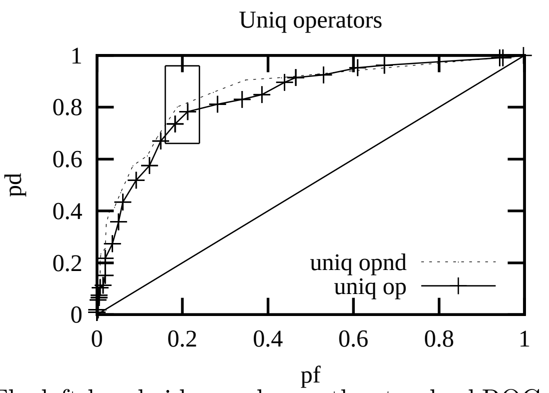
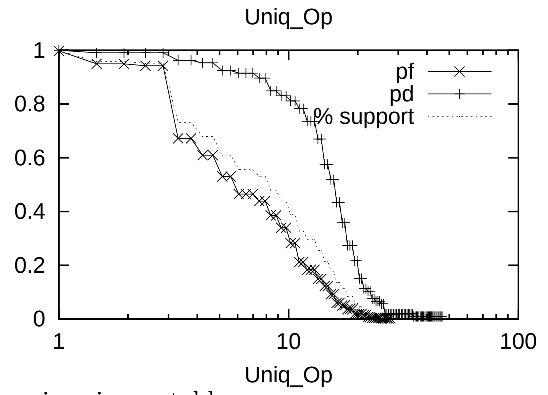
Here are some ROC curves for the metrics shown in Figure 1 and the mystery project "KC-1"[1]

Metric Type	Metric	Definiton
McCabe	v(G)	Cyclomatic Complexity
	ev(G)	Essential Complexity
	iv(G)	Design Complexity
	LOC	Lines of Code
Halstead	N	Length
	V	Volume
	L	Level
	D	Difficulty
	I	Intelligent Content
	E	Effort
	B	Error Estimate
	T	Programming Time
Line Count	LOCode	Lines of Code
	LOComment	Lines of Comment
	LOBlank	Lines of Blank
	LOCodeAndComment	Lines of Code and Comment
Operator/Operand	UniqOp	Unique Operators
	UniqOpnd	Unique Operands
	TotalOp	Total Operators
	TotalOpnd	Total Operands
Branch	BranchCount	Total Branch Count

Fig. 1 Metric Groups.

All the following graphs are left-hand-side, right-hand-side pairs that relate to the same parameter; e.g.





The left-hand-side graphs are the standard ROC curves; i.e. given a table of the following form:

	true=0	true=1
detected=0	A	B
detected=1	C	D

[1] And if you want to know what that project is, ask Ken Costello.

then

$$pf = \frac{C}{A+C}$$

is the probability of a false alarm and

$$pd = \frac{D}{B+D}$$

is the probability of a true detection.

A good ROC curve approaches $pd = 1$ (i.e. we really see something) and $pf = 0$ (i.e. we don't make mistakes). The "best" ROC curve I found for KC-2 was *Unique operands* and that is the dotted line on each left-hand-side graph. FYI, the best we seem to get out of *Unique operands* reads to me as

$$best = < pd, pf > = < 0.8, 0.2 >$$

BUT, then I got worried about the *support* for this conclusion. A "good" conclusion is supported by lots of data. Such "good" conclusions are more likely to be applicable in future applications.

So let's define:

$$support = \frac{C+D}{521}$$

and is the number of times the 521 items in the data set made our detector go "ping". The right-hand-side graphs shows how $support, pf, pd$ change as some attribute changes.

So now I can reveal the secret ordering rule. The graphs are ordered by how much support each parameter gets within 20% of *best* (the region shown as a rectangle in the left-hand graphs).

Of course, this analysis ignores two more possible criteria:

Timeliness: How early in a project lifecycle can a good detector be seen?
External validity: How well do these detectors on unseen data?

(And thats a problem for another day...)

1 Metrics

Oh, and I better explain the metrics.

1.1 McCabe

The McCabe metrics are a collection of four software metrics: essential complexity, cyclomatic complexity, design complexity and LOC [**?**,**?**,**?**]. Of these four, all but LOC are metrics which were developed by T. J. McCabe. McCabe & Associates claim that these complexity measurements provide insight into the reliability and maintainability of a module. For example, around NASA IV&V, a cyclomatic complexity of over 10 or an essential

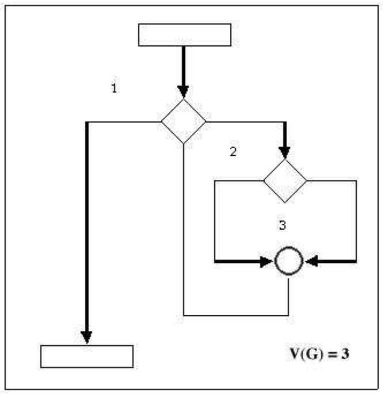


Fig. 2 Example program flowgraph

complexity of over 4 is flagged as a module that will be difficult to maintain and/or debug. This paper will not attempt to make any refutation to those claims and practices; however, these metrics are also commonly used as predictors for error-prone modules. As this paper will demonstrate, these complexity measurements do not *always* point the way towards modules with increased error density.

