计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200400027

软件升级问题的多目标优化方法

赵松辉　任志磊　江　贺
大连理工大学软件学院　辽宁 大连 116600
(adison@mail.dlut.edu.cn)

摘　要　近年来,开源软件包管理成为软件产品重用的一种普遍方式,尤其是在 Linux 发行版操作系统领域。其中,软件升级问题是软件包管理工具必须要解决的关键挑战之一。软件升级问题旨在按照某种优化准则找出能够满足用户升级请求的最合适的升级方案。优化准则由几个不同方向的优化目标组成,因此软件升级问题本质上是一个多目标优化问题。现有的解决软件升级问题的方法均是将多个优化目标聚合成为单个目标的形式再进行处理。这些方法都可能没有恰当地考虑不同的优化目标之间的关系,因此会存在潜在的风险。针对这种风险,文中提出了一个多目标演化框架——SATMOEA(Combining Constraints Solving and Multi-objective Evolutionary Algorithms),将软件升级问题构建为可满足问题+多目标优化问题的形式,并集成了约束求解和多目标优化算法,来对软件升级问题进行求解。基于 MISC 竞赛提供的升级问题标准实例集进行实验,结果表明对于有着大量约束条件的复杂问题实例,多目标演化框架在一次运行中即可有效地计算出各个优化目标均达到帕累托最优的解决方案,相比现有的升级问题求解器提供的升级方案更加多样,并且在一些优化目标上更具优势,可以满足用户在不同场景下的需求。

关键词:软件升级问题;多目标优化;SAT 求解;基于搜索的软件工程;软件仓库
中图法分类号　TP311

Multi-objective Optimization Methods for Software Upgradeability Problem

ZHAO Song-hui, REN Zhi-lei and JIANG He
School of Software, Dalian University of Technology, Dalian, Liaoning 116600, China

Abstract　Open-source Package management as a means of reuse of software artifacts has become extremely popular, most notably in Linux distributions. Software upgradeabilty problem is a significant challenge which package management system must resolve. This problem aims to find the most suitable upgrade scheme that satisfies upgrade requests from users. An upgrade scheme comprises of a sequence of operations, including installing, removing, and/or upgrading packages. In the existing approaches for solving this problem, multiple upgrade requests are handled in aggregate ways. Hence, a potential risk of such approaches is that, the relationships between different upgrade objectives may not be considered properly. This paper introduces a novel approach SATMOEA, which forms software upgradeability problem as a SAT plus multi-objective optimization problem and addresses this problem combining constraint solving and multi-objective search-based optimization algorithms. We evaluate it on real instances provided by MISC (Mancoosi International Solver Competitions) and obtain promising results where we can find some Pareto optimal solutions for a complex instance with myriad constraints in a single run. In comparison with other solvers, it can provide more solutions with better diversity property to satisfy requirements in different scenarios.

Keywords　Software upgradeability problem, Multi-objective optimization, SAT solving, Search-based software engineering, Software repository

1　引言

随着开源思想的流行,越来越多的开发者加入开源软件世界,FOSS(Free and Open Source Software)发行版操作系统中的开源软件仓库的规模日益壮大。同时,开源软件仓库中的软件包之间的依赖和冲突关系越来越复杂,这导致了一个严重的问题:当使用系统原生的包管理器安装或升级软件包时,可能会遇到软件包安装或升级后系统变得不稳定或丧失一些功能的情况,甚至可能由于计算上的高复杂度,包管理器无法找到能够满足系统中软件包之间的依赖和冲突的安装方案,而直接提示软件包安装或升级失败。这些状况会影响正常的生产工作,因此有效地解决软件安装或升级问题十分必要。如图 1 所示,自 2001 年以来,Debian 软件仓库中的软件包数量持续增长,截至 2019 年 12 月,软件包数量已超过 59000[1]。然而,由于软件包之间的联系,软件包的管理在计算上是非常复杂的。例如,一个软件包可能依赖某些软件包

到稿日期:2020-03-07　返修日期:2020-05-11　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
通信作者:任志磊(zren@dlut.edu.cn)

或者与某些软件包冲突，软件包管理工具必须要能够维护好一个描述软件包安装情况的配置方案，该方案须满足每个软件包的依赖项，并且不会造成软件包之间的冲突，搜索这样的配置方案就是软件升级问题。

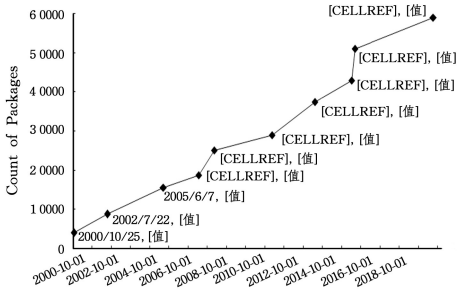


图 1 Debian 软件仓库中的软件包数量的增长趋势

Fig. 1 Growth trend of packages in Debian

然而，配置方案仅仅满足软件包之间的依赖和冲突是不够的，还需要在这个基础上达到某种最优的要求，软件包升级问题本质上是多目标优化问题。考虑到系统的稳定性，用户可能对不同的升级目标感兴趣，如要求被移除的软件包的数量和版本发生改变的软件包的数量尽可能少；此外，考虑到网络传输数据量，用户可能希望需要下载的软件包的体积尽量小等。即使仅考虑其中一个升级目标，软件升级问题可以简化为带权重的最大可满足(partial weighted MAXSAT)问题[2]，这也是一个 NP 难问题。不仅如此，越来越庞大的软件包仓库也使得可量测性成为软件升级过程中的巨大挑战。现有的升级问题求解器都是以聚合的方式简单地将多个升级目标加权转化为一个单目标优化问题，再使用单目标优化算法进行求解。这种方式未考虑不同目标之间的关系和冲突，可能存在一定的风险：如果过度重视某一个目标的优化，那么单目标优化算法在其他目标上的表现就会非常糟糕[3]。

