一种基于粒子群算法的安卓应用体积优化方法

周少杰 [1]
ZHOU Shaojie

摘 要 随着移动应用功能迅速增长，安装包的体积也不断增大，导致手机经常因内存空间不足从而无法正常安装应用。代码混淆技术通过对 Java 字节码进行重构和修改，在提高程序安全性的同时也有利于减少应用体积。然而，目前主流的 R8 和 ProGuard 代码混淆器，其混淆规则众多，不同应用在不同混淆策略下，体积优化差距较大。为此，文章设计了一种基于粒子群算法的安卓应用体积优化方法。该方法以应用体积优化为目标，通过粒子群算法的速度更新和位置更新，搜索最佳策略，为 Java 字节码提供最合适的混淆规则组合。实验结果表明，该方法相对于 R8 和 ProGuard 混淆器的核心规则，能够获得 8.3% 的体积优化效果，有利于生成更小的安卓应用。

关键词 体积优化；R8；ProGuard；粒子群优化算法；安卓应用

doi：10.3969/j.issn.1672-9528.2025.04.019

0 引言

智能手机的普及为移动应用程序的发展提供了强大的硬件基础和广阔的市场空间 [1]。数十亿用户依赖移动应用程序进行社交媒体、移动支付 [2]、获取资讯等。在移动应用市场竞争激烈且用户需求不断变化的环境下，开发者持续提供新功能可以为用户带来新鲜感和更多价值，从而提高用户对应用程序的关注度、使用率和忠诚度，吸引更多新用户下载和使用，在市场竞争中占据优势地位。然而，随着新功能的增加，应用程序的字节码文件也相应增大 [3]。

字节码混淆技术可以有效提升代码安全性，并有效防止 MATE 攻击。除更具安全性，还可以减小代码体积。众多混淆器具备丰富的优化能力，能有效实现无用代码消除、常量折叠、方法内联、代码压缩以及窥孔优化等操作。这种应用于移动端应用程序的大小优化措施不仅可以提升用户体验，减少对移动设备的空间占用，还能提高用户对应用的留存率。

目前，Android R8 和 ProGuard 是常用字节码混淆，且支持的混淆规则数量达到了 59 个。如进行排列组合，将有 2^{59} 种可能性，此外，一些混淆规则需要设置相应的类名、方法名、字段和过滤器等参数，使得可能性更加复杂，超过 2^{59} 种。在如此庞大的可能性范围内，手动找出最佳的组合以优化文件大小将变得非常困难。因此，选择合适的混淆规则并进行组合使用是一项复杂的任务。

相关研究人员针对混淆工具及混淆规则的选择进行了深入研究。Peng 等人 [4] 提出了一种名为 ORChooser 的自适应推荐混淆规则的方法。该方法随机选择规则，并使用混淆距离进行计算，不断优化规则选择以最大化混淆距离，从而确保混淆效果，并在该方法上进行了在 R8 工具上的实验。You 等人 [5] 评估了 Android 平台的代码优化与混淆工具。R8 在精简代码和混淆中表现优异，可减少类、方法数量及 .dex 体积，并优化控制流。ReDex 削减 Obfuscapk 混淆后 APK 中 65% 的 CFG 冗余边，但效率低于 R8。Obfuscapk 使 APK_F-1 的 CFG 基本块数增 14 倍，但体积翻倍，需优化。DeGuard 对 R8 混淆符号恢复率达 77.8%，但解析自定义类名能力有限。工具间互补：R8 提升构建效率，Obfuscapk 增强逆向阻力，ReDex 和 DeGuard 支持逆向分析。但仅用工具的比较或单纯追求混淆效果，并不能将 Android 应用的体积降至最低。

为解决上述问题，本文提出了一种基于粒子群算法的安卓应用体积优化方法，旨在减小优化和混淆后的 Java 字节码文件或 Dalvik 字节码文件的体积。该方法的主要思想包括以下步骤：

