第43卷 第6A期
2016年6月

计 算 机 科 学
Computer Science

Vol. 43 No. 6A
June 2016

软件相似性分析算法的研究综述

黄寿孟 高华玲 潘玉霞
(三亚学院计算机教学部 三亚 572022)

摘 要 软件相似性分析算法是为了更好地保护软件的知识产权。此算法并不会加固程序以增加其抵御攻击的能力,而是对两个或两个以上的程序进行比较,判断是否相互包含。该算法有重复代码筛选、软件作者鉴别、软件"胎记"和剽窃检测,它们最本质的操作就是直接处理程序的源码或二进制可执行文件,将其转换成一种更易于处理的表示形式,从而确定两个程序(或者程序片段)之间的相似度,或是其中一个(部分或全部)是否包含了另一个。最后总结出此类算法的通用格式,并对每种算法作出相应的分析综述表。

关键词 软件相似性,相似度算法,重复代码筛选,作者鉴别,软件"胎记",剽窃检测
中图法分类号 TP301.6 文献标识码 A

Summary of Research on Similarity Analysis of Software

HUANG Shou-meng GAO Hua-ling PAN Yu-xia
(Department of Computer Instruction, Sanya University, Sanya 572022, China)

Abstract The similarity analysis of the software is to protect the intellectual property rights of software. This algorithm will not strengthen the program to increase its ability to resist the attack. It compares two or more than two procedures, to determine whether each contains. This algorithm includes clone detection, software forensic, software birthmarking and plagiarism detection. The most essential operation is the source code or binary executable file of the program. Program is converted into a more easily processing representation, in order to determine the similarity between two (or program fragments) programs, or one of programs (in whole or in part) contains the other. Finally the general form of the algorithms was summarized and the corresponding analysis of each algorithm was made.

Keywords Software similarity, Similarity algorithm, Clone detection, Software forensic, Software birthmarking, Plagiarism detection

1 引言

当代社会是网络和信息数字化技术日益成熟、知识经济广泛流行的社会。保护软件的知识产权除了绝对权威和起根本作用的法律手段之外,重要的是要有强大保障作用的技术手段。目前采用技术手段的软件保护算法有基于代码转换的代码混淆算法、防篡改算法和水印算法,这些算法是把一个未经保护的程序作为输入,使用一种或多种算法对其中的代码进行转换,最终输出一个被保护得更好的程序[1]。其实每一种算法的基本防御手段都是将一个相关对象未被保护的全域作为输入,然后产生一个其中某个对象被更好地保护了的全域作为输出。然而我们最关心的问题是:程序A和程序B共同拥有的一些属性是否能够表明它们来源于同一段代码?一种常见的攻击就是代码剽窃,即攻击者从你的程序中复制了一个重要的代码库,并把它用到了他自己的程序代码中。这种代码剽窃行为可以通过剽窃检测方法来确认,并在学术论文检测、游戏软件产业等领域中得到了广泛的应用。

其实在软件知识产权保护方法中,通常采用软件相似性分析算法,除了以上所说的剽窃检测(Plagiarism Detection)之外,还有另外3种软件产权保护方法:软件作者鉴别(Software Forensic)、重复代码筛选(Clone Detection)、软件"胎记"(Software Birthmarking)。剽窃检测是将某个软件与同类中其他软件相比较,检测此软件是否(部分或全部地)抄自其他软件;软件作者鉴别是将所有可能的作者编写的程序汇集起来,然后与给定的软件相对照,找出编程风格与其最相似的软件作者;重复代码筛选是用来在程序中找出那些因程序员使用"复制_粘贴_修改"而在程序中遗留的相似代码片段;软件"胎记"是指在程序的二进制代码层面检测两个程序是否由同一个代码改编而来。

2 相似度(Similarity)的计算

通常,两段代码很"相似"时表示它们的语义相似或它们源代码中的文字相似或者它们经过某些特定的代码转换之后会相似,那么在计算相似度时需要从两个或多个程序中提取"特征",再比较这些特征的相似性。这些特征的表示形式各有差异,有些只是提取了特征组成的序列或集合,有些可能是

本文受海南省教育厅项目(Hnky2015-51),三亚市院地科技合作项目(2013YD43)资助。
黄寿孟(1975—),男,硕士,讲师,主要研究方向为现代教育技术、计算机基础教学,E-mail:huang123888@126.com;高华玲(1980—),女,硕士,讲师,主要研究方向为数理逻辑、人工智能;潘玉霞(1983—),女,硕士,讲师,主要研究方向为人工智能、专家系统。