针对上述问题和挑战，本文首次尝试结合约束求解和多目标演化算法来处理软件升级问题，并提出了一个多目标优化软件升级框架——SATMOEA，其能够同时对多个升级目标进行优化，而不会发生现有的升级问题求解器中“厚此薄彼”的现象。该框架分为实例解析、约束求解、构建多目标优化问题、演化优化 4 个阶段。实例解析是将软件升级问题实例(CUDF 文件)中描述的与用户请求的软件包相关的依赖和冲突关系抽象出来，表示为描述布尔可满足性问题(SAT)的 CNF 文件；约束求解是使用快速采样技术得到 CNF 文件的一系列可行解；构建多目标优化问题是定义软件升级问题的各个优化目标函数以及约束条件的计算方法；演化优化阶段则是使用本文改进的多目标演化算法对构建好的多目标优化问题进行迭代优化求解，并最终得到帕累托解集。

本文在 MISC 提供的升级问题标准实例集上进行了实验，结果表明该框架能够在一次运行后即可得到一系列在各个优化目标上均为帕累托最优的升级方案，相比现有的软件升级问题求解器一次只能得到一个升级方案，该框架能够提供给用户更多的可选方案以应对多样的应用场景，并且在一些升级目标上有显著的优势。

本文第 2 节论述了软件升级问题领域现有的相关工作和研究进展；第 3 节详细阐述了本文提出的多目标优化软件升级框架的流程与细节；第 4 节介绍了本框架在标准升级问题实例集上的实验及结果；最后总结了本框架的优势与存在的问题，以及未来进一步的研究方向。

2 相关工作

2.1 软件升级问题求解器

在类 UNIX 操作系统中有各种用于解决软件包管理问题的自动化工具，如 Debian 中的 apt-get、RedHat 中的 yum、macOS 中的 fink 等。这些工具基于软件包的依赖和冲突信息来决定如何在已安装其他软件包的系统中安装用户请求安装的新包，并且满足其依赖[4]。然而，由于软件包之间依赖和冲突关系的复杂性，这些工具通常使用某种启发式方法，因此是不完备的求解工具。在有些情况下，尽管某个软件包是可以安装的，但是这些工具却不能成功地找到一个安装方案[5]。

早期有些工作采用了基于布尔可满足性问题(Boolean Satisfiability，SAT)的工具来求解复杂的依赖和冲突，并能确保在安装及升级软件包前后，软件包仓库和系统原已安装的软件包的一致性。Mancinelli 等[6]将软件包安装问题形式化描述为 SAT 问题，并首次使用基于 SAT 的工具为发行编辑提供支持。他们开发的自动化工具能够确保完整性，并且比内置的和手动的工具更加可靠。后来，Argelich 等[7]应用最大可满足问题(Maximal Satisfiability，MaxSAT)的思想从用户角度求解软件包安装问题。Tucker 等[4]提出了优化的方法，通过使用 SAT 求解器、伪布尔问题求解器、整数线性规划求解器来达到更好的结果。他们设计出 Opium 工具用于优化用户提供的一个目标函数。Argelich 等[8]从布尔多级优化(Boolean Multi-level Optimization，BMO)的视角求解软件升级问题，他们提出了两种不同的算法，分别基于 MaxSAT 和伪布尔优化(Pseudo-Boolean Optimization，PBO)。他们的实验表明，专门用于 BMO 的算法比最好的 MaxSAT 和 PB 求解器效率提高了几个数量级。Daniel 等[9]开发了一个基于 SAT4J 伪布尔求解器的软件包升级工具 P2，用于处理 Eclipse 平台上的插件安装问题。Paulo 等[10]将软件安装问题形式化描述为伪布尔优化问题再进行求解。至止，以上求解工具都是采用在不同的软件包管理环境中的不同语言来描述软件升级问题。

为了便于评估和比较这些求解工具的性能，MANCOOSI 项目人员设计了一种标准的文档格式 CUDF(Common Upgradability Description Format)来描述软件升级问题[11]。自此，各种 CUDF 求解器开始出现，它们将 CUDF 文件描述的升级问题转换为 SAT 问题、混合整数线性规划 (Mixed Integer Linear Programming，MILP)问题、伪布尔优化(PBO)问题、ASP(Answer Set Programming)问题、最大可满足问题(MaxSAT)等形式，再采用相应的专门算法进行求解。这些求解器包括 mccs[12]，aspcud[13]，P2CUDF(P2 的新版本)[14]，PackUp[15]和 PackUpHyb[3]。本文使用 CUDF 实例进行实验探究。

上述现有的求解器大多是将软件升级问题按照用户的个性化请求转换为某个单目标优化问题，然后以聚合的方式处

理多个升级目标。尽管这些求解器能够得到一个某种意义上的最优解，但是这些方法具有明显的局限性：在现有的方法中，多个升级请求以聚合的方式被处理，如加权和(weighted sum)标量化转换方法和字典序组合方法。因此，这些方法存在一个潜在的风险：不同的升级目标之间的关系可能没有被恰当地考虑。

2.2 基于搜索的技术在软件工程中的应用

软件升级问题在于找到能够满足软件包的依赖关系并避免冲突的软件包安装方案，这是一个 NP 难问题。使用精确算法求解 NP 难问题需要指数级别的复杂度，但是每一个 NP 难问题都可以用一些近似算法来获得可接受的优化解。基于搜索的软件工程倡导应用运筹学领域的优化技术或(元)启发式计算方法来解决软件工程问题[16]，这些技术或方法按照构建的问题模型主要分为单目标优化方法和多目标优化方法。对于本文涉及的多目标优化领域，目前已有许多成功应用的实例，如优化需求搜索以求解 NRP(Next Release Problem)[17]问题、优化测试数据选择和优先级排序[18]、优化软件项目管理[19]，以及用于优化软件产品线中特征选择问题的 SATIBEA 框架[20]。此外，基于搜索的技术在软件测试数据自动生成[24-25]、程序错误自动修复[26]、软件缺陷定位[27]、软件产品线配置[28-29,32]、云计算构件优化[30-31]等软件工程领域也取得了丰硕的成果。