The following paragraphs present a short overview of the three complexity metrics mentioned previously.

Cyclomatic Complexity, or $v(G)$, measures the number of *linearly independent paths*[2] through a program's flowgraph[3]. $v(G)$ is calculated by:

$$v(G) = e - n + 2$$

where G is a program's flowgraph, e is the number of arcs in the flowgraph, and n is the number of nodes in the flowgraph [1]. For example, Figure 2 is a simple flowgraph; it's cyclomatic complexity is 3, since the graph has 6 arcs and 5 nodes ($v(G) = 6 - 5 + 2 = 3$).

Essential Complexity, or $ev(G)$, is the extent to which a flowgraph can be "reduced" by decomposing all the subflowgraphs of G that are D-structured primes [4]. $ev(G)$ is calculated by:

$$ev(G) = v(G) - m$$

where m is the number of subflowgraphs of G that are D-structured primes. [1]

[2] A set of paths is linearly independent if no path in the set is a linear combination of any other paths in the set

[3] A flowgraph is a directed graph where each node corresponds to a program statement, and each arc indicates the flow of control from one statement to another

[4] D-structured primes are also sometimes referred to as "proper one-entry one-exit subflowgraphs". For a more thorough discussion of D-primes, see [1]

Design Complexity, or $iv(G)$, is the cyclomatic complexity of a module's reduced flowgraph. The flowgraph, G, of a module is reduced to eliminate any complexity which does not influence the interrelationship between design modules. This complexity measurement reflects the modules calling patterns to its immediate subordinate modules [**?**].

1.2 Halstead

Another commonly used collection of software metrics are the Halstead Metrics [2]. They are named after their creator, Maurice H. Halstead. Halstead felt that software (or the writing of software) could be related to the themes which were being advanced at that time in the psychology literature. He created several metrics which are meant to encapsulate these properties; these metrics can be extracted by use of the McCabe IQ tool mentioned previously, and are discussed in detail below.

Halstead began by defining some basic measurements (these measurements are collected on a per module basis):

$$\begin{aligned}
\mu_1 &= \text{number of unique operators} \\
\mu_2 &= \text{number of unique operands} \\
N_1 &= \text{total occurrences of operators} \\
N_2 &= \text{total occurrences of operands} \\
\mu_1^* &= \text{potential operator count} \\
\mu_2^* &= \text{potential operand count}
\end{aligned}$$

These six metrics are self explanatory, with the possible exception of the potential operator/operand counts. Halstead defines μ_1^* and μ_2^* as the *minimum* possible number of operators and operands for a module. This minimum number would occur in a (potentially fictional) language in which the required operation already existed, possibly as a subroutine, function, or procedure. In such a case, $\mu_1^* = 2$, since at least two operators must appear for any function; one for the name of the function, and one to serve as an assignment or grouping symbol. μ_2^* represents the number of parameters, without repetition, which would need to be passed to the function or procedure.

Using these measurements, Halstead defined the *length* of a program P as:

$$N = N_1 + N_2$$

The vocabulary of P is:

$$\mu = \mu_1 + \mu_2$$

The *volume* of P, akin to the number of mental comparisons needed to write a program of length N, is:

$$V = N * log_2\mu$$

V^* is the potential volume - the volume of the minimal size implementation of P.

$$V^* = (2 + \mu_2{}^*)log_2(2 + \mu_2{}^*)$$

The *program level* of a program P with volume V is:

$$L = V^*/V$$

The inverse of level is *difficulty*:

$$D = 1/L$$

According to Halstead's theory, we can calculate an estimate $\hat{L}$ of L as:

$$\hat{L} = 1/D = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$$

The intelligence content of a program, I, is:

$$I = \hat{L} * V$$

The effort required to generate P is given by:

$$E = \frac{V}{\hat{L}} = \frac{\mu_1 N_2 N log_2\mu}{2\mu_2}$$

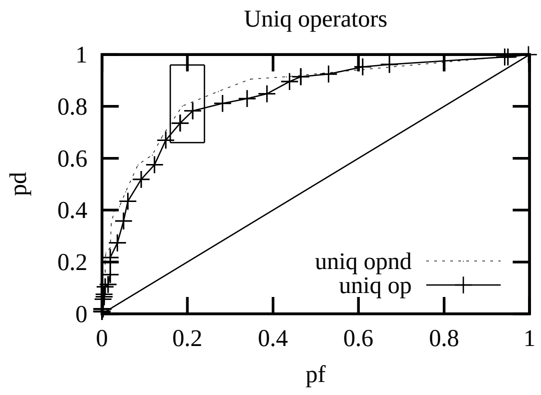
where the unit of measurement E is elementary mental discriminations needed to understand P.

