sequenced genomes. We included in our analysis 529 bacterial genomes and their gene families assigned based on the clusters of orthologous genes and universal nonsupervised orthologous groups, which together cover 44,283 prokaryotic orthologous gene families. The resulting table showing the presence or absence of individual gene families in genomes was then processed to obtain the gene frequency f, defined as the fraction of 529 genomes the family is represented by at least one gene.

Empirical Data for Frequencies and Mutual Dependencies of Linux Packages. The package dependency network of Linux distribution Ubuntu 11.04 Natty was obtained by first getting a complete list of packages from http://packages.ubuntu.com/, and then running the command apt-rdepends to find all of the direct and indirect requirements for each package. The resulting network contains 33,473 packages, 157,667 direct, and 2,439,011 total (direct + indirect) dependency relations. The installation frequency data for 192,392 packages on 2,047,796 computers were downloaded from the package popularity contest project (http://popcon.ubuntu.com). In our analysis we used statistics for the whole archive sorted by the field institution.

Construction of the Dependency Matrices for the Metabolic Network. We used the union of all reactions in the Kyoto Encyclopedia of Genes and Genomes database (29) to construct upstream/downstream relations between enzymes using the following algorithm related to the scope expansion algorithm of ref. 25. For every enzyme, the minimal metabolic pathway connecting the product(s) of this enzyme to the set of five core metabolites was constructed as described in ref. 13. The direct dependency links were then drawn between the selected enzyme and enzymes in the top layer of this pathway, and direct + indirect links connect it to all enzymes in the minimal pathway. The resulting dependency network contains 1,832 reactions/enzymes connected to each other by 3,118 direct and 49,168 direct + indirect dependencies.

Power-Law Fits to the Data. Power-law fits to distributions were performed using MatLab package plfit.m developed by Aaron Clauset and collaborators, and downloaded from http://tuvalu.santafe.edu/~aaronc/powerlaws.

ACKNOWLEDGMENTS. We thank P. Dixit for useful discussions, critical reading, and editing of the manuscript; and K. Dill and J. Peterson for useful comments and suggestions. This work was supported by US Department of Energy Office of Biological Research Grant PM-031.