（1）使用一种混淆器并随机选择需要使用的混淆规则。

（2）计算所生成可执行文件体积或者混淆后字节码文件体积的压缩比例。

（3）利用粒子群算法进行迭代并最大化压缩比例。这种方法不仅将混淆规则的选择转化为优化问题，还显著提升了寻找全局最优解的速度。通过将 Android 应用的 Java 字节码文件纳入相关测试，能够更准确地评估该方法在移动应用中的实际效果。

1. 南京审计大学计算机学院 江苏南京 210000

1 基于粒子群算法的安卓应用体积优化方法

本节介绍了一种基于粒子群算法的安卓应用体积优化方法，该方法采用迭代过程来选择或取消混淆规则。该方法的概述如图 1 所示，该方案主要分为 3 个步骤：

（1）选择混淆工具，例如 Android R8 或 Proguard，并从待选的混淆规则集合中，随机选择需要使用的混淆规则。

（2）通过两个生成的 Dalvik 可执行文件计算所生成文件体积的压缩比例，当达到最大迭代次数，进行下一步操作。

（3）利用改进后的粒子群算法进行迭代更新混淆规则，以选择出最优混淆规则组合。

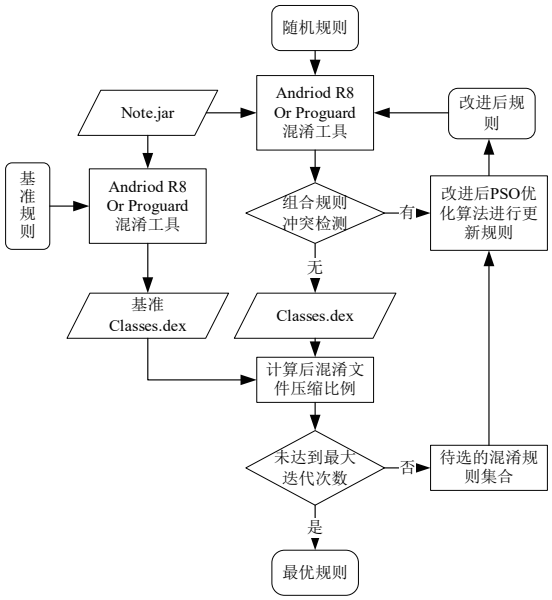

图 1 基于粒子群算法的安卓应用体积优化方法流程图

1.1 解决冲突方案

在混淆工具代码执行过程中，选择的规则组合可能会导致异常抛出，例如 RuntimeException、ClassCastException 等。为解决这些冲突，本文采用粒子群算法更新混淆规则的方法。如果未出现冲突，将保留当前的规则组合。如出现冲突，粒子群优化算法冲突规则列表搜索冲突组合，例如“-dontshrink”和“-repackageclasses”，将删除不适合优化代码大小的规则，然后进行迭代检测是否还有冲突。

1.2 评估方法的定义

评估方法主要用于评估计算混淆后文件压缩比例效果的好坏。同时，评估方法的作用是将混淆规则的复杂选择问题转化为压缩比例的优化问题，可以使用各种优化算法来搜索最佳解决方案。

为评估混淆规则的有效性，基准文件被保留为未进行代码压缩、优化和混淆的状态的源程序。其目的是与经过迭代优化后生成的字节码文件进行比较，以判断混淆规则是否对压缩产生有利影响。

定义 1：字节码混淆 $N \rightarrow N'$ 是混淆文件 N 转换为目标文件 N' 的过程，实际上保持了代码的语义一致性，只是使用更少的字符去进行重命名。这样做的目的是尽可能减少字节码文件的字符数量，同时确保程序仍然可以正常运行。

定义 2：两个字节码文件之间 N 和 N' 之间压缩比例，由于 R8 和 Proguard 输出文件格式并不相同，所以直接读取各自混淆工具所生成文件的体积，压缩比例为：

$$\text{Compression}_{\text{ratio}} = 1 - \frac{\text{Obfuscated file}_{\text{size}}}{\text{Baseline file}_{\text{size}}} \quad (1)$$

1.3 计算压缩比例

步骤 1：获取两个字节码文件的体积。如果使用 Proguard，则读取混淆后的 .jar 格式数据大小和准基的 .jar 格式数据大小。如果使用 Android R8，则读取混淆后的 .dex 格式数据大小和基准的 .dex 格式数据大小。

