软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(2):282−301 [doi: 10.13328/j.cnki.jos.005637]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

集成偏好的高维多目标最优软件产品选择算法*

向 毅[1], 周育人[1], 蔡少伟[2]

[1](中山大学 数据科学与计算机学院,广东 广州 510006)

[2](中国科学院 软件研究所,北京 100190)

通讯作者: 周育人, E-mail: zhouyuren@mail.sysu.edu.cn

摘 要: 在基于搜索的软件工程研究领域,高维多目标最优软件产品选择问题是当前的一个研究热点.既往工作主要采用后验方式(即先搜索再选择)处理软件工程师或终端用户的偏好.与此不同,将用户偏好集成于优化过程,提出了一种新算法以定向搜索用户最感兴趣的软件产品.在算法中,运用权向量表达用户偏好,采用成就标量化函数(achievement scalarizing function,简称 ASF)集成各个优化目标,并定义一种新关系比较个体之间的优劣.为了增强算法快速搜索到有效解的能力,分别采用 DPLL/CDCL 类型和随机局部搜索(SLS)类型可满足性(SAT)求解器实现了替换算子和修复算子.为了验证新算法的有效性,采用 21 个广泛使用的特征模型进行仿真实验,其中最大特征数为 62 482,最大约束数为 343 944.实验结果表明,基于 DPLL/CDCL 类型 SAT 求解器的替换算子有助于算法返回有效软件产品;基于 SLS 类型 SAT 求解器的修复算子有助于快速搜索到尽可能满足用户偏好的最终产品.在处理带偏好的高维多目标最优软件产品选择问题时,综合运用两类 SAT 求解器是一种行之有效的方法.

关键词: 基于搜索的软件工程;软件产品线;最优软件产品选择;高维多目标优化;用户偏好;SAT 求解器

中图法分类号: TP311

中文引用格式: 向毅,周育人,蔡少伟.集成偏好的高维多目标最优软件产品选择算法.软件学报,2020,31(2):282−301. http://www.jos.org.cn/1000-9825/5637.htm

英文引用格式: Xiang Y, Zhou YR, Cai SW. Integrating preference in many-objective optimal software product selection algorithm. Ruan Jian Xue Bao/Journal of Software, 2020,31(2):282−301 (in Chinese). http://www.jos.org.cn/1000-9825/5637.htm

Integrating Preference in Many-objective Optimal Software Product Selection Algorithm

XIANG Yi[1], ZHOU Yu-Ren[1], CAI Shao-Wei[2]

[1](School of Data and Computer Science, Sun Yat-Sen University, Guangzhou 510006, China)

[2](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: In search-based software engineering, one of the active research topics is the many-objective optimal software selection from software product lines. Previous works in this area mainly dealt with the preference from software engineers and end users in a posteriori way (namely selection after search). Different from that, this study integrates users' preference into the optimization process and proposes a new algorithm that can conduct a directed search for software products in which users are most interested. In the new algorithm, the users' preference is expressed as a weight vector, and an achievement scalarizing function (ASF) is used to aggregate all the optimization objectives. In addition, a new relation is defined to compare different individuals. To enhance the ability of the algorithm when searching for valid solutions, a substitution operator and a repair operator are implemented by using DPLL/CDCL-type and SLS-type SAT solvers, respectively. To verify the effectiveness of the new algorithm, 21 feature models widely used before are adopted to conduct simulation

* 基金项目: 国家自然科学基金(61906069, 61773410, 61502464); 广东省基础与应用基础研究基金(2019A1515011411, 2019A1515011700); 中国博士后科学基金(2019M662912); 中央高校基本科研业务费专项资金(x2rjD2190840/2019MS088)

Foundation item: National Natural Science Foundation of China (61906069, 61773410, 61502464); Guangdong Basic and Applied Basic Research Foundation (2019A1515011411, 2019A1515011700); Project Funded by China Postdoctoral Science Foundation (2019M662912); Fundamental Research Funds for the Central Universities (x2rjD2190840/2019MS088)

收稿时间: 2018-05-10; 修改时间: 2018-07-11; 采用时间: 2018-08-29

experiments. In these models, the largest number of features and that of constraints are 62 482 and 343 944, respectively. As shown by the experimental results, the substitution operator, which is based on a DPLL/CDCL-type SAT solver, is beneficial for returning valid software products, while the repair operator implemented by an SLS-type SAT solver contributes to find products that can satisfy the preference of users as much as possible. In summary, the simultaneous use of two types of SAT solvers is a feasible and effective way to handle the many-objective optimal software product selection problem where users' preference should be considered.

Key words: search-based software engineering; software product line; optimal software product selection; many-objective optimization; users' preference; SAT solver

基于搜索的软件工程(search-based software engineering,简称 SBSE)[1,2]运用元启发式搜索技术,比如遗传算法[3]、模拟退火算法[4]等,从问题的解空间出发解决软件工程的相关问题.它是传统软件工程和智能计算相结合的新兴研究领域[5].基于搜索的软件工程几乎适用于软件开发的所有阶段,包括需求分析、软件设计与开发、软件测试与维护等[6−10].

基于搜索的软件工程同样适用于软件产品线的配置问题.软件产品线(software product line,简称 SPL)[11]定义了一系列可通过模块化软件部件实现自动组装的软件产品.一般而言,这些软件产品具有一些通用功能.但是在特定的系统功能方面,各软件产品又存在差异.软件产品线定义的系列产品可由特征模型(feature model,简称 FM)简洁地表示[12].在 FM 中,每个特征(feature)代表软件产品线的一个部件或功能,特征模型则清晰地表达了各特征之间的约束关系[13].配置软件产品线时,除了考虑特征之间的约束关系(满足所有约束关系的配置称为可行或有效软件产品),一般还需要考虑软件工程师或终端用户的特殊要求,如总费用尽可能少、所选特征尽可能丰富、已知缺陷数尽可能少等[14−16].事实上,软件产品线的配置是一个离散优化问题,其搜索空间为 $\{0,1\}^{n_f}$,其中,1 表示某个特征被选中,而 0 则表示该特征未被选中,n_f 表示特征总数.配置软件产品线的任务是在决策空间中搜索到"最优"的可行软件产品,故软件产品线配置问题又称为最优软件产品选择问题(或最优特征选择问题).