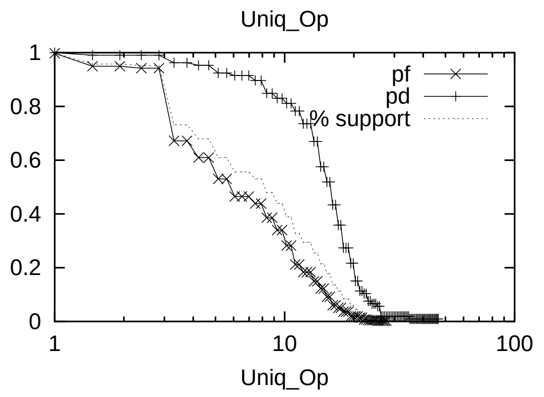
The required programming time T for a program of effort E is:

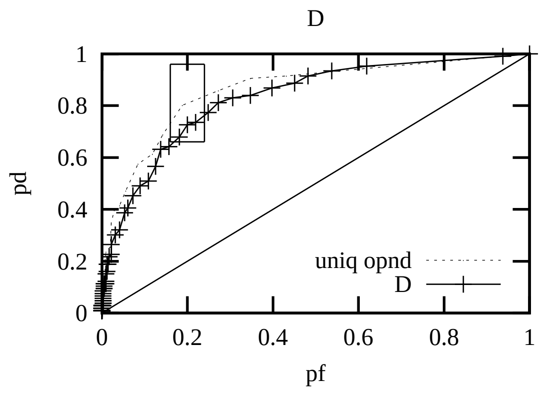
$$T = E/18 seconds$$

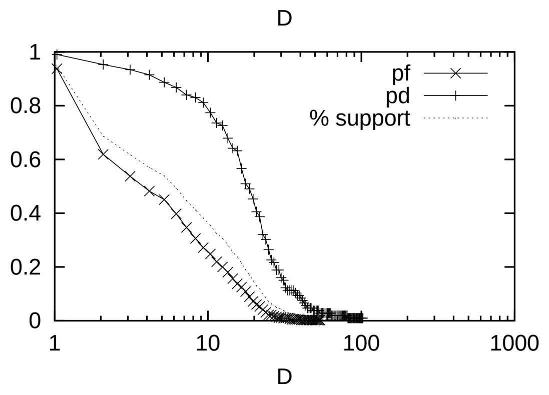
References

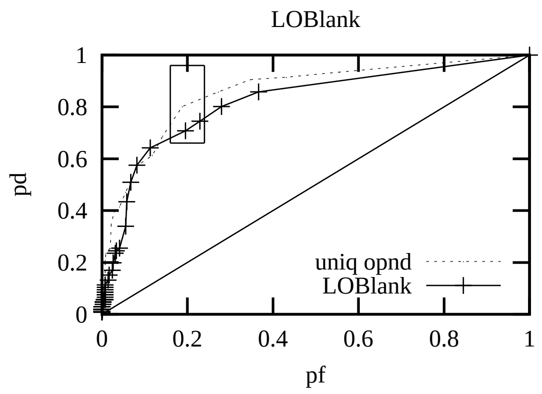
1. N. E. Fenton and S. Pfleeger. *Software Metrics: A Rigorous & Practical Approach (second edition).* International Thompson Press, 1995.
2. M. Halstead. *Elements of Software Science.* Elsevier, 1977.
3. T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, Dec. 1976.