步骤 2：利用已获取的数据和压缩比例公式进行计算。在迭代过程中，将此计算结果作为评估混淆过程的指标。

1.4 粒子群优化算法规则推荐

传统的混淆规则选择方法通常基于经验去选择或者全部应用混淆规则，然后遇到冲突时进行调整。为解决这一问题，本文采用了粒子群算法对混淆规则进行迭代优化。该算法的基本思想是将规则的组合抽象为一个粒子，并通过随机生成一组粒子，不断进行迭代更新来寻求最优解。在算法中，通过目标函数确定个体和全局的最优值，更新粒子的速度和位置，使得个体和全局极值逐渐接近最优解，并最终达到最优状态。以下是速度和位置的更新为：

$$v_{jd}^{i+1} = wv_{jd}^{i} + c_1 r_1 (p_{jd}^{i} - x_{jd}^{i}) + c_2 r_2 (p_{gd}^{i} - x_{gd}^{i}) \quad (2)$$

$$x_{jd}^{i+1} = x_{jd}^{i} + v_{jd}^{i} \quad (3)$$

式中：w 为惯性权重；$j = 1, 2, \cdots, n$；$d = 1, 2, \cdots, D$；i 为迭代次数；c_1、c_2 为非负的加速度学习因子；r_1、r_2 为 [0,1] 上的随机；v_{jd}^{i}为当前迭代次数的粒子速度；p_{jd}^{i}为当前迭代次数的最优的粒子位置；p_{gd}^{i}为全局最优的粒子位置。

迭代过程如算法 1 所示。初始化，将粒子群的位置（即规则组合）和速度进行随机初始化。设置参数：惯性权重 w，学习因子 c_1、c_2 。然后，将每个粒子所代表的混淆规则组合应用于源程序 N 中，生成混淆程序 N'。

算法 1 粒子群算法对混淆规则的优化过程

输入：待选混淆规则集合，程序 N，R8

输出：最优混淆组合

1: 初始化粒子群

2: 设置 w、c_1、c_2

3: 对于每次迭代 i 从 1 到最大迭代次数执行

4: 对于粒子群中的每个粒子 j 从 1 到粒子群总数执行

5: N' = R8(N, 规则组合)

6: 分数 = Compression_ratio(N, N')

// 计算分数由式（1）可知

7: 更新全局最佳位置和分数

8: 结束粒子循环

9: 对于粒子群中的每个粒子 j 从 1 到粒子群总数执行

10: 更新粒子速度和位置

// 更新粒子速度和位置由式（2）可知

11: 结束粒子循环

12: 结束迭代循环

13: 输出最佳的混淆规则组合

接下来，计算每个粒子的适应度得分，即混淆程序的压缩比例。根据适应度更新全局最佳位置和得分。然后，更新每个粒子的速度和位置。该迭代更新的过程会重复进行，直到达到最大迭代次数。最终，输出具有最优位置（即最佳规则组合）的粒子。

2 实验结果与分析

本文验证了基于粒子群算法的安卓应用体积优化方法对输出文件体积的影响，使用基于粒子群算法的安卓应用体积优化方法在 Android R8（版本 8.1.72）和 Proguard（版本 7.4.2）上进行实验来评估其有效性。实验过程均在 Ubuntu20.04.1 x86_64 系统上使用 openjdk 11.0.22 和 Java HotSpot(TM) 64 bit ServerVM。

2.1 实验数据集与预处理

为了选择基准进行评估，首先选择了 SPECjvm2008[6]。SPECjvm2008 是一个用于评估 Java 虚拟机性能的基准测试套件。在 SPECjvm2008 基准测试套件中，主要选择测试程序：Compress、crypto.aes、crypto.rsa、crypto.signverify 以及 serial。

除了选择 SPECjvm2008 作为基准测试，本文还选取了小米文件管理器的社区开源版（MiCode 文件管理器）[7] 和小米录音机的社区开源版（MiCode 录音机）[8] 以及小米便签的社区开源版（MiCode 便签）[9] 作为基准应用。为避免反编译可能会造成一些内部错误，本文对该开源应用从源代码进行构建。基准源码编译打包之后的体积如表 1 所示。