根据优化目标的个数,一般可分为单目标最优软件产品选择方法和多目标最优软件产品选择方法[13].对于单目标最优软件产品选择算法,代表性的工作主要有:Yeh 等人[17]提出了基于遗传算法(GA)的软件产品线配置方法.在该方法中,优化目标为总费用最小化.White 等人[18]将最优软件(或特征)选择问题转化成一个多维多选择的背包问题(multidimensional multi-choice knapsack problem,简称 MMKP),并提出了 Filtered Cartesian Flattening(FCF)算法以搜索满足全局资源约束(即总预算)的最优软件产品.Müller[19]采用模拟退火算法从软件产品线选择最优软件产品.他使用了基于价值的投资组合优化方法,唯一的优化目标为最大化利润,定义为收益和费用之间的固定折衷.此外,Wang 等人[20]采用蚁群算法实现了近似最优软件产品的快速搜索.

基于单目标的最优软件产品选择方法仅考虑了特征的少量信息,有时无法很好地满足软件工程师或终端用户的实际需求[21].事实上,最优软件产品选择问题建模为多目标优化问题(multi-objective optimization problem,简称 MOP)更符合实际.在 MOP 中,至少存在 2 个优化目标且各目标之间往往相互冲突(即一个目标的改善通常会导致至少 1 个其他目标的退化).近几年来,演化多目标(evolutionary multi-objective,简称 EMO)研究领域广泛关注的高维多目标优化问题(many-objective optimization problem,简称 MaOP)是一类特殊的 MOP,其中的优化目标个数至少为 4[22].Sayyad 等人[14]于 2013 年首次提出了高维多目标最优软件产品选择问题.在选择最优软件产品时,他们考虑了 5 个优化目标,并运用两个特征模型比较了 7 种 EMO 算法(包括 NSGA-II[23]和 IBEA[24]等)的性能表现.实验结果表明,就解的质量、配置正确率和用户偏好满足情况而言,IBEA 优于其他 6 种 EMO 算法.自 Sayyad 等人的开创性工作以后,高维多目标最优软件产品选择问题引起了研究者们的广泛关注.Henard 等人[16]将 IBEA 与可满足性(satisfiability,简称 SAT)求解器相结合,提出了选择最优软件产品的新算法 SATIBEA.在该算法中,SAT 求解器的主要作用是实现新的变异和替换算子.同样地,SATIBEA 也需同时优化 5 个目标.在大规模特征模型上的实验结果表明,SATIBEA 具有较好的拓展性,其性能显著优于 Sayyad 等人[15]的算法.Hierons 等人[13]提出了 ShrInk Prioritise(SIP)策略,用于提升 EMO 算法在配置软件产品线时的性能.SIP 有机结合了一种新的编码方法(可压缩表示特征模型的染色体)和"1+n"优化策略(即首先优化作为第 1 个目标的约束违反数,然后同等优化其他目标).实验结果表明,在成功返回有效软件产品方面,基于 SIP 的 EMO 算法表现

出色.值得一提的是,在配置软件产品线时,Hierons 等人[13]首次采用具有真实属性或真实属性值范围的特征模型,这是该研究领域的一大突破.在此之前,特征模型的属性或属性值均是根据 Sayyad 等人[14]的建议人为构造或随机产生的.

本文作者于 2018 年将高维多目标演化算法 VaEA[25]与 SAT 求解器相结合,提出了配置软件产品线的 SATVaEA 算法[26].在该算法中,最优特征选择问题首先被转化成约束高维多目标优化问题,然后运用 VaEA 框架进行求解.为了提升 VaEA 的搜索能力,SATVaEA 使用了两种 SAT 求解器:一种是随机局部搜索(stochastic local search,简称 SLS)类型的 SAT 求解器,目的在于快速修复不可行解;另一种是 DPLL/CDCL 类型的 SAT 求解器(DPLL 和 CDCL 分别是 Davis-Putnam-Loveland-Logemann 程序[27]和 Conflict-Driven Clause Learning 算法[28]的缩写),目的在于提升软件产品的多样性.实验结果表明,与 SATIBEA 和 SIP 等算法相比,在返回多样化的有效软件产品方面,SATVaEA 更为有效和高效.在既往工作中[13−16,26],软件产品线配置算法返回的是折衷各个优化目标的一组 Pareto(近似)最优解.实际应用时,软件工程师或终端用户还需根据各自的偏好(或实际情况,如总预算等),从一组返回解中选择最终的某个解完成软件产品线的配置任务.事实上,既往工作[13−16,26]普遍采用后验方式(即先搜索再选择)[22,26]处理用户的偏好信息.然而有些情况下,终端用户仅对特定的软件产品感兴趣.此时,用户往往会给出自己的偏好信息,如认为某个优化目标更重要,则为相应目标赋予更大权重等.在上述应用场景,将用户偏好集成于优化过程,设计算法定向搜索用户最感兴趣的软件产品更具现实意义.一方面,可在一定程度上避免搜索一组(比如 100 个)折衷解所付出的大量时间开销;另一方面,又可以更大的概率搜索到尽可能满足用户偏好的最终软件产品.

本文针对带偏好的高维多目标最优软件产品选择问题展开研究,提出了集成用户偏好信息的高维多目标软件产品选择算法 PreEA.该算法的基本思想是,采用成就标量化函数(achievement scalarizing function,简称 ASF)[29]集成各个优化目标,运用与 SATVaEA 算法类似的 SAT 求解器增强算法的搜索能力.PreEA 将用户偏好集成于优化过程,每次运行仅返回(尽量)满足用户偏好的唯一解.最后,运用文献[26]中的 21 个特征模型,验证 PreEA 算法的有效性.