1. Koonin EV, Wolf YI (2008) Genomics of bacteria and archaea: The emerging dynamic view of the prokaryotic world. *Nucleic Acids Res* 36(21):6688–6719.
2. Pennarun A, Allombert B, Reinholdtsen P. Ubuntu Popularity Contest. Available at http://popcon.ubuntu.com/. Accessed September 5, 2011.
3. Sala A, Zheng H, Zhao BY, Gaito S, Rossi GP (2010) *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (Assoc for Computing Machinery, New York), 400–401.
4. Price DJ (1965) Networks of scientific papers. *Science* 149(3683):510–515.
5. Java A, Song X, Finin T, Tseng B (2007) in *Proceedings of the Ninth WebKDD and First SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis* (Assoc for Computing Machinery, New York), pp 56–65.
6. Zipf GK (1932) *Selected Studies of the Principle of Relative Frequency in Language* (Harvard Univ Press, Cambridge, MA), 1st Ed.
7. Yule U (1925) A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S. *Phil Trans R Soc B* 213:21–87.
8. Simon HA (1955) On a class of skew distribution functions. *Biometrika* 42(3-4): 425–440.
9. Albert R, Barabási A-L (2002) Statistical mechanics of complex networks. *Rev Mod Phys* 74(1):47–97.
10. Jeong H, Tombor B, Albert R, Oltvai ZN, Barabási A-L (2000) The large-scale organization of metabolic networks. *Nature* 407(6804):651–654.
11. Haegeman B, Weitz JS (2012) A neutral theory of genome evolution and the frequency distribution of genes. *BMC Genomics* 13:196.
12. Maslov S, Krishna S, Pang TY, Sneppen K (2009) Toolbox model of evolution of prokaryotic metabolic networks and their regulation. *Proc Natl Acad Sci USA* 106(24): 9743–9748.
13. Pang TY, Maslov S (2011) A toolbox model of evolution of metabolic pathways on networks of arbitrary topology. *PLOS Comput Biol* 7(5):e1001137.
14. Yamada T, Kanehisa M, Goto S (2006) Extraction of phylogenetic network modules from the metabolic network. *BMC Bioinformatics* 7:130.
15. Powell S, et al. (2012) eggNOG v3.0: Orthologous groups covering 1133 organisms at 41 different taxonomic ranges. *Nucleic Acids Res* 40(Database issue):D284–D289.
16. Lapierre P, Gogarten JP (2009) Estimating the size of the bacterial pan-genome. *Trends Genet* 25(3):107–110.
17. Danchin A, Fang G, Noria S (2007) The extant core bacterial proteome is an archive of the origin of life. *Proteomics* 7(6):875–889.
18. Fang G, Rocha EP, Danchin A (2008) Persistence drives gene clustering in bacterial genomes. *BMC Genomics* 9:4.
19. Danchin A (2009) Bacteria as computers making computers. *FEMS Microbiol Rev* 33(1): 3–26.
20. Koonin EV (2011) *The Logic of Chance: The Nature and Origin of Biological Evolution* (FT Press, Upper Saddle River, NJ).
21. LaBelle N, Wallingford E (2004) Inter-package dependency networks in open-source software. arXiv:cs/0411096.
22. Barabási A-L, Oltvai ZN (2004) Network biology: Understanding the cell's functional organization. *Nat Rev Genet* 5(2):101–113.
23. Maillart T, Sornette D, Spaeth S, von Krogh G (2008) Empirical tests of Zipf's law mechanism in open source Linux distribution. *Phys Rev Lett* 101(21):218701.
24. Louridas P, Spinellis D, Vlachos V (2008) Power laws in software. *ACM Trans Softw Eng Methodol* 18:2:1–2:26.
25. Handorf T, Ebenhöh O, Heinrich R (2005) Expanding metabolic networks: Scopes of compounds, robustness, and evolution. *J Mol Evol* 61(4):498–512.
26. Athreya KB, Ney PE (2004) *Branching Processes* (Dover, New York).
27. Touchon M, et al. (2009) Organised genome dynamics in the *Escherichia coli* species results in highly diverse adaptive paths. *PLoS Genet* 5(1):e1000344.
28. Yan K-K, Fang G, Bhardwaj N, Alexander RP, Gerstein M (2010) Comparing genomes to computer operating systems in terms of the topology and evolution of their regulatory control networks. *Proc Natl Acad Sci USA* 107(20):9186–9191.
29. Kanehisa M, Goto S (2000) KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res* 28(1):27–30.

Supporting Information

Pang and Maslov 10.1073/pnas.1217795110

SI Materials and Methods

Obtaining the Dependency Network and Occurrence Frequency Data for Linux Packages. The package dependency network of Linux distribution Ubuntu 11.04 Natty was obtained by first getting a complete list of packages from http://packages.ubuntu.com/, and then running the command apt-rdepends to find all of the direct and indirect requirements for each package. The resulting network contains 33,473 packages, and 57,667 direct and 2,439,011 total (direct + indirect) dependency relations.

The occurrence frequency data for 192,392 packages on 2,047,796 computers was downloaded from the package popularity contest (popcon) project (http://popcon.ubuntu.com/by_inst) (1). Participants of this project installed on their Linux computers tracking software that automatically reports the installation and subsequent use of different packages to the popcon server. We used the first column reporting the number of computers where this package was installed. A total of 189,711 packages were installed on at least one computer. Other columns not used in this study report the number of computers where this package was or was not used in the past month and the number of computers where it was recently updated.

The popcon project obtained the package data from Ubuntu Linux of a wide range of versions and CPU architectures, whose package repertoire and dependencies are a little bit different from each other. In the analysis we assumed that all participants are using Ubuntu 11.04 with x86 architecture, and based on this version of the Ubuntu Linux we calculated the direct and total dependency degree (k_{dep} and K_{dep}) of every package, and plotted the f vs. K_{dep} in Fig. 2. The packages not included in the official repositories of Ubuntu 11.04 or having zero installation frequency were ignored. Packages that are not required by any other packages (i.e., those with $K_{dep} = 0$) were also ignored.