表 1 基准包体积

基准包名	原始大小 /Byte)
Compress.jar	172 319
crypto_aes.jar	162 804
crypto_rsa.jar	162 474
crypto_signverify.jar	162 418
serial.jar	184 130
Fileexplorer.jar	394 982
Recorder.jar	39 148
Miui_note.jar	161 780

2.2 验证基于粒子群算法的安卓应用体积优化方法

本文使用了 4 个基准应用并基于粒子群算法的安卓应用体积优化方法。在评价方法的效果时，仍然采用文件的体积作为评价指标，以评估混淆工具在代码收缩、优化和混淆方面的效果。在混淆规则推荐算法方面，主要采用随机算法、贪婪算法、遗传算法和改进后的粒子群算法。

本文采用改进后的粒子群算法展开研究，由图 2 结果可知，粒子群算法可以在 3 轮迭代之后稳定达到最优点。相比之下，遗传算法需要超过 5 轮迭代才能使大部分个体接近最优解，而贪婪算法虽然收敛速度较快，但所得到的组合并非最优解。

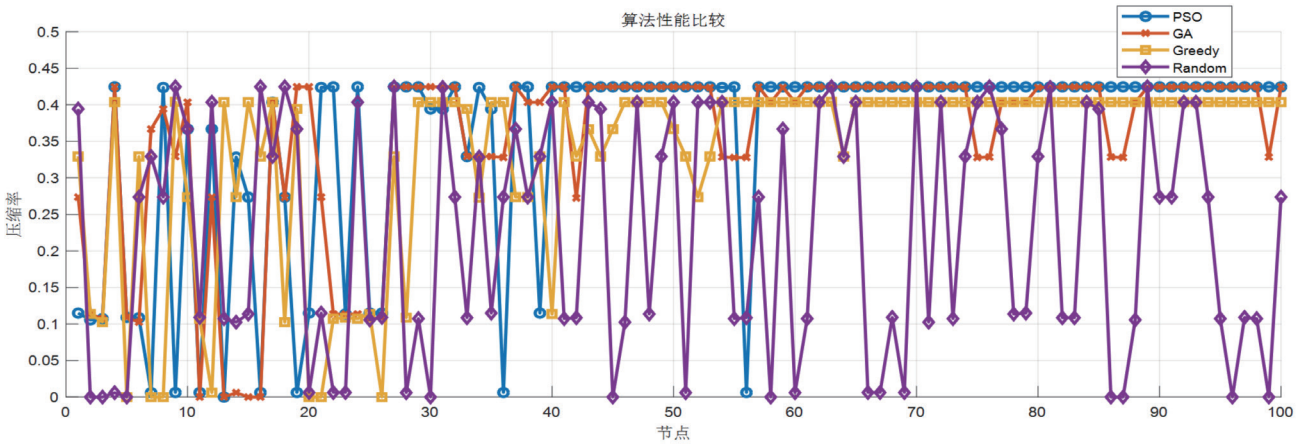

图 2 迭代算法性能比较

综上所述，经过改进后的粒子群算法在获取最优结果的速度和效果上表现最佳，能够快速地收敛并得到最优的混淆规则组合。

用获取最大压缩比例作为计算指标，不使用混淆规则的定义是在确保必要的输入输出和依赖规则的基础上，不使用其他任何混淆规则。用全部混淆规则是使用 Android 通用混淆规则文件基础上用全部核心混淆规则，例如“shrink”“optimize”“obfuscate”。混淆距离用于评估混淆后的结果是否达到充分混淆的标准。这些方法在表 2 和表 3 中都有体现。不使用混淆规则的目的保持代码原有的完整性。使用全部核心混淆的目的是最大程度地混淆代码，增加逆向工程的难度。从表 2、表 3 中可以看出，采用这种方法后，代码文件的字节大小显著减少。