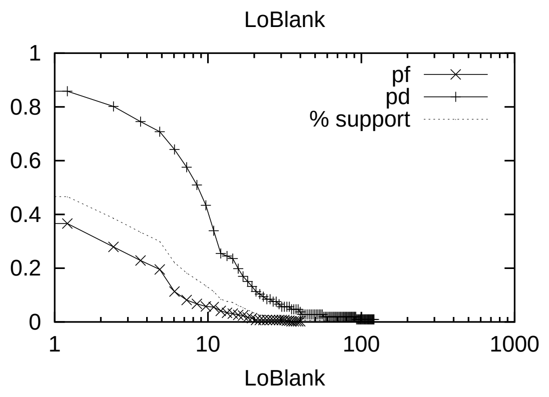

Uniq operators
pd
pf
uniq opnd
uniq op

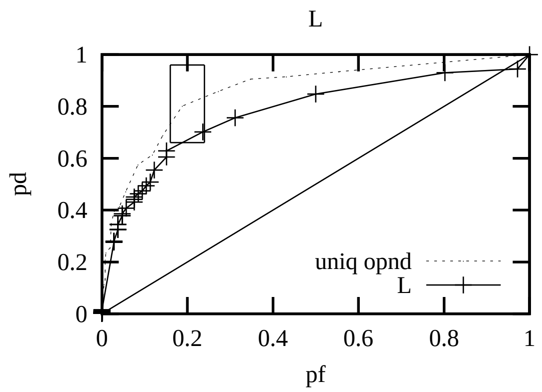

Uniq_Op
pf
pd
% support
Uniq_Op

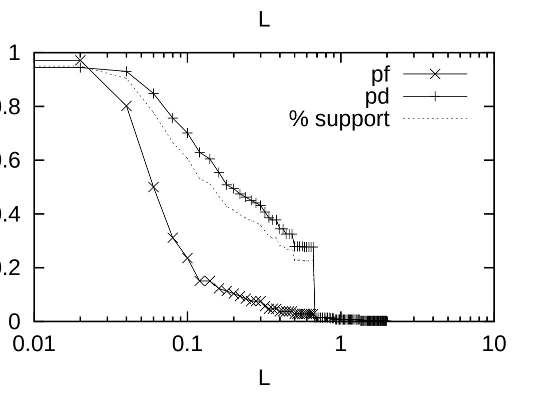

D
pd
pf
uniq opnd
D

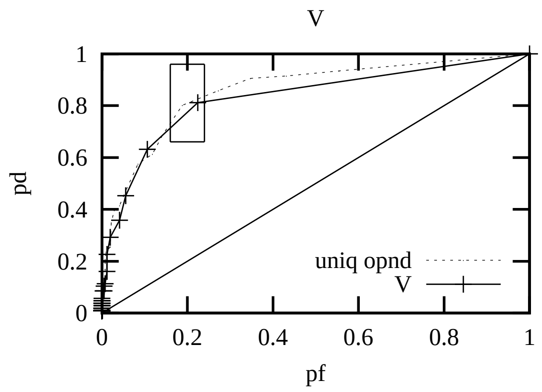

D
pf
pd
% support
D

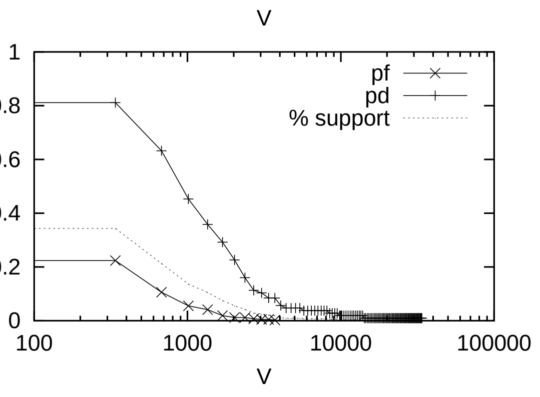

LOBlank
pd
pf
uniq opnd
LOBlank

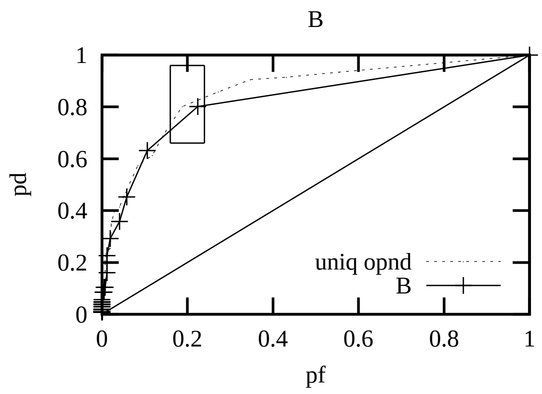

LoBlank
pf
pd
% support
LoBlank

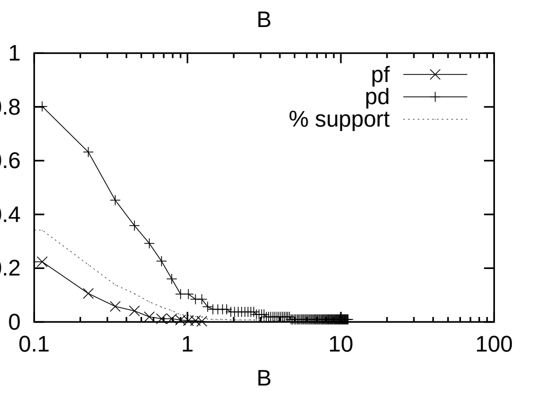

L
pd
pf
uniq opnd
L

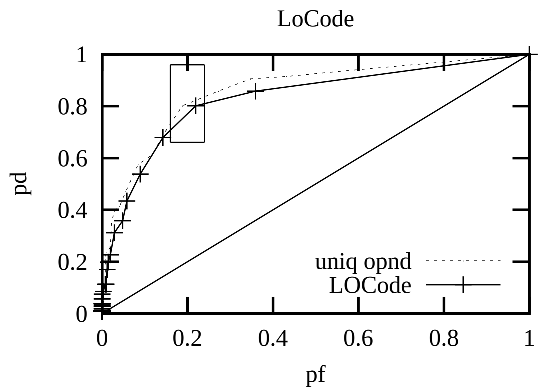