程序的树或者图的表示形式。但是不管这些特征是以什么形式存在的，分析相似度算法中必须用某种方式把它们逐对进行相似性比较。

2.1 序列的相似度

由于序列在许多领域（比如生物信息学）中都是常见的结构，因此文献中评估序列相似度的方法是非常多的，最常用的计算方法是基于 Hamming Distance（汉明距离）和 Levenshtein 距离（编辑距离）。当两组等长序列的相似度可以定义为对应位置上相同的元素在整个串中所占的比例时，就用汉明距离和相似度计算；当被比较的两个序列的长度不相等时，就将一个串转换成另一串最少需要进行的操作步数来表示它们之间的相似度，即编辑距离。

定义 1（汉明距离和相似度） 令 f 为从文档 d 中计算出的一个特征向量 $f(d)=\langle d_1,\cdots,d_n\rangle$ 的函数，令 p 和 q 分别为算出特征向量 $f(p)=\langle p_1,\cdots,p_n\rangle$ 和 $f(q)=\langle q_1,\cdots,q_n\rangle$ 的文档。那么 p 和 q 之间的汉明距离就可以定义为：

$$distance(p,q)=|\{\forall 1\leqslant i\leqslant n\cdot p_i\neq q_i\}|$$

而 p 和 q 之间的相似度就可以定义为：

$$similarity(p,q)=1-\frac{distance(p,q)}{n}$$

定义 2（编辑距离和相似度） 两个序列 p 和 q 之间的编辑距离 $distance(p,q)$ 就是使用插入、删除以及替换等操作将 p 转换成 q 所需的最少步骤。p 和 q 之间的相似度可以定义为：

$$similarity(p,q)=1-\frac{distance(p,q)}{\max(|p|,|q|)}$$

2.2 集合的相似度

集合间的相似度与两个文档间的包含度（containment）[2] 的相关计算定义如下。

定义 3（相似度和包含度） 两个文档 p 和 q 间的相似度 $similarity(p,q)$ 可以这样定义：

$$similarity(p,q)=\frac{|f(p)\cap f(q)|}{|f(p)\cup f(q)|}$$

其中，$f(d)$ 是计算文档 d 的一个特征集合的函数。与此相似，p 在 q 中的包含度 $containment(p,q)$ 定义为：

$$containment(p,q)=\frac{|f(p)\cap f(q)|}{|f(p)|}$$

2.3 图的相似度

计算图的相似度的方法很多，可以把计算序列的编辑距离的标准沿用到图上：将一个图转换成另一个图最少需要多少步操作（插入、删除或替换图中的结点或边）；也可以沿用集合包含计算标准：根据两个图之间最大公共子图（Maximal Common Subgraph）的大小来计算图的相似度[3]。

定义 4（公共子图） 令 G、G_1 和 G_2 都是图。如果 G_1 中存在一个子图与 G 同构，且 G_2 中也存在一个子图与 G 同构，则称 G 是 G_1 和 G_2 的公共子图。

如果 G 是 G_1 和 G_2 的公共子图，且不存在一个所含结点数比 G 还多的 G_1 和 G_2 的公共子图 G'，那么就称 G 是 G_1 和 G_2 的最大公共子图，记为 $G=mcs(G_1,G_2)$。

定义 5（图的相似度和包含度） 令 $|G|$ 为图 G 中结点的数量，图 G_1 和 G_2 间的相似度 $similarity(G_1,G_2)$ 定义为：

$$similarity(G_1,G_2)=\frac{|mcs(G_1,G_2)|}{\max(|G_1|,|G_2|)}$$

图 G_1 在 G_2 中的包含度 $containment(G_1,G_2)$ 定义为：

$$containment(G_1,G_2)=\frac{|mcs(G_1,G_2)|}{|G_1|}$$

3 相似性算法综述

3.1 重复代码筛选

重复代码筛选是指寻找程序中相似代码片段的过程。在检测阶段的工作完成之后还会有一个抽象阶段，在这个阶段中相似的代码会被抽象到一个函数中，并将之前使用这些代码的地方全都换成对新函数的调用。重复代码筛选的算法（见算法 1）框架中，P 表示一个程序，$threshold$ 是两段被认为是重复的代码之间应具有的最小相似度，$minsize$ 是使得将其抽象成一个新函数这一动作有意义的代码片段的最小尺寸。