Construction of Dependency Matrices for the Metabolic Network. The Kyoto Encyclopedia of Genes and Genomes Database (KEGG) (2) contains the data of metabolic reactions present in different organisms, and the universal metabolic network used in this study is the union of all of the reactions in KEGG consisting of 5,759 reactions and 4,785 metabolites. The group of five common metabolites present in the majority of organisms was selected as the core: H_2O, ATP, NAD+, oxygen, and CoA. The final version of the dependency network used in our study contains 1,832 reactions (or associated enzymes) connected to each other by 3,118 direct and 49,168 direct + indirect dependencies.

The goal of the metabolic network is to either convert nutrients taken up from the environment into core metabolites (catabolism), or to convert core metabolites into the constituents of the biomass and other essential ingredients (anabolism). The direction of the dependency network connecting metabolic reactions would be opposite in these two cases. For simplicity we will concentrate on the case of anabolic pathways below. For catabolic pathways we simply inverted the direction of reactions and then applied the procedure used for anabolic pathways.

To determine the set of other enzymes an enzyme i in an anabolic pathway depends on for its operation, we performed the following computational analysis. We selected all metabolic substrates of the enzyme i one by one, and for each of them we constructed the minimal pathway necessary to synthesize this metabolite from our predetermined set of 40 core metabolites. The union of all enzymes in these pathways constructed for each of the substrates of the enzyme i is a good approximation to the minimal set of enzymes necessary to enable the reaction catalyzed by the enzyme i; as such, we can plausibly assume that the enzymes in this union form the total downstream dependency set for the enzyme i.

Furthermore, by analogy to software dependency networks, the direct dependency neighbors of the enzyme i are made by the set of enzymes added at the last layer of our breadth-first search algorithm. Based on this definition, the direct dependency degree of an anabolic enzyme is closely related to the number of metabolic reactions using at least one of its products, which is one of the standard topological definitions of degree in metabolic networks (3). Therefore, the power-law distribution with the exponent −2 we measured for direct dependency degrees is closely related to previously reported scale-free topology on metabolic networks (3).

The rules by which this minimal pathway was constructed were previously described in ref. 4. For the sake of completeness, we included them in the text below.

By repeating the above procedure for all anabolic (catabolic) enzymes located downstream (upstream) from our core metabolites and thus reachable from the core by the scope expansion algorithm (5), we constructed our best approximation to the total dependency network of metabolic enzymes in the KEGG database.

Rules of Addition of Anabolic Pathways in Dependency Network Calculation.

i) At the beginning of the simulation, the model organism starts with a "seed" metabolic network consisting of 5 metabolites including H_2O, ATP, NAD+, oxygen, and CoA. It is assumed that our organism is able to generate all of these metabolites by some unspecified catabolic pathways.

ii) At each step, a new metabolite that cannot yet be synthesized by the organism is randomly selected from the scope (5) of our seed metabolites. This scope consists of all metabolites that in principle could be synthesized from the seed metabolites using all reactions listed in the KEGG database (5).

iii) To search for the minimal pathway that converts core metabolites to this target we first perform the scope expansion (5) of the core until it first reaches the target. In the course of this expansion, reactions and metabolites are added step by step (or layer by layer). Each layer consists of all KEGG reactions that have all their substrates among the metabolites in the current metabolic core of the organism (light blue area in figure 4 of ref. 5) and those generated by reactions in all of the previous layers (see figure 4 of ref. 5 for an illustration).

Mathematical Derivation of the Total Dependency Degree Distribution in the Random Model with $D = 2$. To mathematically derive the distribution of dependency degree K_{dep} in the simple model proposed in this study, we study its dependence on the time, t, a package was added to the growing dependency network. Here, time t is defined as the size of the network when a package was added and may have a nonlinear but monotonic relation to the actual time of addition (e.g., in exponentially expanding systems). $K_{dep}(t)$ can be calculated self-consistently from the following equation:

$$K_{dep}(t) = 1 + \int_{t+1}^{N} K_{dep}(t')D/t'. \qquad [S1]$$

Indeed, the total dependency degree of a package added at time t is given by the sum of total dependency degrees of packages added at later times, t', that directly depend on it. K_{dep} counts both direct and indirect dependencies, and thus indirect