L
pf
pd
% support
L

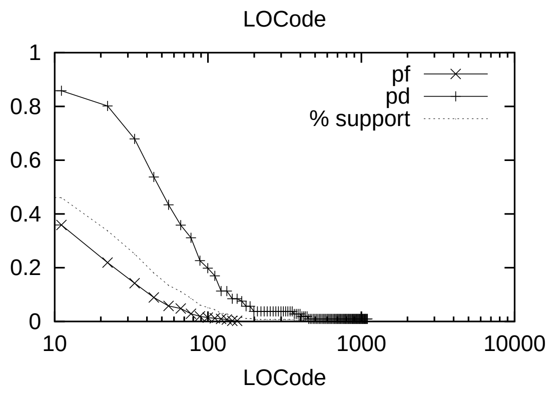

V
pd
pf
uniq opnd
V

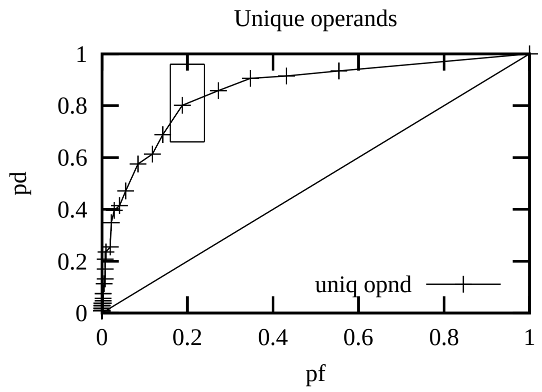

V
pf
pd
% support
V

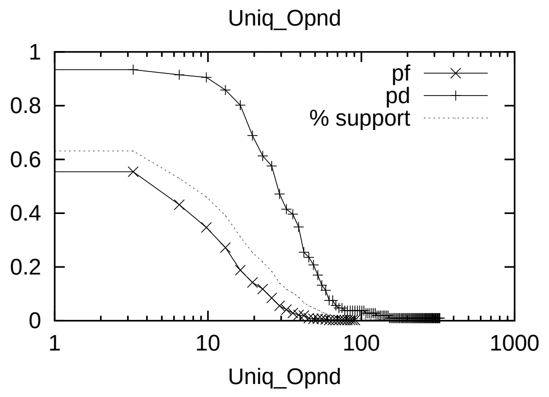

B
pd
pf
uniq opnd
B

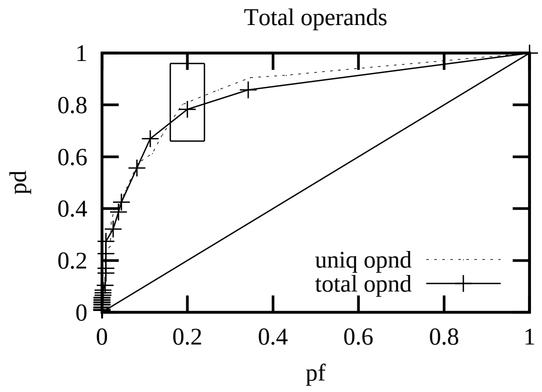

B
pf
pd
% support
B

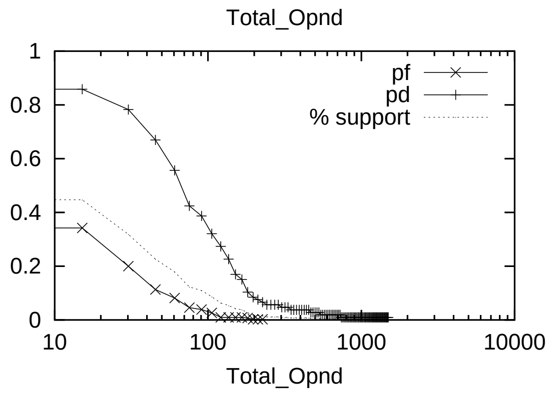

LoCode
pd
pf
uniq opnd
LOCode
0
0.2
0.4
0.6
0.8
1

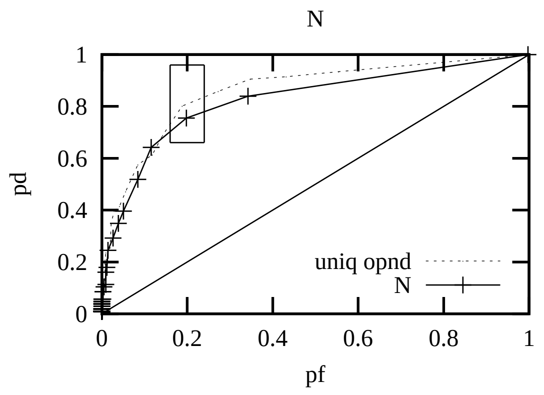

LOCode
pf
pd
% support
LOCode
10
100
1000
10000
0
0.2
0.4
0.6
0.8
1

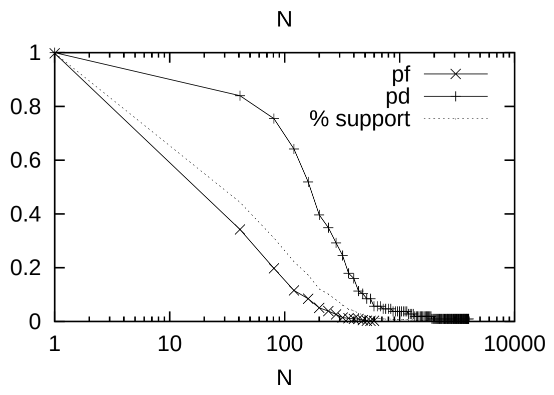