本文第 1 节介绍最优软件产品选择有关的背景知识.第 2 节详细介绍提出的 PreEA 算法.第 3 节是本文的实验部分.第 4 节对全文进行总结,并给出若干未来的研究方向.

1 背景知识

本节介绍配置软件产品线的相关背景知识.软件产品线是由共享软件部件构造相似软件产品集的一种软件工程方法、工具和技术[11].软件工程师通过软件产品线构造不同软件产品以满足不同用户的特定需求[16].运用软件产品线有助于降低软件开发费用、提高软件的可重用性以及简化软件维护[30].软件产品线定义了具有共性和可变性的系列软件产品,其共性和可变性由特征进行表达.这里,特征是软件产品的功能或性质的抽象[31].因此,特征模型是表示软件产品线的一种常用方法[12].

1.1 特征模型

图 1 是一个简化后的移动手机软件产品线所对应的特征模型(该图取自文献[26]).如图 1 所示,该特征模型包含 10 个特征 $x_1,x_2,\ldots,x_{10}$,其中有些特征是强制性的(mandatory),必须包含于其父代特征出现的每个软件产品,如特征“Calls”;有些特征是可选的(optional),可选择性地出现在软件产品中,如特征“GPS”.若父代特征出现时,其子代特征至少出现 1 个,则称子代特征与父代特征是“Or”的关系,如{“Camera”,“MP3”}与“Media”的关系;若子代特征出现且仅能出现 1 个,则称子代特征与父代特征是“Xor”的关系,如{“Basic”,“Color”,“High Resolution”}与“Screen”的关系.除了父代与子代之间的关系外,还有一类关系称为树间约束(cross-tree constraints),即以下形式的条件(“require”)或排斥(“exclude”)语句:若特征 F 出现,则特征 A 也必须出现(条件语句);若特征 F 出现,则特征 B 不能同时出现(排斥语句).例如,当“Camera”出现时,特征“High resolution”必须出现;“GPS”与“Basic”不能同时出现.

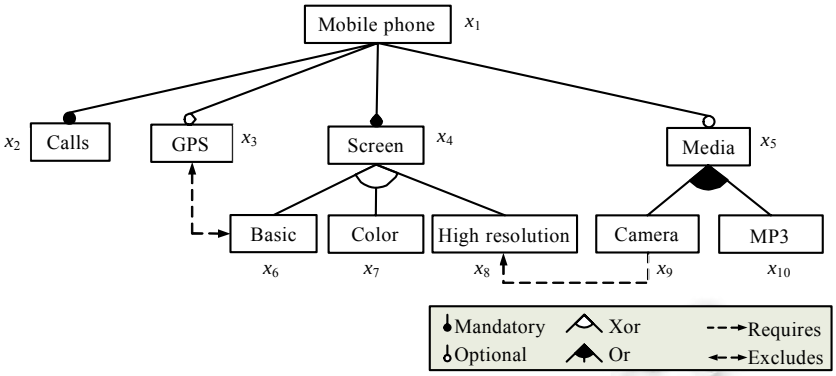

Fig.1 Feature model for a simplified mobile phone software product line

图 1 一个简化的移动手机软件产品线对应的特征模型

以上所有关系均可由命题公式表达.例如,图 1 中的特征模型可表示为

$$\left.\begin{aligned} FM = x_1 \wedge (x_1 \leftrightarrow x_2) \wedge (x_1 \leftrightarrow x_4) \wedge (x_3 \rightarrow x_1) \wedge (x_5 \rightarrow x_1) \wedge (x_4 \leftrightarrow xor(x_6, x_7, x_8)) \wedge \\ (x_5 \leftrightarrow (x_9 \vee x_{10})) \wedge (x_9 \rightarrow x_8) \wedge \neg(x_3 \wedge x_6) \end{aligned}\right\} \tag{1}$$

根据离散数学的相关知识,上述命题公式可进一步转换成合取范式(conjunctive normal form,简称 CNF),即

$$\left.\begin{aligned} FM = x_1 \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_4) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_5) \wedge (x_4 \vee \neg x_6) \wedge \\ (x_4 \vee \neg x_7) \wedge (x_4 \vee \neg x_8) \wedge (\neg x_4 \vee x_6 \vee x_7 \vee x_8) \wedge (\neg x_6 \vee \neg x_7) \wedge (\neg x_6 \vee \neg x_8) \wedge (\neg x_7 \vee \neg x_8) \wedge \\ (x_5 \vee \neg x_9) \wedge (x_5 \vee \neg x_{10}) \wedge (\neg x_5 \vee x_9 \vee x_{10}) \wedge (\neg x_9 \vee x_8) \wedge (\neg x_3 \vee \neg x_6) \end{aligned}\right\} \tag{2}$$

由公式(2)可知,图 1 中的特征模型转换成 CNF 后,共有 19 个约束(又称子句).这些约束必须同时满足才可构成有效的软件产品.

1.2 产品配置

与特征模型相对应的产品配置 C 指明了各特征在软件产品中是否出现,即 $C=\{l_1,l_2,\ldots,l_n\}$,其中,$l_i(i=1,2,\ldots,n_f)$取值 x_i 或$\neg x_i$,分别表示特征 x_i 被选中或未被选中.例如,对图 1 的特征模型,$C_1=\{x_1,x_2,x_3,x_4,x_5,\neg x_6,x_7,\neg x_8,\neg x_9,x_{10}\}$是一个软件配置,且该配置是有效的或可行的(满足公式(2)中的所有约束).相反,$C_2=\{x_1,x_2,x_3,x_4,x_5,\neg x_6,x_7,\neg x_8,x_9,x_{10}\}$是一个无效或不可行产品,因为它违反了公式(2)中的约束$(\neg x_9 \vee x_8)$.