本文从软件升级问题的自然本质出发，此问题实质上是多目标优化问题。我们首次采用多目标演化算法来进行求解，同时优化多个升级目标，而不是以聚合的方式处理这些目标，最后得到一组帕累托最优解提供给用户或服务器管理员，以供他们在不同场景下进行选择。

3 多目标优化软件升级框架

针对软件升级问题，本文提出一个多目标优化框架 SATMOEA，该框架结合了可满足问题(SAT)的求解和多目标演化算法(MOEA)。框架的结构设计如图 2 所示。

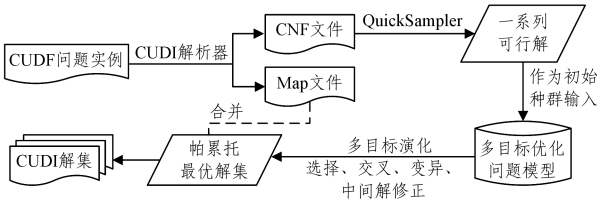


图 2 软件升级问题的多目标优化框架

Fig. 2 Overview of SATMOEA framework

本框架求解一个软件升级问题实例的流程如下：

(1)使用 CUDF 解析器(Parser)，将输入的描述一个升级问题实例的 CUDF 文件解析为相应的描述可满足问题的 CNF 文件，即把 CUDF 实例转为 CNF 实例；同时，生成一个映射文件(Map)，描述 CNF 中的数值编号与 CUDF 中的软件包之间一对一的映射关系。

(2)对于步骤(1)中得到的 CNF 实例，使用 MaxSAT 快速采样技术高效地得到一定数量的解，即为升级问题的一部分可行解，然后选取这些解中的部分或全部作为演化算法的初始种群。QuickSampler 采样工具实现了 MAX-SAT 快速采样技术，通过较少次数地调用 MaxSAT 求解器就能得到大量 CNF 实例的有效解。

(3)建立有约束的多目标最小化问题。其中，约束条件为问题解必须为 CNF 的可行解。目标有 5 个，分别为：1)需要移除的软件包的数量；2)版本发生变化的软件包的数量；3)新增的软件包的数量；4)非最新版本的软件包的数量；5)未满足的推荐软件包的数量。

(4)选择演化算法并配置演化相关参数(种群大小、演化代数、变异策略及概率、交叉策略及概率、选择策略)，然后执行演化算法，得到帕累托最优解集(Pareto Front)。

3.1 软件升级问题的形式化构建

本文将软件升级问题构建为“可满足问题＋多目标优化问题”的形式，以便使用约束求解算法和多目标演化算法逐步求解软件升级问题。

3.1.1 构建可满足问题

一个典型的 CUDF 实例文件的结构如图 3 所示，它能够描述软件仓库中所有软件包的信息(Universe)、当前系统中已安装的软件包的信息(Installed)、用户的请求(Request)，以及软件包之间错综复杂的依赖冲突关系[11]。每个软件包的信息由多个字段描述，主要包括软件包名字、版本号、是否已安装、依赖项、冲突项等。关于依赖项和冲突项的说明如下：假设软件包 A 的依赖项中有软件包 B，那么，当安装软件包 A 时，必须先安装软件包 B；当卸载软件包 B 时，软件包 A 也将随之被卸载。假设软件包 A 的冲突项中有软件包 B，那么，A 和 B 不能同时存在于同一个系统中，即当安装软件包 A 时，须保证软件包 B 未被安装或已被卸载。

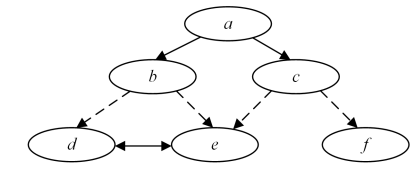
图 3 CUDF 文件的结构

Fig. 3 Structure of CUDF file

求解软件升级问题的先决条件是：必须首先解决软件包之间的冲突和依赖问题，在此基础上才能进一步搜索最优的升级方案，即一个软件升级方案一定是满足系统中所有软件包的冲突和依赖的升级方案。基于 CUDF 的文件结构，本文设计了专门的语法解析器，用于解析 CUDF 文件，获取其中蕴含的软件包之间的依赖与冲突关系信息；然后将冲突和依赖关系形式化为布尔可满足性问题，也就是 SAT 问题。因此，CUDF 解析器的作用是将 CUDF 文件转化为描述相应的 SAT 问题的 CNF 文件。SAT 问题用于判断是否存在一组变量赋值满足给定的布尔表达式，它的定义如下：一个布尔表达式由变量、合取操作符(∧)、析取操作符(∨)、否操作符(¬)和括号组成，当存在一种对变量的赋值方案，使得布尔表达式的值为真时，称这个布尔表达式可满足。

考虑图 4 所示的升级问题实例，假设从一个 CUDF 实例中获取到的软件包之间的关系如图 4(a)所示，用户的请求是安装软件包 a。图 4 中，实线单箭头表示“与”依赖关系，在

图 4 表示的关系中，软件包 b 和 c 均已安装的情况下，软件包 a 才能被安装；虚线单箭头表示“或”依赖关系，即软件包 d 和 e 中安装任意一个之后，才能安装软件包 b，软件包 c，e，f 同理；实线双箭头表示冲突关系，即软件包 d 和 e 不能同时安装在一个系统中，只能安装其中一个。