Unique operands
pd
pf
uniq opnd
0
0.2
0.4
0.6
0.8
1

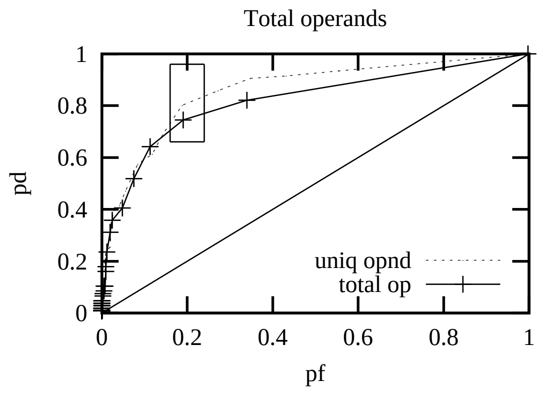

Uniq_Opnd
pf
pd
% support
Uniq_Opnd
1
10
100
1000
0
0.2
0.4
0.6
0.8
1

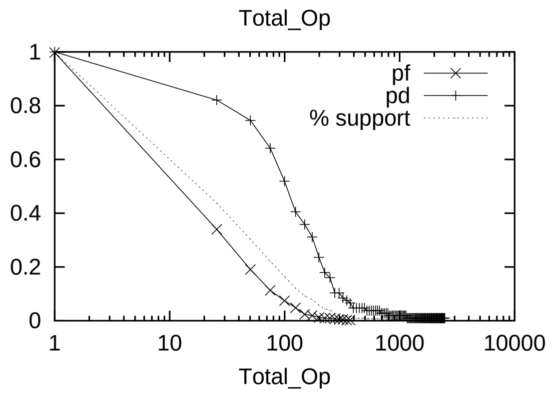

Total operands
pd
pf
uniq opnd
total opnd
0
0.2
0.4
0.6
0.8
1

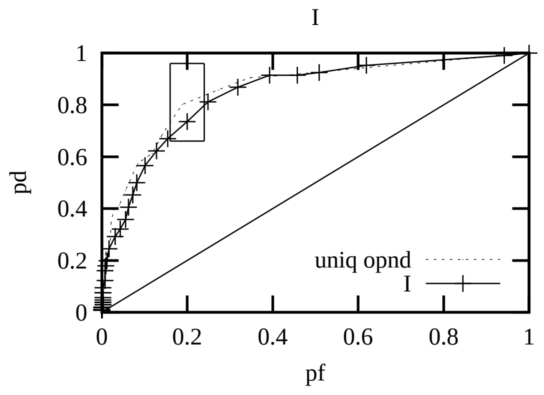

Total_Opnd
pf
pd
% support
Total_Opnd
10
100
1000
10000
0
0.2
0.4
0.6
0.8
1

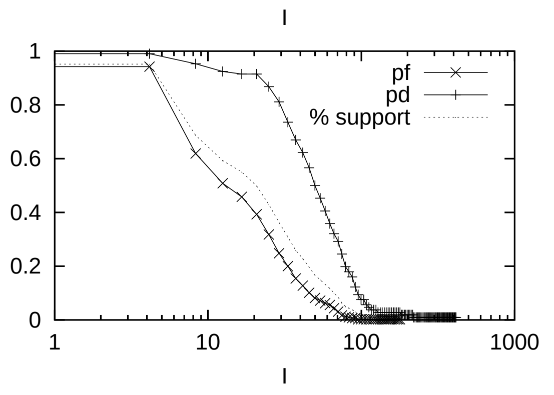

N
pd
pf
uniq opnd
N

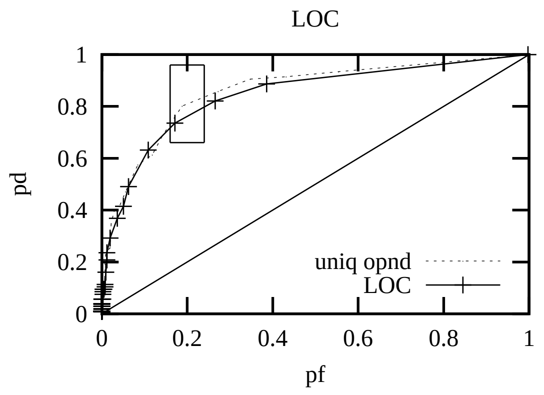

N
pf
pd
% support
N