1.3 离散高维多目标优化问题

针对软件产品线配置问题,本文抽象出以下离散高维多目标优化问题:

$$\text{Min } \boldsymbol{F}(\boldsymbol{x})=(f_1(\boldsymbol{x}),f_2(\boldsymbol{x}),\ldots,f_m(\boldsymbol{x})) \tag{3}$$

式中 $\boldsymbol{x}=(x_1,x_2,...,x_{n_f}) \in \{0,1\}^{n_f}$,其中,$x_i$=1 表示特征 x_i 被选中,而 x_i=0 表示特征 x_i 未被选中;n_f表示决策变量个数(即特征总数);$m \geqslant 4$ 表示目标个数;$\boldsymbol{F}(\boldsymbol{x})$称为 $\boldsymbol{x}$ 的目标向量.

为了定义优化目标,根据 Sayyad 等人[14]的建议,每个特征 x_i 均附加 3 个属性:*cost*,*used before* 和 *defects*,其属性值由均匀分布随机产生.具体地,*cost* 取值为 5.0~15.0 之间均匀分布的随机数,*defects* 取值为 0~10 之间的随机整数,而 *used before* 则取随机布尔值(true 或 false).此外,*used before* 和 *defects* 属性之间存在依赖关系,即如果 *used before* 为假,则 *defects*=0.

在高维多目标最优软件产品选择问题中,通常考虑的优化目标有[26]:

- $f_1(\boldsymbol{x})$定义为未被选中的特征总数,即$f_1(\boldsymbol{x})=|\{x_i|x_i=0\}|$,其中,$|\cdot|$表示集合的势.对于一个软件产品而言,未被选中的特征应尽可能少,以便提供更为丰富的功能.
- $f_2(\boldsymbol{x})$定义为之前未使用的特征总数,即 $f_2(\boldsymbol{x})=|\{x_i \mid x_i=1 \wedge x_i^{(used\ before)}=\text{false}\}|$.其中,$x_i^{(used\ before)}$ 表示 x_i 的 *used before* 属性值.考虑条件 x_i=1 的原因是,我们仅对当前被选中的特征是否之前被使用过进行判断.

由于之前未使用的特征更有可能是有错误的[13],因此我们希望最小化之前未使用的特征总数.

- $f_3(\boldsymbol{x})$定义为已知缺陷数,即 $f_3(\boldsymbol{x})=\sum_{(x_i=1)\wedge(x_i^{(used\ before)}=\text{true})} x_i^{(defects)}$.计算已知缺陷数的条件是当前特征 x_i 被选中(x_i=1)且该特征之前被使用过(即 $x_i^{(used\ before)}=\text{true}$).对于软件产品而言,已知缺陷数应尽能少.
- $f_4(\boldsymbol{x})$定义为总费用,计算公式为 $f_4(\boldsymbol{x})=\sum_{x_i=1} x_i^{(cost)}$.自然地,软件产品所需的总费用越少越好.

在实际应用中,根据软件工程师的经验或者终端用户的需求,也可考虑其他优化目标.本文选择以上优化目标,主要目的是与既往工作保持一致[13−16,26].关于上述优化目标的计算,我们给出以下实例:首先,根据 Sayyad 等人[14]的方法产生图 1 中特征模型的属性值,见表 1.假设当前解为 $\boldsymbol{x}$={1,0,1,1,0,1,1,0,1,0},则 $f_1(\boldsymbol{x})$=4,$f_2(\boldsymbol{x})$=2,$f_3(\boldsymbol{x})$=4+7+1+0=12,$f_4(\boldsymbol{x})$=10.30+9.21+13.31+13.22+9.18+9.69=64.91.

Table 1 Attribute values of the feature model used in Fig.1

表 1 图 1 中特征模型的属性值

特征编号	*cost*	*used before*	*defects*
1	10.30	true	4
2	5.25	false	0
3	9.21	false	0
4	13.31	true	7
5	9.71	false	0
6	13.22	true	1
7	9.18	true	0
8	10.89	true	9
9	9.69	false	0
10	12.57	true	10

2 PreEA 算法

为了从软件产品线搜索到尽可能满足用户偏好的唯一(近似)最优产品,本文提出了集成用户偏好的演化算法(preference based evolutionary algorithm,简称 PreEA).算法的基本思想是,采用 ASF[29]集成用户的偏好信息,将(高维)多目标优化问题转化成单目标优化问题,运用演化算子(交叉和变异等)产生新种群.为了增强演化算法的搜索性能,引入两种 SAT 求解器分别作为修复和替换算子.在给出 PreEA 算法的流程之前,首先介绍集成用户偏好的方法.

2.1 集成用户偏好

用户通过指定权向量$\boldsymbol{\omega}$=($\omega_1,\omega_2,\ldots,\omega_m$)给出对各个优化目标的偏好程度.若无特定偏好,则权向量可取(1/m, 1/m,…,1/m).集成用户偏好的 ASF 函数,其定义如下[29].

$$ASF(F(\boldsymbol{x}),\boldsymbol{\omega})=\max_{i=1,\ldots,m}\left\{\omega_i\frac{f_i(\boldsymbol{x})-z_i^*}{z_i^{nad}-z_i^*}\right\}+\rho\sum_{i=1}^{m}\left\{\omega_i\frac{f_i(\boldsymbol{x})-z_i^*}{z_i^{nad}-z_i^*}\right\} \tag{4}$$

其中,$\boldsymbol{x}$ 是当前个体,$\boldsymbol{F}(\boldsymbol{x})$是其目标向量,$z_i^*$ 和 z_i^{nad} 分别是理想点 $\boldsymbol{z}^*$和天底点 $\boldsymbol{z}^{nad}$ 的第 i 个分量.考虑到各目标函数值的变化范围一般不同,所以除以 $z_i^{nad}-z_i^*$ 以消除量纲.公式(4)中,ρ是较小正数,一般可取为 0.001[29].由文献[29]可知,通过最小化公式(4),可沿着$\boldsymbol{\omega}$确定的搜索方向找到 Pareto 前沿(PF)上的某个 Pareto 最优解,该解即对应用户感兴趣的特定软件产品.