(a)Sample

```
1 -> [a]
2 -> [b]
3 -> [c]
4 -> [d]
5 -> [e]
6 -> [f]
```

(b)Sample. map

```
p cnf 6 6
 1  0
-1 2 0
-1 3 0
-2 4 5 0
-4 -5 0
-3 5 6 0
```

(c)Sample. cnf

图 4 一个问题实例及其对应的 Map 文件和 CNF 文件

Fig. 4 CUDF instance and its corresponding Map and CNF

可以用式(1)所示的布尔表达式描述软件包 a，b，c 之间的关系：

$$a \wedge (\neg a \vee b) \wedge (\neg a \vee c) \tag{1}$$

这是一个合取范式，要使合取范式的取值为真，必须使每一个子句的取值为真。当某个变量值为 true 时，表示安装对应的软件包；否则表示不安装。式(1)中，第 1 个子句 a 表示用户请求安装软件包 a，即 a 的取值必须为 true；第 2 个子句 $\neg a \vee b$ 表示当 a 的取值为 true 时，b 必须为 true，才能使子句的取值为真，也就是软件包 a 依赖软件包 b 的关系；第 3 个子句同理。

式(2)描述了软件包 b 依赖软件包 d 或 e 的关系以及软件包 d 与 e 之间的冲突关系：

$$(\neg b \vee d \vee e) \wedge (\neg d \vee \neg e) \tag{2}$$

在第 1 个子句中，当 b 为 true 时，须使 d 或 e 中至少一个为 true，才能使子句为真，这就表示了软件包 b 与软件包 d 和 e 之间的“或”依赖关系。与之同理，可以用式(3)来描述软件包 c，e，f 之间的关系。在第 2 个子句中，当变量 d 和 e 之间任意一个取值为 true 时，另一个变量必须为 false，才能使子句为真，即两个变量不能同时为 true。

$$(\neg c \vee e \vee f) \tag{3}$$

使用合取符号将式(1)—式(3)连接起来可得到式(4)，这样就完整地表达了图 4 实例中的依赖和冲突关系。

$$a \wedge (\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee d \vee e) \wedge (\neg d \vee \neg e) \wedge (\neg c \vee e \vee f) \tag{4}$$

为了数值化表示这些关系，CUDF 解析器首先对所有涉及的软件包进行编号，编号从 1 开始，即生成软件包与数值编号的一一映射关系，如图 4(b)所示的 Map 文件(Sample. map)。

图 4(c)所示的 CNF 文件(Sample. cnf)是由式(4)直接转化得到的。其中，第 1 行“p cnf 6 6”是问题描述行，表示这是一个 CNF 文件，并且包含 6 个变量和 6 个子句。从第 2 行开始的每一行都对应式(4)中的一个子句，转化的规则是：使用 Map 文件定义的软件包的编号替换公式中的字母变量；使用空格替换析取符号(∨)；最后再加上空格和数字 0 作为结尾，数字 0 表示子句结束符。

CNF 文件所描述的 SAT 问题实例可以通过 SAT 求解器进行求解。当构建的 SAT 问题在有限时间内找不到可行的赋值方案时，求解器会将其标记为“UNSATIFIABLE”，即该 CUDF 实例所描述的用户请求很难被满足；当 SAT 求解器能够找到一个可行的赋值方案时，会将问题标记为“SATISFIABLE”，并提供一个可行的赋值方案。赋值方案规定了每一个软件包是否被安装，当编号为正数时，表示安装对应的软件包，当编号前面加上了负号变成负数时，表示不安装对应的软件包。另外，赋值方案以“v”为起始标志，以“0”为结束标志。例如，对于图 4 所示的问题实例，SAT 求解器的输出文件如图 5 所示。根据它提供的赋值方案并对照图 4(b)的 Map 文件，即可得出：需要安装的软件包有 a，b，c，d，f，不需要安装的软件包是 e。

```
s SATISFIABLE
v 1 2 3 4 -5 6 0
```

Sample.cnf.out

图 5 SAT 求解器的输出文件

Fig. 5 Output file of SAT solver

传统的 SAT 求解器最多只能提供一个可行解，直到 2018 年 Dutra 等[21]提出了一个高效的 SAT 问题求解工具 QuickSampler，它一次能够得到大量的可行解。这也是我们在本工作采用多目标演化算法搜索最优的软件升级方案的动机之一，使用 QuickSampler 工具一次性得到 CNF 实例的大量可行解恰好可以作为演化算法的初始种群。

3.1.2 构建多目标优化问题

对于多目标优化问题的构建，本文选择 MISC 提出的 5 个优化准则作为最小化目标，分别是：1)需移除的软件包数量(Remove)，对应目标函数为 $R(X)$；2)版本发生改变的软件包数量(Change)，对应目标函数为 $C(X)$；3)新增的软件包数量(New)，对应目标函数为 $N(X)$；4)不是位于最新版本的软件包数量(Notdate)，对应目标函数为 $Nd(X)$；5)未满足推荐软件包的数量(Unsatisfied-recommend)，对应目标函数为 $Us(X)$。本文将约束条件设置为：必须满足 CNF 实例中的每一个子句，即赋值方案须保证真值为假的析取范式的数量为 0。我们用长度为 n(表示 CNF 实例涉及到的软件包数量)的二进制串来表示一个解，第 i 位上的取值即表示编号为 i 的软件包是否需要安装：0 代表不需安装，1 代表需要安装。例如，图 5 所示的一个可行解可以转化为二进制串“111101”。

在演化算法中，优化函数的每一个解 X 表示一个个体，若干个体组成一个种群，我们称之为代；选择种群中的部分个体，让它们随机配对交叉或随机变异，产生新的个体，即构成新一代种群，这就是生物进化的思想。我们用 X_k^p 表示演化第 p 代中的第 k 个解，它是一个特殊的 n 维向量，每一位只能取值 1 或 0，表示某个软件包是否需要安装，我们称之为二进制向量。该解对应的上述 5 个目标函数值分别定义为：1)$R(X_k^p)$，表示解决方案 X_k^p 导致的原来系统中需要移除的软