dependencies of upstream packages are transferred to their downstream neighbors. In a random model, the likelihood of a package added at time t' to send a direct dependency link to a package added at time t is simply D/t'. It is easy to check that

$$K_{dep}(t) = (t/N)^{-D} \quad \textbf{[S2]}$$

is a solution of this equation. Indeed,

$1 + \int_{t+1}^{N} D(t'/N)^{-D} dt'/t' \simeq 1 + (t/N)^{-D} - (N/N)^{-D} = (t/N)^{-D} = K_{dep}(t)$. Eq. **S1** simply adds up the dependency degrees of multiple upstream neighbors of a node and thus ignores the inevitable overlap between these sets of nodes; this is a good approximation as long as the resulting $K_{dep}(t) \ll N$, and thus the overlap is small. It is clear, however, that if $D > 1$, Eq. **S2** cannot hold forever because it predicts $K_{dep}(1) = (1/N)^{-D} = N^D \gg N$. The total dependency degree cannot be larger than N, and this value is approximately reached at $t = N_c$ determined by

$$(N_c/N)^{-D} = N \quad \text{or} \quad N_c = N^{(D-1)/D}. \quad \textbf{[S3]}$$

N_c is the number of nearly universal "core" components in the system with total dependency degree $K_{dep} \simeq N$.

Eq. **S2** fully determines the power-law tail of the distribution of dependency degrees. Indeed, $P(K_{dep} \geq K) = P((t/N)^{-D} \geq K) = P((t \leq NK^{-1/D}) = NK^{-1/D}/N = K^{-1/D}$. Hence, $P(K_{dep} = K) = -dP(K_{dep} \geq K)/dK$ is given by

$$P(K_{dep}) \sim K_{dep}^{-(1+1/D)}. \quad \textbf{[S4]}$$

For $D = 2$, which is close to its empirical value in real-life biological and technological systems used in this study, one recovers familiar scaling laws:

$$P(K_{dep}) \sim K_{dep}^{-1.5} \quad \textbf{[S5]}$$

and

$$N_c = \sqrt{N}. \quad \textbf{[S6]}$$

Mathematical Derivation of the Total Dependency Degree Distribution in a Tree Generated by a Galton–Watson Branching Process. A Galton–Watson branching process is a Markov process in which every node in generation l produces some random number of "child nodes" in generation $l + 1$, according to a fixed probability distribution that does not vary from node to node. We denote as p_0 the probability for the process to terminate at each node, and p_d is the probability for a node to have a branch with d child nodes. The first node of the tree generated by a Galton–Watson branching process is denoted as the root. In biological and technological systems considered in this study, the root node represents the set of core metabolites, or the basic Linux packages that serve many high-level user applications. The scaling properties of the Galton–Watson process are fully determined by a single parameter, $\overline{d} = \sum_{d=0}^{d_{\max}} d \times p_d$, which is the average number of child nodes of any given node has. For $\overline{d} < 1$, referred to as an undercritical branching process, the cascade will terminate very quickly and is irrelevant to this study. Conversely, for $\overline{d} > 1$, referred to as a supercritical branching process, the cascades will likely never terminate. Moreover, for a given number of nodes N in a tree generated by an overcritical branching process, the total number of layers L is logarithmically small: $L \sim \log N / \log \overline{d}$. Real-life complex multicomponent systems such as metabolic networks and large software projects are characterized by a large number of hierarchical levels (4) incompatible by that in an overcritical branching process. Thus, overcritical branching processes will be also ignored in this study. In what follows, we limit our calculations to the third case in which $\overline{d} = 1$ is denoted as a critical branching process; this was previously demonstrated to be a good approximation to universal metabolic network (4), and the present study presents convincing evidence that it describes large software projects as well.