配置软件产品线时,用户往往优先关注软件产品是否有效(无效产品对软件工程师和终端用户没有任何意义).若有效,才关注优化目标是否满足.因此,本文定义一种新的关系$\prec_{ASF}$比较两个解(解又可称为个体),具体地,

$$\boldsymbol{x}\prec_{ASF}\boldsymbol{y}\Leftrightarrow(VC(\boldsymbol{x})<VC(\boldsymbol{y}))\text{或}(VC(\boldsymbol{x})=VC(\boldsymbol{y})\text{且 }ASF(\boldsymbol{F}(\boldsymbol{x}),\boldsymbol{\omega})<ASF(\boldsymbol{F}(\boldsymbol{y}),\boldsymbol{\omega})) \tag{5}$$

其中,$VC(\boldsymbol{x})$表示个体 $\boldsymbol{x}$ 违反的约束个数.换言之,在新关系$\prec_{ASF}$下,个体 $\boldsymbol{x}$ 优于个体 $\boldsymbol{y}$ 当且仅当 $\boldsymbol{x}$ 的约束违反数比 $\boldsymbol{y}$ 少,或者二者的约束违反数相等,但是 $ASF(\boldsymbol{F}(\boldsymbol{x}),\boldsymbol{\omega})<ASF(\boldsymbol{F}(\boldsymbol{y}),\boldsymbol{\omega})$成立.由此可见,$\prec_{ASF}$偏好约束违反数少和 ASF 值更优的个体.

2.2 PreEA算法框架

PreEA 的框架如算法 1 所示.算法的输入为待配置的特征模型(FM)、用户给定的权向量($\boldsymbol{\omega}$)、种群规模(N)以及控制两类 SAT 求解器调用频率的参数($\theta\in[0,1]$).算法的输出则为最终种群中的最优个体 $\boldsymbol{x}_{best}$.

算法 1. PreEA 算法框架.

输入:特征模型(FM)、权向量($\boldsymbol{\omega}$)、种群规模(N)、控制两类 SAT 求解器调用频率的参数(θ).

输出:最终种群中的最优个体($\boldsymbol{x}_{best}$).

1: 化简特征模型 FM
2: 产生 N 个随机解,构成初始种群 $P=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_N\}$
3: 对每个 $\boldsymbol{x}\in P$,计算其目标向量 $\boldsymbol{F}(\boldsymbol{x})$和 $ASF(\boldsymbol{F}(\boldsymbol{x}),\boldsymbol{\omega})$(参考公式(4))
4: 由新关系$\prec_{ASF}$确定初始种群 P 中的最优个体,记为 $\boldsymbol{x}_{best}$
5: **while** 终止条件未满足 **do**
6: 运用遗传算子(交叉和变异)产生子代种群 Q
7: $S\leftarrow P\cup Q$ //S 是 P 和 Q 的并集
8: **if** S 中存在无效解 **then**
9: 随机选择 S 中的某个无效解 $\boldsymbol{x}_{invalid}$
10: **if** $r<\theta$ **then** //r 是位于[0,1]区间的随机数
11: 对 $\boldsymbol{x}_{invalid}$ 执行修复算子 //运用 SLS 类型 SAT 求解器修复 $\boldsymbol{x}_{invalid}$
12: **else**
13: 对 $\boldsymbol{x}_{invalid}$ 执行替换算子 //运用 DPLL/CDCL 类型 SAT 求解器产生有效解并替换 $\boldsymbol{x}_{invalid}$
14: **end if**
15: **end if**
16: 对每个 $\boldsymbol{x}\in S$,计算其目标向量 $\boldsymbol{F}(\boldsymbol{x})$和 $ASF(\boldsymbol{F}(\boldsymbol{x}),\boldsymbol{\omega})$
17: 根据$\prec_{ASF}$关系对 S 中的个体进行排序,选择前 N 个个体构成下一代种群 P
18: **if** $\boldsymbol{x}_1\prec_{ASF}\boldsymbol{x}_{best}$ **then** //$\boldsymbol{x}_1$ 是下一代种群 P 中的最优个体
19: $\boldsymbol{x}_{best}\leftarrow\boldsymbol{x}_1$ //更新 $\boldsymbol{x}_{best}$
20: **end if**
21: **end while**
22: **return** $\boldsymbol{x}_{best}$

PreEA 算法涉及到的主要操作如下.

2.2.1 化简特征模型

在配置软件产品线之前,有必要化简相应的特征模型.由第 1.1 节知,特征模型可表示成 CNF 形式的命题公式.因此,可采用布尔约束传播(Boolean constraint propagation,简称 BCP[32])对特征模型进行预处理,其基本思想是消除单文字约束.实施特征模型化简的好处有两个:一是可以识别出固定特征(fixed features),包括强制性特征(在任何产品中均须出现的特征,其赋值固定为 1)和废弃特征(在任何产品中均不能出现的特征,其赋值固定为 0);二是减少约束总数,有利于节约计算成本.实践表明,上述化简方法确实有助于大幅度降低特征模型的复杂度[26].例如公式(2)中的特征模型,x_1 是单文字,必须赋值为 1 才能满足第 1 个约束(即单文字 x_1 构成的约束).此时可删除包含 x_1 的所有约束,如$(x_1\vee\neg x_2)$,$(x_1\vee\neg x_4)$,$(x_1\vee\neg x_3)$和$(x_1\vee\neg x_5)$(因为这些约束已经满足).此外,通过删除文字$\neg x_1$,可进一步化简包含$\neg x_1$ 的约束,如$(\neg x_1\vee x_2)$和$(\neg x_1\vee x_4)$.经过一轮化简后,公式(2)中的特征模型可表示为

$$FM=x_2\wedge x_4\wedge(x_4\vee\neg x_6)\wedge(x_4\vee\neg x_7)\wedge(x_4\vee\neg x_8)\wedge(\neg x_4\vee x_6\vee x_7\vee x_8)\wedge(\neg x_6\vee\neg x_7)\wedge(\neg x_6\vee\neg x_8)\wedge(\neg x_7\vee\neg x_8)\wedge(x_5\vee\neg x_9)\wedge(x_5\vee\neg x_{10})\wedge(\neg x_5\vee x_9\vee x_{10})\wedge(\neg x_9\vee x_8)\wedge(\neg x_3\vee\neg x_6).$$