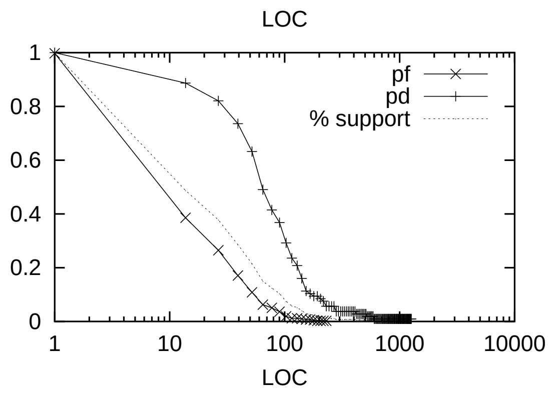

Total operands
pd
pf
uniq opnd
total op

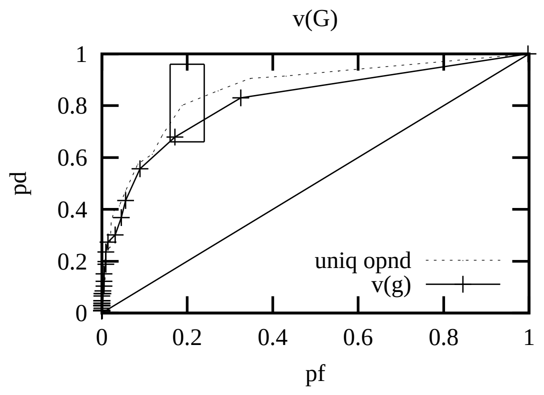

Total_Op
pf
pd
% support
Total_Op

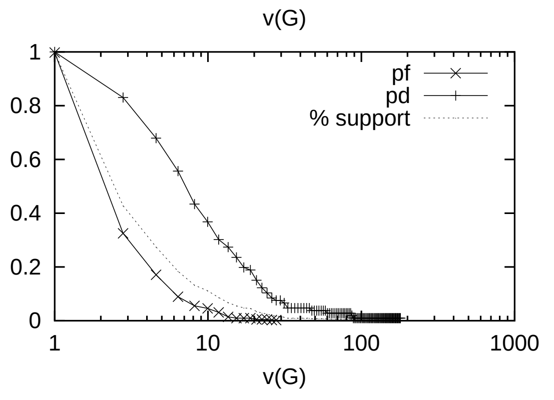

I
pd
pf
uniq opnd
I

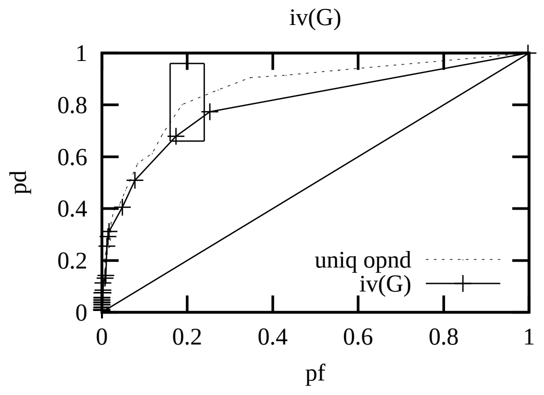

I
pf
pd
% support
I

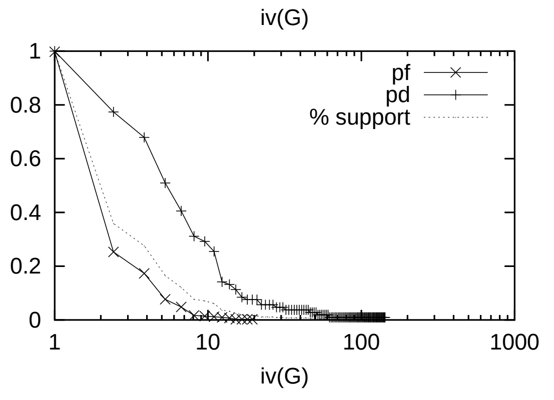

LOC
pd
pf
uniq opnd
LOC
0
0.2
0.4
0.6
0.8
1

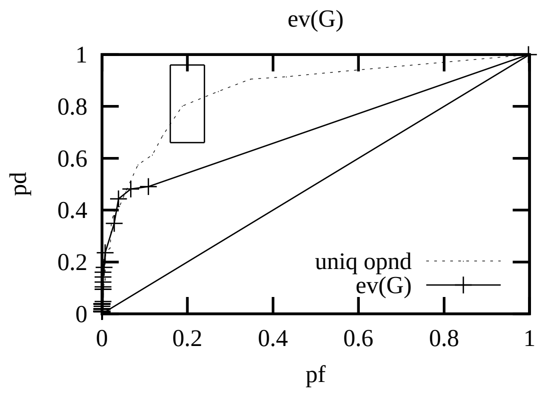

LOC
pf
pd
% support
LOC
1
10
100
1000
10000

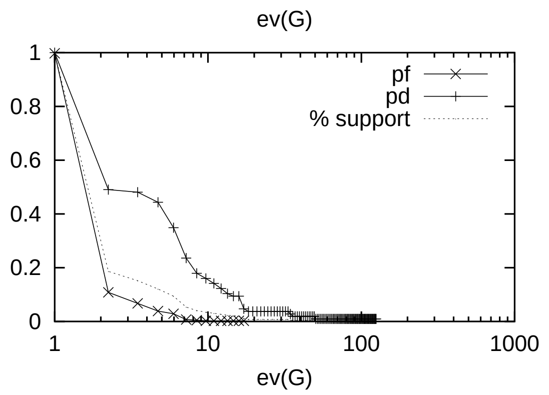

v(G)
pd
pf
uniq opnd
v(g)