算法 1 Detect(P,threshold,minsize)

(1) 构造 P 的某种表示形式 rep，这种表示形式应该便于我们找出程序中相似的部分。找出程序中足够相似并且足够大（使得把它们抽象成一个函数是有意义的）的代码片段对：

res←∅

rep←方便处理的 P 的表示形式

对于每对代码片段 f,g∈rep 且 f≠g，执行如下操作：

if similarity(f,g)>threshold && size(f)≥minsize && size(g)≥minsize then

res←res∪⟨f,g⟩

(2) 把上个步骤中得到的代码对从它们原来的函数中剥离出来，并将其抽象成一个新的函数，同时把原来的代码替换成用相应的参数对新函数的调用：

对于每对代码片段⟨f,g⟩∈res，执行如下操作：

b(r)←f 和 g 参数化后的函数

P←P∪b(r)

将 f 替换成调用 b(r1) 的语句，同时将 g 替换成调用 b(r2) 的语句

(3) 返回 res 和 P

重复代码筛选是一种代码维护工具，其所作所为就是为了改进程序源码的质量。有些重复代码筛选器可以直接对源码进行操作，但是大多数情况下，重复代码筛选器会先把程序转换成一种更为高级的表示形式，比如词法单元流、抽象语法树或者程序依赖关系图。

3.2 软件作者鉴别

软件作者鉴别指通过源程序代码（片段）寻找编程风格，由此判断其程序员。随着编程技术的进步和对程序员要求的不断完善，可以收集到的程序员的习惯有缩进代码的方式、选择变量名的方式、能允许的函数和模块增长到的规模，以及对于一些常见的问题倾向于用什么算法和数据结构去解决。尽管程序员的编程风格会随着经验的增加而改变，但是它并不会变得完全不同，总会有些特征是保持不变的。软件作者鉴别算法（见算法 2）框架可分为 3 个步骤：确定哪些代码特征可以被认为是软件作者的良好指示器，从所有可能作者的样本代码和被检查的程序中提取这些特征，比较这些特征集的相似性。A 是所有可能编写未知程序 P 的程序员组成的集合。对于任何一个程序员 $a\in A$，$U[a]$ 是他的样本程序代码组成的集合。

算法 2 Detect(A,U,P,threshold)

(1) 从不同程序员编写的海量程序中确定出一个特征集 C，它应该在不同作者编写的程序之间呈现出最大的变化倾向，而在同一个作者编写的不同程序间呈现出最小的变化倾向。

(2) 从各个程序员提供的样本程序以及 P 中收集相关特征：

sig←∅
对于每个作者 $a\in A$ 及代码样本集 $s\in U[a]$，进行如下操作：
profile[a] ←从 s 中提取的属于特征集 C 的特征
sig←从 P 中提取出来的属于 C 的特征
(3) 确定哪个程序员的 profile 记录与 P 的特征最为相符：
对于每个作者 $a\in A$，进行如下操作：
sim←similarity(sig, profile[a])
if sim>threshold，进行如下操作：
res←res∪⟨a, sim⟩
将 res 按相似度排序后返回

软件作者鉴别的大多数工作都是针对源码进行分析的。并不是所有从程序中提取出来的特征都有助于确定程序的编写者，所以一个非常重要的预处理步骤就是从海量的程序中确定哪些编程风格特征的确能反映程序员的个性特征，即哪些特征在同一个程序员编写的程序中变化最小而在不同程序员编写的程序中变化最大。下一个步骤是从可能的编写者的样本中收集这些特征的集合(该程序员的 profile)，并从被检查程序中也提取这些特征集合。为了使判断的结果更为准确，需要得到各个可能的编写者的有效的样本代码集，而实际上这一点因每个特定的程序员而实际情况操作起来特别困难。

3.3 剽窃检测

剽窃检测是检查程序代码是否抄袭别人的代码，检测系统的输出结果会更为复杂，可以将疑似抄袭程序很高的程序对中的两个程序用对照图的形式显示出来，并将各个相似的部分标上对应的颜色。剽窃检测的效率低于软件作者鉴别，但高于重复代码筛选。通用剽窃检测算法的基本思路是跟踪记录程序员的编程代码的变化情况，再与其他程序员的编程代码比较，并将其相似部分识别出来。算法 3 是剽窃检测算法概览，其中 U 是学生提交的程序的集合。