此时,x_2 和 x_4 均为单文字.可采用类似的方法继续化简,最终,

$$FM=(x_6\vee x_7\vee x_8)\wedge(\neg x_6\vee\neg x_7)\wedge(\neg x_6\vee\neg x_8)\wedge(\neg x_7\vee\neg x_8)\wedge(x_5\vee\neg x_9)\wedge$$
$$(x_5\vee\neg x_{10})\wedge(\neg x_5\vee x_9\vee x_{10})\wedge(x_8\vee\neg x_9)\wedge(\neg x_3\vee\neg x_6).$$

可以看到,所有约束均不含单文字,且约束个数仅为原来的一半左右.

2.2.2 遗传算子

化简特征模型后,运用演化算法在特征模型定义的决策空间搜索最优软件产品.首先,随机初始化规模为 N 的初始种群 $P=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_N\}$(算法 1 的第 2 行),在对每个个体进行初始化时,仅对非固定特征(又称可变特征)进行随机赋值(0 或 1),因此,PreEA 将一个软件配置编码为二进制位串.然后,计算每个个体的目标向量 $\boldsymbol{F}(\boldsymbol{x})$和 $ASF(\boldsymbol{F}(\boldsymbol{x}),\boldsymbol{\omega})$(算法 1 的第 3 行).紧接着,根据新关系$\prec_{ASF}$确定初始种群中的最优个体 $\boldsymbol{x}_{best}$.最后,运用遗传算子产生子代种群 Q(算法 1 的第 6 行).

具体地,首先运用二元锦标赛选择策略挑选出两个父代个体.在二元锦标赛选择策略中,随机挑选两个个体,运用新关系$\prec_{ASF}$确定其中较优者.重复两次二元锦标赛选择过程,则可选出两个父代个体.然后,将单点交叉算子[13]作用于父代,产生两个子代个体.单点交叉算子将第 1 个父代个体从初始点到交叉点的位串与第 2 个父代个体相应的位串进行交换.最后,采用位变异算子[33],以一定概率翻转各子代个体的位串.值得注意的是,我们仅对可变特征对应的变量进行位变异,这是因为翻转任何固定特征所对应的变量都将导致无效产品.重复上述程序,直到产生 N 个子代.合并父代和子代种群,形成包含 $2N$ 个个体的混合种群 S(算法 1 的第 7 行).

2.2.3 修复算子

若混合种群 S 中存在无效解,则随机选择其中之一,对其执行修复或替换算子(算法 1 的第 8 行~第 15 行).修复和替换算子的执行频率由控制参数θ决定.若产生的随机数 r 小于θ,则执行修复算子.在 PreEA 算法中,修复算子由 SLS 类型的 SAT 求解器实现,其目的在于提高算法的开发(exploitation)能力.具体实现时,本文选择简单有效的 ProbSAT[34].ProbSAT 是一类 SLS 类型 SAT 求解器.它根据概率分布选择待翻转的变量.下面详细介绍 ProbSAT 求解器.

ProbSAT 的流程如算法 2 所示.

算法 2. ProbSAT 算法流程.

输入:特征模型的某个赋值($\boldsymbol{x}$)和最大翻转次数(*maxFlips*).

输出:经变量翻转后的赋值 $\boldsymbol{x}$.

```
1:  for j=1 to maxFlips
2:    if x 已满足所有约束 then
3:      return x
4:    end if
5:    随机选择一个不满足的子句 C_u
6:    for i=1 to |C_u|
7:      x←子句 C_u 中第 i 个位置所对应的变量
8:        计算变量 x 的 break 值,即 break(x)
9:        计算σ(x)值   //σ(x)是 break(x)的函数,参考公式(6)或公式(7)
10:   end for
11:   var←根据公式(8)定义的概率选择变量 x
12:   翻转变量 var
13: end for
14: return x
```

算法的输入为特征模型的某个赋值$\boldsymbol{x}$(通常为无效解)和最大翻转次数(*maxFlips*),输出则为经变量翻转之后的赋值.算法允许的最大翻转次数为 *maxFlips*,若在此之前已找到满足所有约束的赋值,则算法结束(算法 2 的第 2 行~第 4 行).否则,随机挑选一个不满足的子句 C_u,计算该子句中每个变量 x 的 *break* 值及相应的$\sigma(x)$值(算法 2

的第 5 行~第 10 行).所谓变量 x 的 *break* 值,记为 $break(x)$,指的是在当前赋值下,翻转变量 x 后,由满足变为不满足的子句个数.一般而言,挑选翻转变量时优先选择具有较小 *break* 值的变量.$\sigma(x)$定义为 $break(x)$的函数,文献[34]给出了以下两种主要形式:

$$\sigma(x)=c_b^{-break(x)} \tag{6}$$

或

$$\sigma(x)=[\varepsilon+break(x)]^{-c_b} \tag{7}$$

其中,c_b>1 是控制参数,由用户给定或自适应调整;ε是常数,一般取为 1.