件包的数量；2）$C(X_k^p)$，表示解决方案 X_k^p 使原来系统发生版本变更的软件包的数量；3）$N(X_k^p)$，表示解决方案 X_k^p 使原来系统新增软件包的数量；4）$Nd(X_k^p)$，表示解决方案 X_k^p 中不是位于最新版本的软件包的数量；5）$Us(X_k^p)$，表示解决方案 X_k^p 中软件包的推荐项未被满足的数量。

这些函数值可以根据 X_k^p 定义的软件包的安装方案与 CUDF 实例中描述的软件包的安装情况以及软件包的版本信息、推荐安装列表对比计算得出。另外，我们用 Violate(X_k^p) 表示违背子句的数量，即 X_k^p 定义的赋值方案使得 CNF 实例中结果为假的子句（析取范式）的数量。例如，对于图 4 所示的实例来说，当 X_k^p=(1,0,1,0,1,0)时，它对应的 CNF 赋值方案是“1 −2 3 −4 5 −6”，CNF 中的违背子句有且仅有“−1 2 0”，原因是“−1 2 0”规定的是赋值方案中须包含−1 或 2，而 X_k^p 对应的赋值方案中既不包含−1，也不包含 2。此时，$Violate(X_k^p)=1$。

综上所述，软件升级问题可以构建为如下形式的包含 5 个优化目标和 1 个约束条件的多目标最小化问题。

$$\min F(X) = (R(X), C(X), N(X), Nd(X), Us(X))^{\mathrm{T}}$$

s. t. $Violate(X)=0$，X 是一个二进制向量

3.2 演化及中间解修正算法

当多目标优化问题模型构建完成后，我们使用多目标演化算法进行求解。演化算法的输入是初始种群，因此首先需要确定初始种群。初始种群可以完全随机化，即随机产生若干个体构成初始种群，这些个体的每一位都是随机的 0 或 1，不能保证满足多目标优化问题的约束条件；借助 QuickSampler 工具，能够一次性得到 CNF 实例的若干个可行解，于是我们还可以将这些可行解全部作为初始种群中的个体。

确定初始种群之后，正式进入演化过程。与生物进化论相似，选择、交叉、变异是演化算法中的 3 种常用操作。选择是指从种群中选出适应度较好的个体，用于进行交叉操作从而产生新一代个体，不同的选择算法计算适应度的方法有所不同。交叉是指将两个父代个体的基因局部进行交换，产生两个新的子代个体，在本问题中，我们用二进制向量表示问题的解就可以很方便地进行单点或多点交叉操作。个体的基因有一定的概率发生变异，我们采用位翻转的变异策略，当问题的解的某一位从 0 变为 1 或相反时，表示发生了变异。

如果父代个体是可行解，当它们完成交叉和变异操作（交叉操作对于解的改变是非常显著的）之后，产生的新解极有可能变成不可行解，而不可行解违背了优化问题的约束条件。因此，将产生的不可行解调整为可行解，是十分必要的。我们在多目标演化算法的基础上，增加了中间解修正操作，以维持解的可行性。

中间解修正算法如算法 1 所示。该算法的基本思想是：先找出不包含在被违背的约束条件中的软件包，固定它们的取值（0 或 1），具体来讲就是将它们的取值作为约束条件，添加到原来的 CNF 约束条件之后；然后由 SAT 求解器计算出其余的软件包的取值。下面考虑这样的一个 CNF 实例：

$$(p_1 \vee p_5) \wedge (p_2 \vee p_3) \wedge (p_2 \vee p_5)$$

当我们得到了它的一个不可行解 M 的赋值方案（0,0,1,1,0）时可知，该方案违背了 CNF 定义的两个约束条件（$p_1 \vee p_5$）和（$p_2 \vee p_5$），这两个约束条件涉及 p_1，p_2，p_5 3 个软件包，我们将这 3 个软件包的取值从 C 中移除，将 p_3 和 p_4 的取值作为约束条件添加到原来的 CNF 实例中，则新的 CNF 变为：

$$(p_1 \vee p_5) \wedge (p_2 \vee p_3) \wedge (p_2 \vee p_5) \wedge p_3 \wedge p_4$$

我们将新的 CNF 交给 SAT 求解器进行求解，计算出 p_1，p_2，p_5 的取值，得到一个新的可行解 N=(1,1,1,1,0)。于是，一个中间解的修正操作就完成了。

算法 1 中间解修正算法

输入：一个演化过程的中间解 M=(x_1, x_2, …, x_m)，满足 Violate(M)>0；CNF 文件 F_{old} 中的子句集合 C=(c_1, c_2, …, c_n)

输出：修正后的可行解 N

```
1. 令集合 B 表示中间解中需要修正的索引集合；
2. B←Ø；
3. for i←1 to n do
4.    if c_i=false then
5.       将 c_i 包含的索引添加到集合 B 中；
6.    end if
7. end for
8. for i←1 to m do
9.    if 集合 B 中不包含 x_i then
10.      将子句“x_i 0”添加到集合 C 中；
11.   end if
12. end for
13. 使用集合 C 中的子句构造出一个新的 CNF 文件 F_new；
14. 调用 SAT 求解模块对 F_new 进行求解，得到可行解 N；
15. return N.
```

4 实验与分析

4.1 实验设置

本文选择国际软件包安装与升级问题求解器竞赛[22]提供的 CUDF 标准实例数据集进行实验，其中 CUDF 实例模拟了真实系统环境中软件包的复杂度。实验按照软件包名称计数，这些实例包含的软件包全集的数量为 27 710～59 094，平均数量为 35 275.5，与 Debian 实际的软件包仓库的规模十分接近。因此，本文获得的实验结果可以模拟现实环境中求解器的性能表现。

在多目标演化模块中，本文在开源的多目标优化算法框架 jMetal[23]提供的基于指示器的演化算法（Indicator-Based Evolutionary Algorithm，IBEA）的基础上，调整了交叉算子和变异算子的实现方式以适配软件升级问题，并且增加了演化中间解的修正操作。演化算法中的一些重要参数设置如表 1 所列。