The direct dependency degree k_{dep} of a node in the Galton–Watson process is given by its number of child nodes plus 1 (to account for the dependency of a node on itself). The distribution of k_{dep} is then determined by p_d as $P(k_{dep} = 1) = p_0$, $P(k_{dep} = 2) = p_1$, $\ldots$ $P(k_{dep} = d) = p_{d-1}$; it can have any functional form as long as its average is equal to $1 + \overline{d}$, that for a critical branching process is equal to 2. It is important to emphasize that the Galton–Watson branching process does not provide an explanation for the power-law form of the distribution of direct dependency degrees (Fig. 3). However, the total dependency degree $K_{dep}(i)$ corresponds to the size of the entire subtree initiated at the node i. The Galton–Watson branching process is a Markov process and thus each node can be thought as starting its own instance of a branching process that is independent of the branching ratios of its predecessors. Hence, one would naively expect that the total dependency degree of N nodes in a tree generated by the critical branching process will have the same power-law distribution $P(K_{dep}) \sim K_{dep}^{-1.5}$ as N independently started branching processes. However, a quick calculation convinces one otherwise. Indeed, the largest dependency degree $D_{\max}$ in this case will be determined by the equation $1/N = P(K_{dep} > D_{\max}) = \sum_{D_{\max}} K_{dep}^{-1.5} \sim D_{\max}^{-0.5}$, or $D_{\max} = N^2$. Thus, the dependency degree cannot be larger than N—the total number of nodes in the tree. The size of the universal network with N nodes imposes a strict cutoff of N on sizes of its subtrees. Thus, the following process reproduces the distribution of sizes of subtrees of a critical branching tree with N nodes. In this process one simulates the critical branching process N times and stops it when and if its size s reaches N nodes if it does not terminate on its own before that. Therefore, among N nodes of the critical branching tree, one expects to find $N \times P(s \geq N) = N \times N^{-0.5} = \sqrt{N}$ nodes with the largest total dependency degree $K_{dep} = N$. The rest of the nodes follow the power-law distribution $P(K_{dep}) \sim K_{dep}^{-1.5}$, and this is indeed what we see in our numerical simulations on the universal network of 5,000 nodes generated by the critical branching process with $p_0 = p_2 = 1/2$ (data not shown).

1. Pennarun A, Allombert B, Reinholdtsen P. Ubuntu Popularity Contest. Available at http://popcon.ubuntu.com. Accessed September 5, 2011.
2. Kanehisa M, Goto S (2000) KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res* 28(1):27–30.
3. Jeong H, Tombor B, Albert R, Oltvai ZN, Barabási A-L (2000) The large-scale organization of metabolic networks. *Nature* 407(6804):651–654.
4. Handorf T, Ebenhöh O, Heinrich R (2005) Expanding metabolic networks: Scopes of compounds, robustness, and evolution. *J Mol Evol* 61(4):498–512.
5. Pang TY, Maslov S (2011) A toolbox model of evolution of metabolic pathways on networks of arbitrary topology. *PLOS Comput Biol* 7(5):e1001137.