算法 3　Detect(U, threshold)

res←∅
对于每一对程序 $f,g\in U$，进行如下操作：
sim←similarity(f, g)
if sim>threshold then
res←res∪⟨f, g, sim⟩
res←按相似度对 res 排序
return res

3.4 软件胎记检测

软件胎记检测就是将程序中不会因语义保留的转换而改变的特性提取出来的过程，主要是从可执行代码而不是源代码中提取出来的。比如，假设程序 P 中所有的代码都被抄袭到了程序 Q 中，也就是说 Q 只是 P 改头换面后的另一个版本而已，尽管 Q 经过了足够的混淆，使得抄袭者可以争辩说 Q 是完全独立开发的，那么只要胎记检测到程序 Q 执行代码的行为与 P 的一模一样，或者一些函数或几个模块的二进制代码一样，就可以判断程序 P、Q 之间的相似度比例。主体实现代码如算法 4 所示。

算法 4　Detect(P, Q, threshold)

bmP←从 P 中提取的“特征”
bmQ←从 Q 中提取的“特征”
if similarity(bmP, bmQ)>threshold then
　　return “复制品”
else
　　return “非复制品”

4　算法分析

软件相似性算法的分析是指根据某一算法需要多少计算比较时间和其采用存储空间的大小来判定与分析其优劣。对于软件相似性的所有算法，可以预测其适合在什么样的环境中有效地运行，当然对于采用不同算法的有效性，也相应做出不同比较，下面给出的算法的详细操作步骤请查阅相关引用文献。

4.1 基于 k-gram 的分析

比较两个文档的 k-gram 集合是一种很流行的比较文档相似性的方法，可以用在对文本文档或源代码的剽窃检测、软件作者辨识以及二进制可执行文件的“胎记”检测中。算法 $SSSWA_{winnow}$[4-6] 有选择地记录 k-gram 的 hash 集，因为从原始文档得到的 hash 集中每个 hash 出现的次数应该是相近的，所以不必要保存所有的 hash，只要保留少数几个 hash 用于比对就够了；在源码级剽窃检测系统 MOSS 中使用算法 $SSSWA_{MOSS}$ 将 n 个程序逐对进行比较和检查，以提高整个检测系统的性能；若比较的是两个二进制可执行代码程序，则采用基于 k-gram 的“胎记”$SSMC_{kgram}$ 算法。

4.2 基于 API 的分析

一个程序使用 API 标准库或者系统调用的方式，程序的“胎记”算法就难以被混淆，因为 API 不易被混淆者拆成好几段，而且攻击者不能随便在程序中添加、删除或替换调用 API 语句。面向对象的“胎记”算法 SSTNMM 是 Tamada 和其他研究者合作提出的一系列基于 Java API 数据类型和方法调用的“胎记”提取算法：算法 $SSTNMM_{SMC}$[7] 根据类中调用方法的序列算出“胎记”，从头到尾地扫描一遍类文件，把其中使用的每种方法都记录下来，再将其中对已知类的方法的调用全都提取出来；算法 $SSTNMM_{Is}$ 根据类从 root 类到该类的继承路径算出“胎记”，某个类的胎记就是它的所有超类(不包括那些非已知类)的继承顺序；算法 $SSTNMM_{UC}$ 根据类中使用的数据类型(其他类)算出“胎记”，它计算的类是一个经过排序的已知类的顺序列表。对于动态函数调用“胎记”算法 SSTONMM[8,9]，根据用特定的输入数据运行程序得到的 trace 记录(在 Unix 系统中使用 ltrace 命令来得到程序对系统和库函数的调用记录)，再从已知库函数的调用信息中提取“胎记”，通过分析各个 API 出现的频率来比较软件相似性；动态 k-gram API“胎记”算法 SSSDL[10,11] 结合前面几种“胎记”算法，用特定的输入执行程序，收集已知 API 被调用的顺序，而忽略那些不能被视为具有良好的相似性的普通 API 调用，构造余下 API 调用序列的 k-gram 集并把结果作为程序的“胎记”，不必保存整个 trace 记录。

4.3 基于树的分析

源代码状态的程序本身具一定层次结构，即树状结构。使用树作为程序表示形式的软件相似性分析算法并不多，常用的是基于抽象语法树的重复代码筛选算法 SSEFM[12]：抽象语法树与任何一种分析无关，它只记录原始程序中的信息，也就是说语法分析树中记录的是对代码进行分析后得到的结果。若某棵代码树的结构拥有大量的结点，并能多次在抽象语法树中匹配成功，几乎没有子树，那么说明存在某些重复代码。用树结构表示重复代码那部分子树，可以将其抽象成某个函数，并把程序(抽象语法树)中所有与之匹配的部分内容