表 2 应用体积优化方法在 Android R8 下的结果

测试包	不用规则	全部核心规则	混淆距离	压缩比例
Compress	176 168	105 012	118 088	101 352
crypto_aes	170 448	106 744	108 412	102 980
crypto_rsa	170 196	106 932	108 644	103 176
crypto_sig	170 364	106 760	108 424	103 000
serial	184 212	104 864	104 864	101 132
fileexplo	408 124	71 332	69 120	64 204
Recorder	54 872	37 712	38 504	31 964
Miui_note	203 936	133 276	131 528	105 292

表 3 应用体积优化方法在 Proguard 下的结果

测试包	不用规则	全部核心规则	混淆距离	压缩比例
Compress	171 018	78 111	79 245	70 555
crypto_aes	161 382	77 641	78 780	72 159
crypto_rsa	161 052	77 576	78 694	72 160
crypto_sig	160 984	77 601	75 423	72 186
serial	182 686	77 598	76 991	72 147
fileexplo	369 415	151 652	149 802	137 148
Recorder	36 310	30 246	32 771	26 944
Miui_note	149 807	119 111	116 231	106 517

综合分析表 2 和表 3 数据，可以得出以下结论：

首先，压缩比例最小化效果显著。使用压缩比例作为指标，可以有效地减少字节码文件的体积。例如，相比于不用规则，R8 可以将代码文件缩小到 39.4%~84.3%，而 Proguard 可以缩小到 25.8%~62.9%。

其次，规则选择的重要性。通过选择合适的混淆规则，可以显著减少字节码文件的体积。

最后，R8 与 Proguard 的比较。虽然 Proguard 在混淆 Java 应用时效果优于 R8，但在混淆安卓应用时，由于需要保留更多的程序入口点，其优化效果不如 R8 显著。这说明在选择混淆工具时，需根据具体的应用场景和需求进行权衡。

根据安卓默认打包 APK 时使用 zip 压缩格式对表 2 中的可执行文件进行压缩。其中，计算方式为：

$$\text{压缩比} = \frac{\text{不用混淆规则 dex 文件体积}}{\text{zip 格式压缩后文件体积}} \tag{4}$$

$$\text{差值百分比} = \frac{\text{全部核心混淆规则的压缩比} - \text{压缩比例的压缩比}}{\text{压缩比例的压缩比}} \tag{5}$$

由此可以得出结论：一是压缩比例处理方法在减少字节码大小方面效果最为显著。根据表 4 显示的结果，所有测试包在该方法下的字节码大小均为最小。这表明使用压缩比例处理方法安卓应用在压缩之后体积仍然是最小的。二是核心混淆规则对减小应用体积效果明显，使用全部核心混淆规则也显著减少了字节码大小，但其效果不如压缩比例处理方法显著。

表 4 zip 压缩格式下的各个测试包压缩比

测试包	不用规则	全部核心规则	压缩比例	差值百分比
Compress	2.482 46	3.881 29	3.957 76	0.019 70
crypto_aes	2.496 86	3.682 89	3.760 91	0.021 18
crypto_rsa	2.495 18	3.676 57	3.754 60	0.021 22
crypto_sig	2.497 71	3.692 88	3.773 46	0.021 82
serial	2.493 53	4.062 72	4.152 29	0.022 05
fileexplo	2.486 46	11.687 93	12.386 9	0.059 81
Recorder	1.488 63	2.181 19	2.378 81	0.090 60
Miui_note	2.295 83	3.100 32	4.029 16	0.299 60

3 结语

本文提出了一种基于粒子群算法的安卓应用体积优化方法，该方法专注于为 Java 字节码提供最合适的混淆规则组合。为了实现这一方法，在安卓应用程序中进行了实验，并在 R8 混淆器和 Proguard 混淆器中进行了实现。实验结果表明，通过选择合适的混淆规则能够有效减小字节码文件的体积，并且该方法在 zip 压缩后仍能保持压缩最大化。然而，该方法还有一些局限性，如该方法目前只兼容 Android R8 和 Proguard 混淆工具，对于其他混淆工具尚不适用。下一步工作将关注新兴混淆技术和工具的发展，以保持本方法的先进性和实用性。

参考文献：

[1] LI T, FAN Y L, LI Y, et al. Understanding the long-term evolution of mobile App usage[J]. IEEE transactions on mobile computing, 2021, 22(2): 1213-1230.