表 1 演化算法中的参数设置

Table 1 Parameter values used in IBEA

Name	Value
Selection Operator	Binary Tournament
Crossover Operator	Single-Point Crossover
Mutation Operator	Bit-Flip Mutation
Crossover Probability	0.9
Mutation Probability	0.05

4.2 实验结果与分析

本文主要对以下 4 个研究问题进行实验分析。

RQ1：提高初始种群中可行解的比例对演化算法结果的提升效果如何？

RQ2：相比其他高维多目标演化算法，IBEA 算法是否具有优势？

RQ3：中间解修正会对演化结果造成怎样的影响？

RQ4：SATMOEA 的性能是否优于其他软件升级问题求解器？

4.2.1 探究约束求解对演化结果的提升效果

对于初始种群，有两种构建策略：1）使用 SAT 求解器对 CNF 实例进行求解，得到一个可行解作为初始种群中的一个个体，其余个体随机生成（必定包含大量的不可行解），构成完整的初始种群；2）使用 QuickSampler 工具，它能快速得到指定数量的可行解，然后使用这些可行解建立初始种群，即初始种群全部为可行解。

我们使用以上两种方法构建初始种群，分别执行基于指示器的演化算法：初始种群的大小设为 30，迭代 4 次，统计演化算法终止时得到的帕累托解集中可行解的情况，如表 2 所列。实验结果表明，初始种群中可行解的数量越多，演化得到的帕累托解集中的可行解的数量就越多。因此，最好是将初始种群全部设为可行解（可以借助 QuickSampler 工具实现），以便在最后的帕累托解集中得到更多的可行解。

表 2 比较两种初始种群构建策略得到的帕累托解集中可行解的情况

Table 2 Comparison of two different methods to build initial population

项目	策略 1	策略 2
初始种群的组成	1 个可行解，29 个随机解	30 个可行解
帕累托解集的大小	22	15
帕累托解集中可行解的数量	1	3
帕累托解集中可行解的比例/%	4.55	20

4.2.2 不同多目标演化算法的对比

Sayyad 等[32]在软件产品线工程中的特征选择问题的研究中发现，相比其他多目标演化算法（Multi-Objective Evolutionary Algorithm，MOEA），基于指示器的演化算法（IBEA）表现更好。为了验证 IBEA 在软件升级问题上是否也具有同样的优势，本文在软件升级问题实例上对比了 IBEA，NSGAII，SPEA2，NSGAIII 4 种演化算法的表现。选取这几种算法，一方面是参考 Sayyad 等的研究工作中对比的算法，另一方面是因为它们都支持二进制类型的问题解。

在本实验中，设置种群大小为 30，迭代次数为 10，4 种演化算法的演化结果如表 3 所列。其中，第 2－6 列数据是不同的演化算法得到的帕累托集在各个目标函数上的值。HV 和 Spread 是衡量多目标演化算法得到的帕累托集的两个重要的质量指标，可以由 jMetal 演化算法框架方便地计算出来，Size 表示帕累托集包含解的数量。HV 代表帕累托集构成的前沿面所覆盖的空间的体积（Hyper Volume）[33]，这个体积越大，表示帕累托集越接近真实的帕累托前沿面，说明帕累托集的质量越高。Spread 代表帕累托集扩展的程度[34]，多目标演化算法希望得到的帕累托集能够在真实的帕累托前沿面上分布得更广泛、更均匀。Spread 的计算方式如式（5）所示：

$$Spread=\frac{d_f+d_l+\sum_{i=1}^{N-1}|d_i-\bar{d}|}{d_f+d_l+(N-1)\bar{d}} \tag{5}$$

其中，N 表示帕累托集中包含的点（解）的数量，d_i 表示两个相邻解之间的欧氏距离，$\bar{d}$ 是 d_i 的平均值，d_f 和 d_l 分别是帕累托集中的边界解与极值点（Extreme solution）之间的欧氏距离。当帕累托集中只包含一个解时，Spread 的值为 1。一个好的帕累托集应该包含各个目标的极值点（$d_f=d_l=0$），并且各个解分布均匀（$d_i-\bar{d}$ 趋近于 0），此时 Spread 的值接近于 0。因此，Spread 的值越小说明帕累托集的分布情况越好。

IBEA 与其他 3 种多目标演化算法对比的实验结果如表 3 所列。该结果显示，IBEA 得到的帕累托集中不仅包含更多的解，并且在 HV 和 Spread 两个质量指标上均表现出了一定的优势。因此，本文选择 IBEA 作为优化软件升级问题的多目标演化算法。

表 3 4 种多目标演化算法的帕累托集的对比

Table 3 Comparison of Pareto solution sets obtained by 4 MOEAs

Algorithm	$R(X)$	$C(X)$	$N(X)$	$Nd(X)$	$Us(X)$	HV	Spread	Size
IBEA	1414	103	34	375	14	0.33	0.29	5
	1153	103	35	448	16			
	1159	103	35	453	14			
	1473	104	33	365	14			
	1413	105	33	375	14			
NSGAII	1569	89	26	340	11	0.18	0.32	3
	1706	92	25	292	11			
	1707	91	25	291	11			
SPEA2	1521	117	36	375	12	0.18	0.38	3
	1736	120	36	310	13			
	1737	119	36	308	13			
NSGAIII	1521	117	36	375	12	0	1	1

4.2.3 探究中间解修正操作对演化结果的影响

为了探究增加中间解修正操作对演化算法结果的影响，本文设置初始种群的大小为 50（全部由 QuickSampler 得到的可行解构成）、迭代次数为 10，分别执行无修正操作的演化算法和有修正的操作的演化算法各一轮，然后计算帕累托解集中可行解所占的比例。

实验结果如表 4 所列。其中，无修正操作的演化算法得到的帕累托解集中包含 43 个解，超过了有修正操作的演化算法得到的帕累托解集；然而，无修正操作的帕累托解集中可行解的数量是 0，也就是说，无修正操作的演化算法最终得到的解全部为不可行解，这对软件升级问题的求解是没有意义的。有修正操作的演化算法得到的帕累托解集中全部为可行解，因此中间解修正操作对于演化结果的质量有显著的提升。