(子树)替换成对这个函数的调用。按照从最有可能表示重复代码的子树到最不可能表示重复代码的子树的顺序遍历整棵树,从中选出重复代码的子树。

4.4 基于图的分析

程序拥有类似于图的结构,函数可以很自然地用控制流图表示出来,函数中各个语句之间的依赖关系可以表示为依赖关系图(PDG)。基于 PDG 的重复代码筛选算法 SSKH[13],构造程序中各个函数的 PDG,从每一对匹配结点出发,沿着表示依赖关系的边计算一个后向切片,以此找出两个 PDG 间的同构子图;基于 PDG 的剽窃检测算法 SSLCHY[14]使用通用的同构子图匹配算法而非切片技术来寻找同构子图,并且为了提高性能,在比对前先进行预处理步骤,去掉那些明显不是候选剽窃代码的部分;整个程序的动态"胎记"算法 $SSMC_{WPP}$[15,16]将控制流图表示成一个有向无环图(DAG),去掉图中所有的终结符,把 DAG 中剩下的部分当作整个程序的胎记,在保持较好的压缩率的前提下,这个胎记还能够反映出程序的重复执行模式。

4.5 基于软件度量的分析

软件复杂度度量是通过统计程序不同方面的特征,来获得关于该程序的一些信息,从量化的方面刻画软件的一些本质特征。算法 SSKK[17]用寻找代码片段量化数据的相似度的方式进行重复代码筛选,假设程序中两段代码是相似的(也就是潜在的重复代码),那么进行度量的结果(量化数据)一定是相似的;若两段代码不是重复代码,那么度量的结果也有所不同,其算法先构造程序的抽象语法树,再自下而上地遍历整棵树,计算其中每一棵子树的复杂度量化数据集,然后逐对地比较每棵子树的量化数据,量化结果最为相似的就最可能是重复的代码。算法 SSLM[18]则是通过比较度量各个程序员代码样本得到的量化数据来完成软件作者鉴别工作,对程序源码的度量分成 3 类:代码布局度量、编程风格度量、程序结构度量,采用度量指标区别程序员方法,即先列出所有可能的度量指标组,用这些度量指标组分析评估数据,再与程序员指标组匹配,得到最接近指标组。

结束语 本文介绍的软件相似性算法可以归纳成下面通用的模式,其中 P 和 Q 代表程序,R 是程序的表示形式:

```
repP ← 转换 P 为表示形式 R
repQ ← 转换 Q 为表示形式 R
for 每个代码片段 p∈repP 且 q∈repQ do
  if similarity(p,q) > 阈值 then
...
```

不同的重复代码筛选、软件作者鉴别、"胎记"和剽窃检测的算法,其应用领域也有所不同,但是必须在不同环境中做出权衡取舍:采用简单还是复杂的程序表示形式;采用较小有损还是较大无损的程序表示形式;相似度比对时是浅度比对还是深度比对;采用准确但性能较差的算法还是执行速度快的启发式算法;程序员的批量修改能力强还是差。很多算法是基于整个程序依赖关系图(PDG)进行检测工作的,这是因为 PDG 确实是强有力的程序表示形式,分析人员可以从中方便地提取出任何两个语句之间的次序关系,但是相对于程序度量的量化结果,PDG 构造起来更困难,也需用更多的内存存放构造结果。当然重复代码筛选、软件作者鉴别、"胎记"和剽窃检测技术在许多方面有一定的共通性,具体情况如表 1 所列。

表 1 相似性分析算法比较

类别	算法	程序表示形式	状态
重复代码筛选	SSKH	PDG	静态
	SSKK	度量程序后得到的量化数据	静态
	SSEFM	抽象语法树的所有子树构成的集合	静态
软件作者鉴别	SSLM	对程序进行度量后得到的数据	静态
剽窃检测	SSLCHY	PDG	静态
	$SSSWA_{MOSS}$	源码文本的 k-gram 集	静态
"胎记"	SSSDL	API 函数被调用次序的 k-gram 集	动态
	$SSMC_{kgram}$	字节码的 k-gram 集	静态
	$SSTNMM_{SMC}$	API 函数调用在源码中出现顺序	静态
	$SSTNMM_{IS}$	类的继承顺序	静态
	$SSTNMM_{UC}$	类的集合	静态
	SSTNMM	API 函数被调用的频率	动态
	$SSMC_{WPP}$	图	动态