公式(6)中,$\sigma(x)$是 $break(x)$的指数函数;而公式(7)中,$\sigma(x)$是 $break(x)$的多项式函数.无论何种情况,$break(x)$越大,$\sigma(x)$的值就越小.实际上,$\sigma(x)$类似于变量 x 的适应值.$break(x)$越大,说明翻转 x 后有越多已满足的子句变为不满足.此时,x 的适应值应越小.公式(6)和公式(7)定义的$\sigma(x)$符合上述性质.根据$\sigma(x)$的值,可定义选择变量 x 进行翻转的概率:

$$p(x)=\frac{\sigma(x)}{\sum_{z\in C_u}\sigma(z)} \tag{8}$$

可以看到,若$\sigma(x)$越大,则变量 x 被选择的概率也越大.ProbSAT 根据公式(8)定义的概率选择某变量进行翻转(算法 2 的第 11 行).在具体实现时,可运用轮盘赌选择策略.下面以具有实例来说明 ProbSAT 算法的工作原理.假设对某个赋值 $\boldsymbol{x}$,随机选择的某个不满足子句为 $C_u=\{\neg x_1\vee x_6\vee x_9\vee\neg x_{10}\}$,并假设各个变量的 *break* 值分别为 20,4,100 和 50.关于变量 *break* 值的计算是 SAT 求解器中的一个重要课题,实际计算时可采用 Cai[35]提出的快速算法.假设ε=1,c_b=2.164[34].根据公式(7),各变量的$\sigma(x)$值分别为$(1+20)^{-2.164}$=0.0014,$(1+4)^{-2.164}$=0.0307,$(1+100)^{-2.164}$=4.5988×10^{-5} 和$(1+50)^{-2.164}=2.0175\times10^{-4}$.根据公式(8),各变量被选择的概率分别为 0.043 3,0.949 1,0.001 4,0.006 2.由此可见,变量 x_6 被选中的概率最大.事实上,该变量的 *break* 值最小,意味着翻转该变量后,由满足变为不满足的子句最少.

对无效解 $\boldsymbol{x}_{invalid}$,将其作为算法 2 的输入.执行完修复操作后,程序返回的最终解或者是有效的,或者仍是无效的(但是违反的约束个数会比之前的 $\boldsymbol{x}_{invalid}$ 更少).

2.2.4 替换算子

如算法 1 的第 13 行所示,当 $r\geqslant\theta$时,PreEA 会对无效解 $\boldsymbol{x}_{invalid}$ 执行替换操作.替换算子由 DPLL/CDCL 类型的 SAT 求解器实现.本文选择著名的 Sat4j[36].它是求解布尔可满足性及优化问题的 Java 库.事实上,Sat4j 的开发可追溯到 2004 年,当时是为了运用 Java 语言实现 Minisat 求解器[37].多年以来,Sat4j 受到研究人员的广泛关注.例如,在软件产品线工程研究领域,Sat4j 的应用十分广泛[16,26,38−40].考虑到 Sat4j 如此流行,本文也选择该求解器.

为了产生尽量多样化的随机解,以增强算法的探索(exploration)能力,根据文献[16]的建议,调用 Sat4j 之前需设置其变量赋值策略.具体的程序如算法 3 所示.PreEA 算法调用 Sat4j 求解器,尝试搜索有效解.若能找到有效解,则用其替换 $\boldsymbol{x}_{invalid}$.

算法 3. 随机化 Sat4j 的赋值策略.

```
1:  rand=RandomInt(1,3)   //rand 是位于[1,3]区间的随机整数
2:  if rand=1 then
3:      执行反文字赋值策略   //对变量 x 赋值为其反文字¬x
4:  else if rand=2 then
5:      执行正文字赋值策略   //对变量 x 赋值为其正文字 x
6:  else
7:      执行随机文字赋值策略   //对变量 x 赋值为正文字 x 或反文字¬x(二者的概率均为 0.5)
8:  end if
```

执行完修复或替换算子之后,对混合种群的每个个体计算目标向量 $\boldsymbol{F}(\boldsymbol{x})$和 $ASF(\boldsymbol{F}(\boldsymbol{x}),\boldsymbol{\omega})$.根据$\prec_{ASF}$关系选择排名靠前的 N 个个体构成下一代种群 P(算法 1 的第 16 行、第 17 行).若下一代种群中的最优个体比当前 $\boldsymbol{x}_{best}$

更优,则相应地更新 $\boldsymbol{x}_{best}$(算法 1 的第 18 行~第 20 行).

2.3 有关PreEA算法的讨论

PreEA 分别采用 SLS 类型和 DPLL/CDCL 类型 SAT 求解器(即 ProbSAT[34]和 Sat4j[36])实现了修复算子和替换算子,目的分别在于提高算法的开发和探索能力.从第 3 节的实验结果将看到,修复和替换算子确实有助于改善 PreEA 的整体搜索性能.

本文作者之前提出的 SATVaEA 算法同样运用了两类 SAT 求解器.PreEA 与 SATVaEA 的不同点主要体现在以下几方面.

- 两算法的求解目的不同:SATVaEA 算法的目的是返回一组折衷各优化目标的(近似)Pareto 最优解,而 PreEA 则是返回尽可能满足用户偏好的唯一解(满足用户偏好的程度由 ASF 值刻画,参考公式(4)).图 2 展示了 SATVaEA 和 PreEA 运行 10 次获得的最终解(示意图).如图所示,SATVaEA 得到的是逼近真实 Pareto 前沿的一组解,而 PreEA 得到的是真实前沿上的某个(近似)解.可以看到,PreEA 算法 10 次运行的最终解(由实心圆圈表示)集中在某个特定区域.虽然每次运行获得的最终解存在一定差异,但是这些解均分布在目标区域周围,体现了 PreEA 算法的设计初衷,即定向搜索用户感兴趣的软件产品;
- 两算法处理偏好的方式不同:SATVaEA 采用后验方式(先搜索再选择)处理用户偏好,而 PreEA 则将用户偏好集成于优化过程;
- 两算法实现的 SLS 类型 SAT 求解器不同:SATVaEA 实现的 SLS 类型 SAT 求解器是快速 WalkSAT[35],而 PreEA 实现的是 ProbSAT[34].从第 3.3 节将看到,ProbSAT 比快速 WalkSAT 有更好的性能表现.