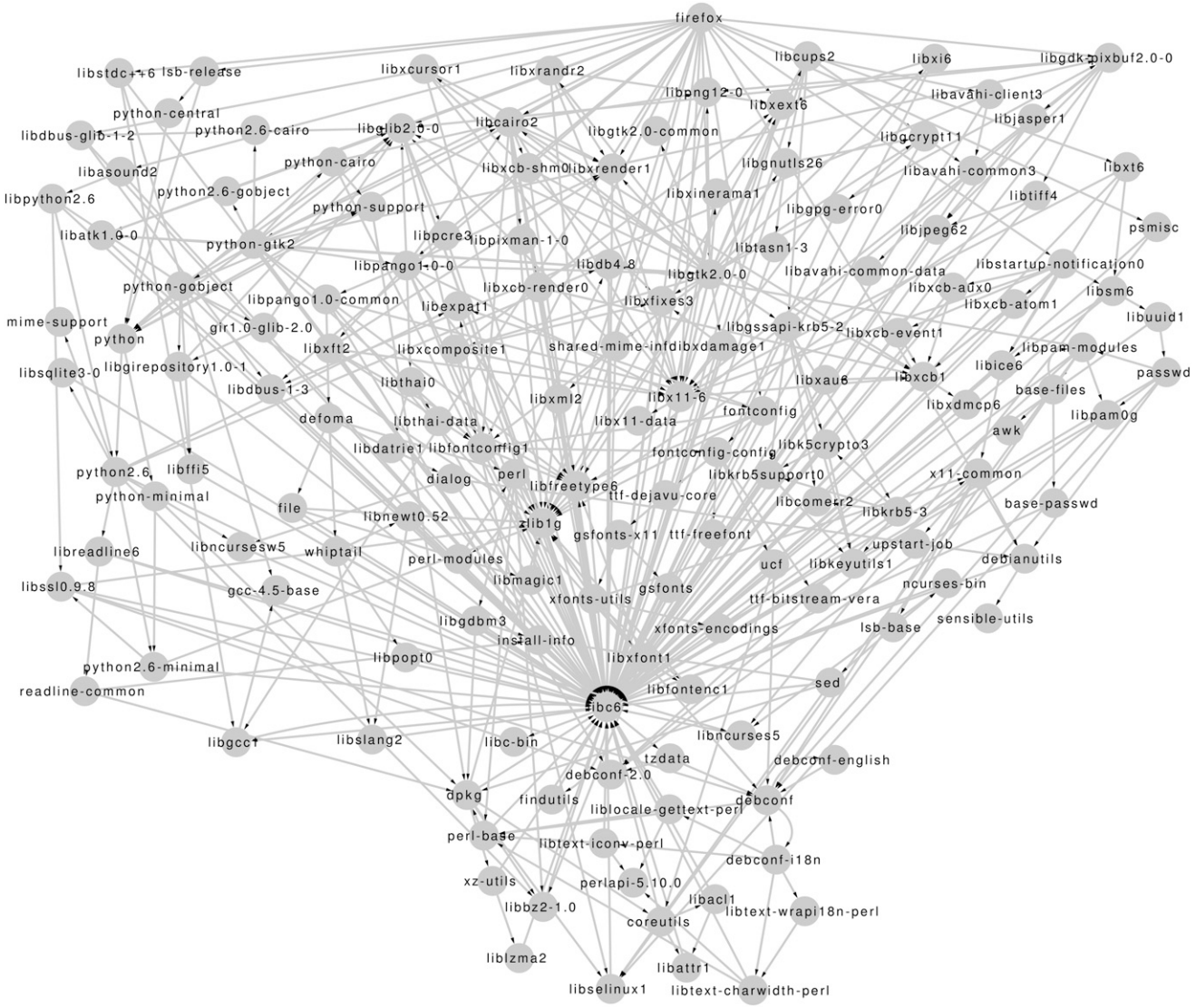

Fig. S1. Pathway diagram created with Cytoscape (1) showing 153 Linux packages directly or indirectly required for installation of the package Firefox (the top node).