总之,希望表 1 中的综述信息有助于今后的检测工作,能对以后交叉研究工作带来某些提示与灵感,比如采用动态技术进行重复代码检测、软件作者鉴别或剽窃检测的算法还有待研究;抽象语法树的子树分析方法能否应用于剽窃检测技术等。

参 考 文 献

[1] Collberg C, Nagra J. 软件加密与解密[M]. 崔孝晨,译. 北京:人民邮电出版社,2012

[2] Broder A Z. On the resemblance and containment of documents [C]// Compression and Complexity of Sequences (SEQUENCES'97). IEEE,1997:21-29

[3] Bunke H, Shearer K. A graph distance metric based on the maximal common subgraph[J]. Pattern Recognition Letters, 1998, 19:255-259

[4] Aiken A. Moss—a system for detecting software plagiarism [EB/OL]. (2011-04-29)[2014-01-15]. www.cs.berkeley.edu/~aiken/moss.html

[5] Aiken A, Schleimer S, Auslander J, et al. Method and apparatus for indexing document content and content comparison with world wide web search service[P]. U.S. Assigned to the Regents of the University of California, 2004, 6:6757675

[6] Schleimer S, Wilkerson D, Aiken A. Winnowing: Local algorithms for document fingerprinting[C]// Proceedings of the 2003 SIGMOD Conference. 2003

[7] Tamada H, Nakamura M, Monden A, et al. Detecting the theft of Programs using birthmarks[J]. Nara Institute of Science and Technology, 2003

[8] Tamada H, Nakamura M, Monden A, et al. Design and evaluation of birthmarks for detecting theft of Java programs[C]// IASTED International Conference on Software Engineering. 2004:569-575

[9] Tamada H, Okamoto K, Nakamura M, et al. Dynamic software birthmarking to detect the theft of windows applications[C]// Proceedings of the international Symposium on Future software Technology. 2004

[10] Schuler D, Dallmeier V. Detecting software theft with API call sequence sets[C]// Proceedings of the 8th Workshop Software Reengineering. 2006

[11] Schuler D, Dallmeier V, Lindig C. A dynamic birthmark for Java [C]//22nd IEEE/ACM International Conference on Automated Software Engineering. 2007

(下转第 507 页)

息、制定行动方案并组织实施控制的周而复始的过程[10]。网络作战组织指挥能力包含了3个基本要素:指挥主体、指挥客体和指挥手段。指挥主体主要指指挥员及指挥机构;指挥客体是指网络作战战士。指挥主体通过一定的指挥手段来对指挥客体的作战活动进行组织指挥控制。从面向要素角度评估网络作战组织指挥能力,主要涉及指挥人员、指挥对象、指挥手段及指挥机构。

指挥人员指挥能力评估是指在指挥活动履行职责和发挥作用程度的评估。其能力素质主要包含察情能力、决策能力、组织能力和控制能力。

指挥对象操作能力、网络作战能力的发挥,最终是通过指挥对象的具体操作实现的,指挥对象既要服从指挥,又要适应网络空间环境的变化,协调一致,灵活对抗。其业务能力主要包含指令理解能力、履职尽责能力、主动创造能力、态势掌握能力和协同配合能力。

指挥机构战(技)术能力中指挥机构是实施指挥控制的基本条件。评估指挥机构主要从其隐蔽性、可靠性、适应性及生存能力4个方面考虑。

指挥方式与手段是主体与客体的一种媒介。网络作战指挥中,指挥方式是否恰当,以及指挥手段的功能和作用发挥的程度,都直接影响网络作战指挥控制效能的高低。在指挥方式方面,主要从指挥方式在网络作战过程中的转换条件和时机是否适宜,以及其发挥作用的程度、即方式适当度评估,时机适宜度;在指挥手段方面,主要从指挥手段在网络作战过程中的先进性及可靠程度评估。

3.5 网络作战保障能力

网络作战保障是指网络作战顺利开展的基础网络环境以及参与网络作战的作战力量,是实施网络作战的重要基础,形成网络作战能力的重要保证和条件。从其构成要素角度,基于基础网络支撑能力、作战人员能力素质、武器装备、法规制度4个方面对网络保障能力进行评估。