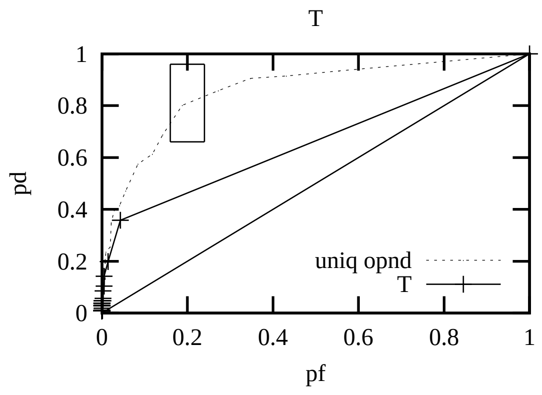

v(G)
pf
pd
% support
v(G)
1
10
100
1000

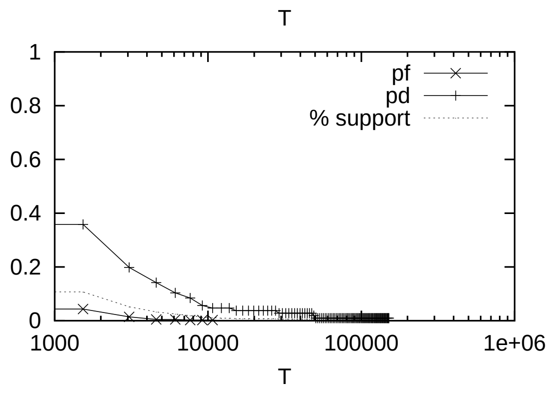

iv(G)
pd
pf
uniq opnd
iv(G)

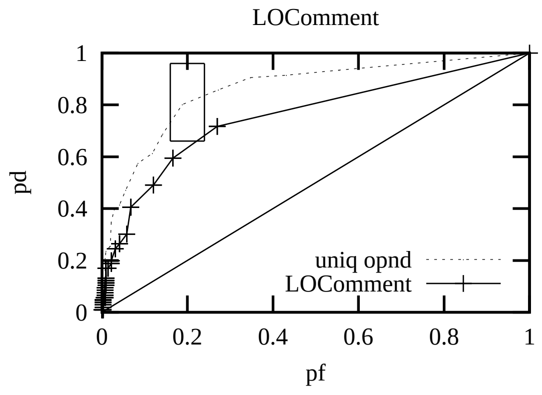

iv(G)
pf
pd
% support
iv(G)
1
10
100
1000

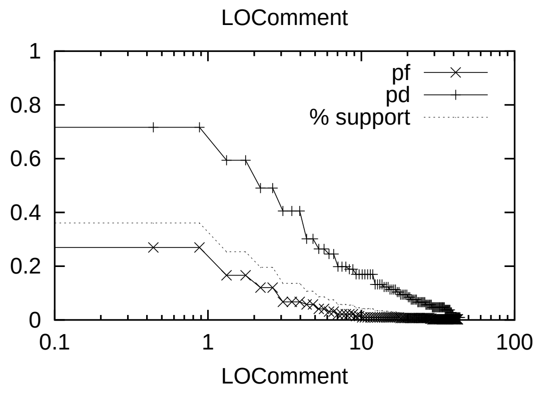

ev(G)
pd
pf
uniq opnd
ev(G)
0
0.2
0.4
0.6
0.8
1

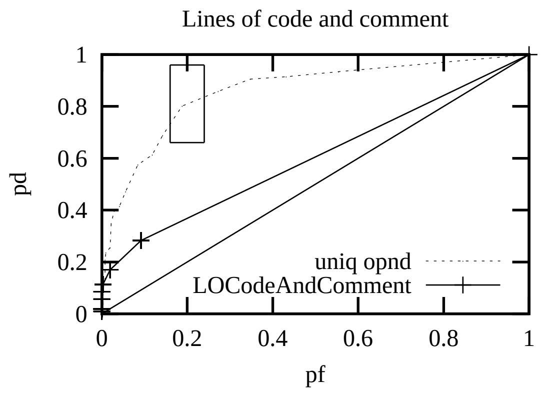

ev(G)
pf
pd
% support
1
10
100
1000
0
0.2
0.4
0.6
0.8
1

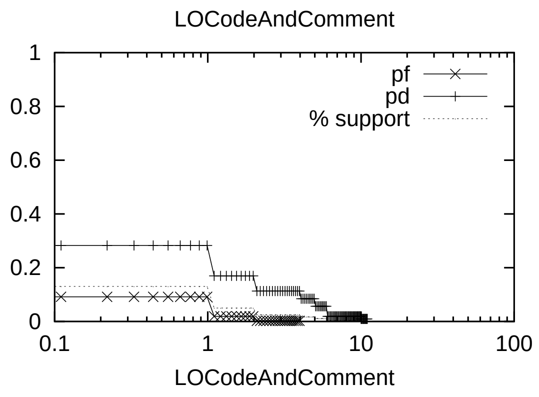

T
pd
pf
uniq opnd
T
0
0.2
0.4
0.6
0.8
1


T
pf
pd
% support
1000
10000
100000
1e+06
0
0.2
0.4
0.6
0.8
1


LOComment
pd
pf
uniq opnd
LOComment
0
0.2
0.4
0.6
0.8
1


LOComment
pf
pd
% support
0.1
1
10
100
0
0.2
0.4
0.6
0.8
1


Lines of code and comment
pd
pf
uniq opnd
LOCodeAndComment
0
0.2
0.4
0.6
0.8
1


LOCodeAndComment
pf
pd
% support
0
0.2
0.4
0.6
0.8
1
0.1
10
100
LOCodeAndComment