表 4 比较有无修正操作得到的可行解的情况

Table 4 Comparison of feasible solutions in Pareto Front with/without Correction

是否有修正操作	帕累托解集的大小	可行解的数量	可行解占比/%
无	43	0	0
有	15	15	100

4.2.4 探究 SATMOEA 框架与其他求解器的性能对比情况

为了探究 SATMOEA 框架求解软件升级问题的性能是否优于现有的其他求解器，本文对比了它们求解同一个 CUDF 实例得到的解在不同的升级目标上的表现，结果如表

5 所列。其中，$R(X)$表示需要移除原来系统中软件包的数量，$C(X)$表示原来系统中发生版本变更的软件包的数量，$N(X)$表示新增软件包的数量，$Nd(X)$表示不是最新版本的软件包的数量，$Us(X)$表示未满足已安装的软件包中的推荐的数量。前 5 行数据分别表示由现有的 5 种不同的求解器得到的升级方案计算出的各个目标函数值，第 6－12 行数据表示 SATMOEA 框架得到的帕累托解集中的每一个解对应的各个目标函数值。

表 5 SATMOEA 框架与其他求解器在各个优化目标上的对比情况

Table 5 Comparison of optimization objectives with other solvers

solvers	$R(X)$	$C(X)$	$N(X)$	$Nd(X)$	$Us(X)$
cudf2msu	0	613	320	65	15
cudf2pbo	0	616	320	65	15
mccs	0	610	320	65	15
aspcud	0	614	320	65	15
p2cudf	0	615	320	65	15
SATMOEA	**0**	1005	**88**	986	45
	230	881	109	954	101
	0	968	**87**	1028	45
	696	**424**	405	915	98
	1171	629	346	**31**	101
	0	987	**88**	999	42
	0	988	**88**	998	42

需要注意的是，对于一个 CUDF 实例，其他求解器只能得到单一的升级方案，而 SATMOEA 框架运行一次就能够计算出一组帕累托最优解集，其中包含一系列可行的升级方案，用户可以从中选择不同的升级方案，以应对不同的场景。另外，从表 5 中加粗的数据可以看出，本框架能够找到不被其他求解器占优的解，即一个或多个目标函数值优于其他求解器相应的目标函数值。因此，当用户希望新增的软件包数量尽可能少时，可以选择 SATMOEA 框架提供的第 1，2，3，6，7 个解，对应表 5 中的第 6，7，8，11，12 行；而当用户希望位于最新版本的软件包尽可能多时，可以选择 SATMOEA 框架提供的第 5 个解，对应表 5 中的第 10 行。

为了更显著地表现 SATMOEA 框架的优势，我们将各个优化目标上能达到的最小值单独列出来，如表 6 所列。其中，第 1 行表示其他求解器的解综合起来所能达到的各个目标函数的最小值，第 2 行表示 SATMOEA 框架演化得到的帕累托解集中的解在各个目标上能够达到的最小值。数据表明，在 $R(X)$目标函数上，SATMOEA 框架所能达到的最小值与其他求解器相当，均为 0；在 $C(X)$，$N(X)$，$Nd(X)$3 个目标函数上，SATMOEA 表现出了明显的优势；然而，对于 $Us(X)$目标函数，SATMOEA 与其他求解器相比，表现更差。因此，SATMOEA 框架不仅在解的数量上明显优于其他求解器，而且在解的质量上也有一定的优势。

表 6 SATMOEA 框架与其他求解器在各个优化目标上的最小值对比情况

Table 6 Comparison of minimal values of all objectives with other solvers

solvers	$R(X)$	$C(X)$	$N(X)$	$Nd(X)$	$Us(X)$
others	0	610	320	65	**15**
SATMOEA	**0**	**424**	**87**	**31**	42

结束语 为了避免聚合的方式对不同升级目标之间的关系处理不当而引起的风险，本文从多目标优化的角度解决软件升级问题，成功构建了一个结合约束求解和多目标演化优化的软件升级问题求解框架 SATMOEA，并在演化算法的基础上增加了中间解修正算法，使帕累托解集中可行解的比例达到了 100%；与其他求解器只能提供单一的升级方案相比，该框架能够提供一系列不同的软件升级方案，以应对不同的软件升级场景，为软件升级问题的求解研究提供了一种新的思路。同时，由于演化算法需要对种群中每个个体计算适应度等，一定程度上增加了计算量，这是演化算法先天的缺陷，未来将考虑采用并行化的演化算法对 SATMOEA 框架进行优化，以提高求解的效率。

参 考 文 献

[1] Debian -The Universal Operating System [EB/OL]. (2020-03-26)[2020-03-26]. https://www.debian.org.

[2] IGNATIEV A, JANOTA M, MARQUES-SILVA J. Towards efficient optimization in package management systems[C]// Proceedings of the 36th International Conference on Software Engineering. 2014: 745-755.

[3] REN Z, JIANG H, XUAN J, et al. Analyzing Inter-objective Relationships: A Case Study of Software Upgradability[C]// International Conference on Parallel Problem Solving from Nature. Cham: Springer, 2016: 442-452.

[4] TUCKER C, SHUFFELTON D, JHALA R, et al. Opium: Optimal package install/uninstall manager[C]// 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007: 178-188.

[5] ROBERTO D C. How to manage your software upgrades: tales from the Mancoosi frontline[EB/OL]. (2010-04-19)[2020-03-26]. http://www.dicosmo.org/MyOpinions/index.php? post/2010/04/19/101-how-to-manage-your-software-upgrades-tales-from-the-mancoosi-frontline.

[6] MANCINELLI F, BOENDER J, DI COSMO R, et al. Managing the complexity of large free and open source package-based software distributions[C]// Twenty-First IEEE/ACM International Conference on Automated Software Engineering (ASE'06). IEEE, 2006: 199-208.