网络支撑能力是展开网络作战的基础,保证网络作战武器装备实现互联、互通、互操作的关键部分,是各种作战能力实现的物质基础。评估支撑能力主要从网络运行能力、网络维护能力、网络覆盖能力3个方面考虑。

网络作战人员是网络作战评估最为活跃的因素,也是网络作战保障能力重要的构成要素。网络作战人员主要涉及指挥人员与指挥对象,根据其担负的不同职责,要求具备相应的能力素质。有关能力的评估已经在网络作战指挥控制中进行了讨论,此处主要从网络作战特点角度出发对作战人员特定素质进行讨论,从敏锐的政治鉴别力、极高的业务素养、极强的问题洞察力、良好的协作精神和团队意识4个方面进行评估。

网络作战武器装备是网络作战的构成要素和物化形式,是网络作战赖以进行的物质基础以及网络作战能力的载体。参照美国工业界系统效能咨询委员会评价武器系统用的模型——ADC模型[11],主要根据武器装备的可用性、可信性和固有能力3大要素对武器装备进行评估。网络作战侦察、攻击、防御能力,是武器装备及其他网络作战力量共同作用的集中体现。因此,依据指标构建独立性原则,武器装备的固有能力这里不再讨论。将武器装备的可用性、可信性作为评估武器装备的两个指标,考虑到网络作战武器装备的复杂性和多样性,同时将武器装备的完备性及先进性作为另外两个评估指标。

法规制度是合法开展网络作战活动的依据,是执行网络作战的准绳,必须具备相对完善的法规制度。对法规制度的评估主要围绕法规制度的完善程度和法规制度对任务的适应程度两方面进行评估。

结束语 通过对网络作战能力评估指标体系的研究和确立,为有效地评估网络作战能力提供了参考和依据。确立指标体系化,下一步的工作就是对各项指标数据的获取及评估模型的选择,由于篇幅所限,这些问题不再论述。

参 考 文 献

[1] 徐志明,卢昱,邹利鹏,等. 空间信息网络对抗效能评估指标体系研究[J]. 计测技术,2005,25(2):11-13

[2] 贾珺,战晓苏,程文俊. 基于灰色关联分析的网络战综合能力评估[J]. 系统仿真仿报,2012,24(6):1185-1188

[3] 彭子枚. 网络攻击效能评估若干关键技术研究[D]. 长沙:国防科技大学,2011

[4] 贾爱国,王新辉. 信息优势的度量与效能评估[M]. 北京:军事科学出版社,2006

[5] 么健石,郑美,谭越郡. 空间信息对抗作战效能评估内容与指标体系[J]. 信息对抗学术,2014(1):37-41

[6] 任海泉. 军队指挥学[M]. 北京:国防大学出版社,2007

[7] 董亚卓,詹武,常歌,等. 指挥信息系统指标体系划分方法综述[J]. 电光与控制,2014,12(1):50-54

[8] 李大光,李万顺. 基于信息系统的网络作战[M]. 北京:解放军出版社,2010

[9] 徐小岩. 计算机网络战[M]. 北京:解放军出版社,2003

[10] 智韬. 编队级网络对抗效能评估研究[D]. 郑州:解放军信息工程大学,2009

[11] 陈健,滕克难,杨春周. 基于ADC法防卫装备体系打击效能评估模型研究[J]. 舰船电子工程,2014(3):32-35

(上接第470页)

[12] Ertl M A. Stack caching for interpreters[J]. SIGPLAN Not., 1995,30(6):315-327

[13] Komondoor R, Horwitz S. Using slicing to identify duplication in source code[C]// Proceedings of the 8th International Symposium on Static Analysis. 2001:40-56

[14] Liu Chao, Chen Chen, Han Jia-wei, et al. Gplag: detection of software plagiarism by program dependence graph analysis[C]// KDD'06. ACM, 2006:872-881

[15] Myles G. Software Theft Detection Through Program Identification[M]. University of Arizona, 2006

[16] Myles G, Collberg C. Detecting software theft via whole program path birthmarks[C]// 7th International Conference Information Security. 2004

[17] Kontogiannis K. Evaluation experiments on the detection of programming patterns using software metrics[C]// Working Conference on reverse Engineering. 1997:1-44

[18] Krsul I, Spafford E. Authorship analysis: Identifying the author of a program: CSD-TR-94-030[R]. Computer Science Deparment, Purdue University, 1994