1. Shannon P, et al. (2003) Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Res* 13(11):2498–2504.

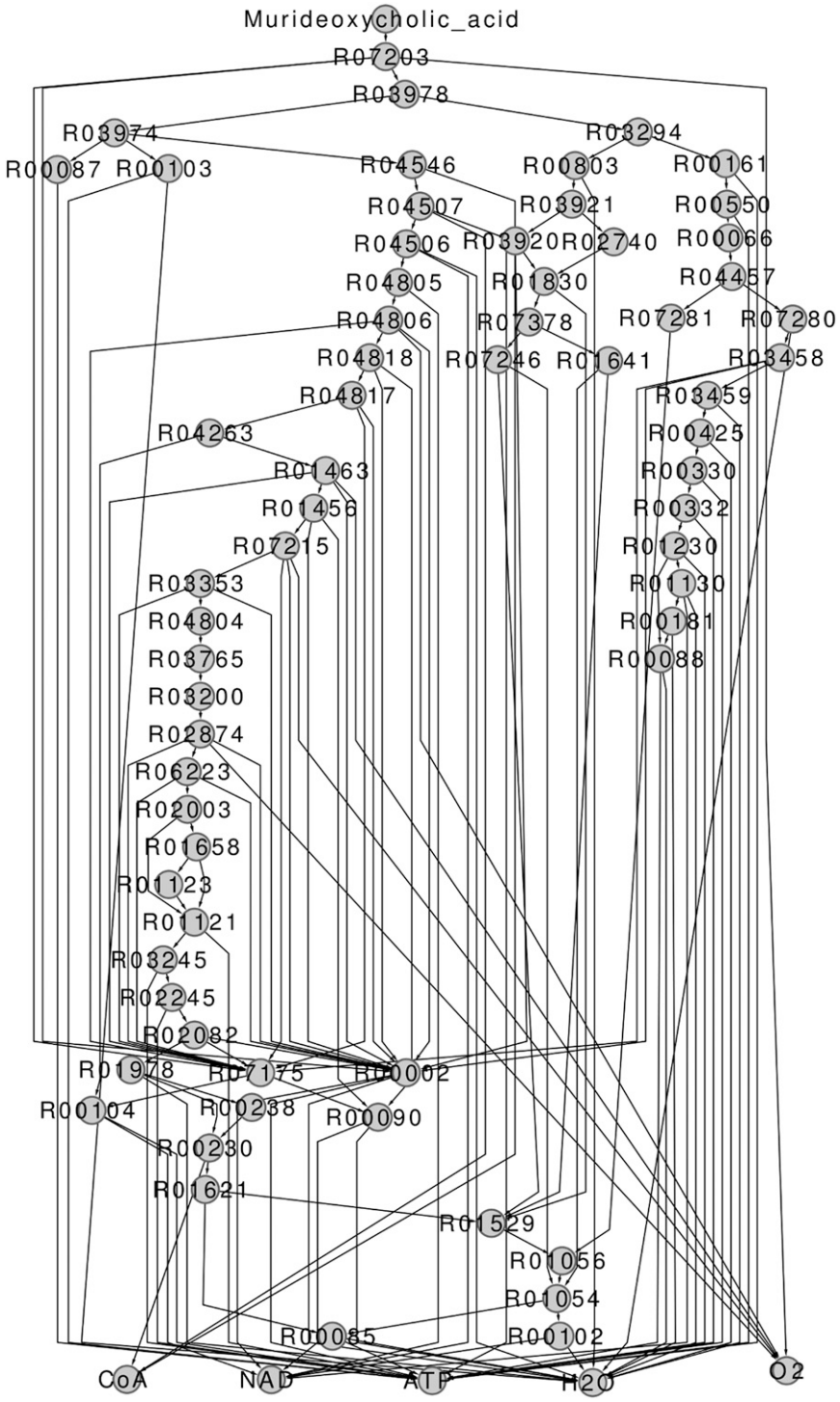

Fig. S2. Pathway diagram created with Cytoscape (1) showing 65 reactions (or equivalently, enzymes catalyzing these reactions) that the production of the metabolite murideoxycholic acid (KEGG database compound C15515; the top node) directly or indirectly depends on. Reaction numbers are given in KEGG notation.

1. Shannon P, et al. (2003) Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Res* 13(11):2498–2504.

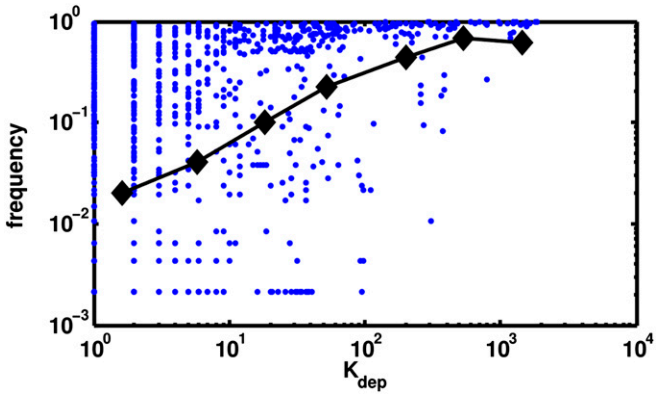

Fig. S3. The frequency of occurrence f (y axis) vs. the total (direct + indirect) dependency degree of metabolites K_{dep}. The two quantities are positively correlated (Spearman's $r_s = 0.45$). The black curve and symbols shows the average f in the logarithmic bins of K_{dep}.