[7] ARGELICH J, MANYA F. Partial Max-SAT solvers with clause learning[C]// International Conference on Theory and Applications of Satisfiability Testing. Berlin: Springer, 2007: 28-40.

[8] ARGELICH J, LYNCE I, MARQUES-SILVA J. On solving Boolean multilevel optimization problem-s[C]// Twenty-First International Joint Conference on Artificial Intelligence. 2009.

[9] LE BERRE D, RAPICAULT P. Dependency mana-gement for the eclipse ecosystem: eclipse p2, metadata and resolution[C]// Proceedings of the 1st international workshop on Open component eco-systems. 2009: 21-30.

[10] TREZENTOS P, LYNCE I, OLIVEIRA A L. Apt-pbo: solving the software dependency problem using pseudo-boolean optimization[C]// Proceedings of the IEEE/ACM international conference on Automated software engineering. 2010: 427-436.

[11] RALF T, STEFANO Z. Common Upgradeability Description Format (CUDF) 2.0[EB/OL]. (2009-11-24)[2020-03-26]. ht-

tp://www.mancoosi.org/reports /tr3.pdf.

[12] MICHEL C,RUEHER M. Handling software upgradeability problems with MILP solvers[J]. Electronic Proceedings in Theoretical Computer Science,2010,29(1):1-10.

[13] GEBSER M,KAMINSKI R,SCHAUB T. aspcud:A linux package configuration tool based on answer set programming[J]. Electronic Proceedings in Theoretical Computer Science,2011,65(2):12-25.

[14] ARGELICH J,BERRE D L,LYNCE I,et al. Solving Linux upgradeability problems using boolean optimization[J]. Electronic Proceedings in Theoretical Computer Science,2010,29(2):11-22.

[15] JANOTA M,LYNCE I,MANQUINHO V,et al. PackUp:Tools for package upgradability solving[J]. Journal on Satisfiability,Boolean Modeling and Computation,2012,8(1/2):89-94.

[16] HARMAN M. The current state and future of search based software engineering[C]// Future of Software Engineering (FOSE'07). IEEE,2007:342-357.

[17] DURILLO J J,ZHANG Y Y,ALBA E,et al. A study of the multi-objective next release problem[C]// 2009 1st International Symposium on Search Based Software Engineering. IEEE,2009:49-58.

[18] WALCOTT K R,SOFFA M L,KAPFHAMMER G M,et al. Timeaware test suite prioritization[C]// Proceedings of the 2006 international symposium on Software testing and analysis. 2006:1-12.

[19] SARRO F,PETROZZIELLO A,HARMAN M. Multi-objective software effort estimation[C]// 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE,2016:619-630.

[20] HENARD C,PAPADAKIS M,HARMAN M,et al. Combining multi-objective search and constraint solving for configuring large software product lines[C]// 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE,2015,1:517-528.

[21] DUTRA R,LAEUFER K,BACHRACH J,et al. Efficient sampling of SAT solutions for testing[C]// 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE,2018:549-559.

[22] MANCOOSI project. International Solver Competition 2012 [EB/OL]. [2020-03-26]. http://ww w. mancoosi. org/misc-2012/index. html.

[23] DURILLO J J,NEBRO A J. jMetal:A Java framework for multi-objective optimization[J]. Advances in Engineering Software,2011,42(10):760-771.

[24] JIANG S J,WANG L S,XUE M,et al. Test Case Generation Based on Combination of Schema Using Particle Swarm Optimization[J]. Journal of Software,2016,27(4):785-801.

[25] BAO X A,BAO C,JIN Y T,et al. Com-binatorial Test Case Generation Method Based on Simplified Particle Swarm Optimization with Dynamic Adjustment[J]. Computer Science,2018,45(11):199-200.

[26] XUAN J F,REN Z L,WANG Z Y,et al. Progress on approaches to automatic program repair[J]. Journal of Software,2016,27(4):771-784.

[27] ZHOU M Q,JIANG G H. New Spectrum-based Fault Localization Method Combining Hitting Set and Genetic Algorithm[J]. Computer Science,2018,45(9):207-212.

[28] LU H,ZHANG L,YUE T. Differential IBEA for non-conformity resolution in interactive CPS production line configuration [J]. Journal of Software,2016,27(4):901-915.

[29] XIANG Y,ZHOU Y R,CAI S W. Integrating Preference in Many-objective Optimal Software Product Selection Algorithm [J]. Journal of Software,2020,31(2):282-301.

[30] MENG F C,CHU D H,LI K Q,et al. Solving SaaS components optimization placement problem with hybrid genetic and simulated annealing algorithm[J]. Journal of Software,2016,27(4):916-932.

[31] ZHENG Y J,ZHANG B,XUE J Y. Selection of key software components for formal development using water wave optimization[J]. Journal of Software,2016,27(4):933-942.

[32] SAYYAD A S,MENZIES T,AMMAR H. On the value of user preferences in search-based software engineering:a case study in software product lines[C]// 2013 35th International Conference on Software Engineering (ICSE). IEEE,2013:492-501.

[33] ZITZLER E,THIELE L. Multiobjective evolutionary algorithms:a comparative case study and the strength Pareto approach[J]. IEEE Transactions on Evolutionary Computation,1999,3(4):257-271.

[34] DEB K,PRATAP A,AGARWAL S,et al. A fast and elitist multi objective genetic algorithm:NSGA-II[J]. IEEE Transactions on Evolutionary Computation,2002,6(2):182-197.

ZHAO Song-hui, born in 1995, postgraduate, is a student member of China Computer Federation. His main research interests include multi-objective optimization and search-based software engineering.

REN Zhi-lei, born in 1984, Ph.D, associate professor, is a member of China Computer Federation. His main research interests include evolutionary computation, automatic algorithm configuration, data mining, and data analysis in software engineering.