[2] BOJJAGANI S, SASTRY V N, CHEN C M, et al. Systematic survey of mobile payments, protocols, and security infrastructure[J]. Journal of ambient intelligence and humanized computing, 2023, 14: 609-654.

基于 LED 可见光的通信系统设计

綦振禄 [1]
QI Zhenlu

摘 要 可见光通信是利用可见光作为传输介质的一种新兴通信方式，LED 可见光通信利用 LED 照明设备发出的用肉眼观察不到的高速调制光波信号，对信息进行调制和传输。文章探讨了一种利用可调稳压器集成电路，采用幅度调制（AM）的电路设计思路，用单音信号直接调制 LED 光信号强度，实现了一种精简的基于 LED 可见光的通信系统设计，在传输效能、复杂度等方面具有优势。

关键词 可见光通信；三端可调稳压器；幅度调制；数字调制；发光二极管

doi：10.3969/j.issn.1672-9528.2025.04.020

0 引言

可见光通信（visible light communication, VLC）是一种利用可见光波段（波长介于 380~780 ns 之间）作为传输媒介的创新通信技术 [1]。这种通信方式常被称作“Lifi”，预示着其可能对现有的无线网络传输技术，特别是 Wi-Fi，产生深远的影响 [2-3]。与传统无线射频通信相比，VLC 使用未被传统无线电通信占用的频谱资源，从而避免了获取频谱许可的需求。此外，VLC 系统具备多项优势，包括强大的抗干扰能力、宽广的光谱频段、高通信速率、无电磁干扰以及优良的安全性和保密性。同时相较于无线红外和紫外通信系统，VLC 在信号功率方面更高，且不存在健康安全风险 [4]。2013 年，上海复旦大学的研究人员成功开发出一项国际前沿技术，该技术能够使用室内可见光传输网络信号，并且达到了 3.5 Gbit/s 的数据传输速率 [5]。

作为第四代照明光源的 LED，凭借其高亮度、低能耗、长寿命、小型化以及环保特性，在全球照明市场的比重逐年增长。LED 的高亮度和低能耗使其成为室内可见光通信（VLC）系统的理想光源，逐渐受到广泛关注。基于 LED 的 VLC 系统通过高速调制 LED 光源发出的可见光信号来传输信息，已经成为当前研究的热点领域之一 [6]。

本文提出了一种基于常规三端稳压器电路的 LED 可见光通信系统设计方案，采用幅度调制（AM）方法，直

1. 公安部第一研究所 北京 100048

[3] LI T, XIA T, WANG H D, et al. Smartphone App usage analysis: datasets, methods, and applications[J].IEEE communications surveys & tutorials, 2022, 24(2): 937-966.

[4] PENG Y R, CHEN Y T, SHEN B J. An adaptive approach to recommending obfuscation rules for Java bytecode obfuscators[C]//2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). Piscataway: IEEE, 2019, 1: 97-106.

[5] YOU G, KIM G, CHO S J, et al. A comparative study on optimization, obfuscation, and deobfuscation tools in android[J].Journal of internet services and information security, 2021, 11(1): 2-15.

[6] LENGAUER P, BITTO V, MOSSENBOCK H, et al. A comprehensive java benchmark study on memory and garbage collection behavior of dacapo, dacapo scala, and SPECjvm2008[C]//Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering. NewYork: ACM, 2017: 3-14.

[7] MIUI Team. MIUI file explorer[EB/OL]. [2024-05-10]. https://github.com /MiCode/FileExplorer.

[8] MIUI Team. MIUI sound recorder[EB/OL]. [2024-05-10]. https://github. com /MiCode /SoundRecorder.

[9] MIUI Team. MIUI notes[EB/OL]. [2024-05-10].https://github.com /MiCode /Notes.

【作者简介】

周少杰（1998—），男，浙江绍兴人，硕士，研究方向：大数据审计和区块链审计。

（收稿日期：2024-11-26）