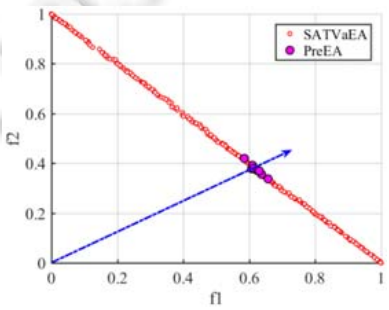

Fig.2 Difference between PreEA and SATVaEA

图 2 PreEA 与 SATVaEA 的区别

3 实验仿真

本节是论文的实验部分,首先介绍所使用的特征模型及终止条件,然后介绍本文的实验设置及性能指标,然后给出实验结果与分析以及控制参数对算法性能的影响,最后将 PreEA 与相关算法做进一步比较.

3.1 特征模型及终止条件

本文选择文献[26]中的 21 个特征模型进行仿真实验,各模型的特征总数、约束总数和终止条件见表 2.表中前 7 个特征模型取自文献[13],而后 14 个则取自 LVAT 库(LVAT 是 Linux VariabilityAnalysis Tools 的缩写,见网址 http://code.google.com/p/linux-variability-analysis-tools),该库是由 Berger 等人[41]通过逆向工程获取的实际特征模型.例如,Drupal 是一个开源内容管理框架,通过模块化部件可灵活地构建个人网站以及大型社区网站;uClinux 是微控制器领域中的 Linux 系统;FreeBSD 是一种类 Unix 操作系统;表 2 中命名为 2.6.*的 3 个特征模型则表示 x86 架构下 Linux 内核的配置选项.从表 2 可以看出,本文采用的特征模型的规模通常较大,最大规模高达 62 482.对于前 7 个特征模型,算法的终止条件为最大函数评估次数 *MaxFEs*,取值为 50 000.以上设置与文献[13]保持一致.对于剩下的特征模型,终止条件则为最大运行时间 *MaxRT*.根据特征模型的难易程度,*MaxRT* 可取不同值.如表 2 所示,*MaxRT* 的最小值和最大值分别为 6s 和 600s.本文 *MaxRT* 的设置与文献[26]保持一致.

Table 2 Twenty-one feature models used in this study

表 2 本文采用的 21 个特征模型

特征模型	特征总数	约束总数	终止条件
WebPortal	43	68	*MaxFEs*=50000
E-shop	290	426	
Drupal	48	79	
Amazon	79	250	
Random-10000	10 000	14 296	
RealAmazon	79	250	
RealDrupal	48	79	
toybox	544	1 020	*MaxRT*=6s
axTLS	684	2 155	
freebsd-icse11	1 396	62 183	*MaxRT*=30s
fiasco	1 638	5 228	
uClinux	1 850	2 468	
busybox-1.18.0	6 796	17 836	
2.6.28.6-icse11	6 888	343 944	*MaxRT*=200s
uClinux-config	11 254	31 637	
buildroot	14 910	45 603	*MaxRT*=400s
freetz	31 012	102 705	
coreboot	12 268	47 091	*MaxRT*=600s
embtoolkit	23 516	180 511	
2.6.32-2var	60 072	268 223	
2.6.33.3-2var	62 482	273 799	

最后需要说明的是,除 RealAmazon 和 RealDrupal 两特征模型外,其他所有特征模型均采用第 1.3 节定义的 4 个优化目标.对于上述两特征模型,Hierons 等人[13]根据模型的真实属性值(或属性值范围)定义了 8 个优化目标.在这些目标中,约束违反数是第 1 个目标,同时也是最重要的一个目标.由于本文定义的新关系$\prec_{ASF}$已优先考虑了该目标(参考公式(5)),因此对 RealAmazon 和 RealDrupal,仅需考虑剩下的 7 个优化目标(详情参考文献[13]).

3.2 实验设置及性能指标

为了探讨各算法部件对 PreEA 性能的影响,本文设计了如下变体算法.

- PreEA+WalkSAT.该算法将 PreEA 中的 ProbSAT 替换为快速 WalkSAT[35],而其他算法部件保持不变.在快速 WalkSAT 求解器中,Cai 提出了一种计算变量 *break* 值的快速算法.为公平起见,计算 *break* 值时(算法 2 的第 8 行),ProbSAT 也采用同样的方法.有关 *break* 值快速算法的详情见文献[35].
- PreEA−RO.该算法移除了 PreEA 中的修复算子(即 ProbSAT)但保留替换算子(即 Sat4j).在算法命名时,“−”表示“移除”,“RO”是 Repair Operator(修复算子)的首字母缩写.显然,PreEA−RO 表示移除 PreEA 算法中的修复算子.
- PreEA−SO.与 PreEA−RO 相反,该算法移除了 PreEA 中的替换算子(Sat4j)而保留修复算子(ProbSAT).在算法名称中,“SO”是 Substitution Operator(替换算子)的首字母缩写.
- PreEA−RO−SO.该算法同时移除修复和替换算子.

为了降低随机干扰,所有对比算法均独立运行 30 次.算法的权向量($\boldsymbol{\omega}$)、种群规模(N)、交叉概率(p_c)、变异概率(p_m)、调用频率(θ)以及与算子有关的参数,其具体设置见表 3.注意,本文所有的实验均将权向量取值为$\boldsymbol{\omega}$=(1/m,…,1/m),其中,m=4 或 m=7.事实上,权向量可根据用户的偏好设置为其他值.本文选择$\boldsymbol{\omega}$=(1/m,…,1/m)仅作为示例以说明问题.实验的硬件环境为:Intel i5 处理器、2.20GHz 主频、8.00GB 内存.

为了比较各算法,引入两个性能指标,分别为最终解的约束违反数(violated constraints,简称 VC)和 ASF 值(由公式(4)计算).VC 值反映了最终解所违反的约束个数,该值越小越好.理想情况下,VC 值为 0,说明未违反任何约束.此时,相应的解称为有效解.一般而言,软件工程师和终端用户仅对有效解感兴趣.因此,VC 是需优先考虑的指标.若两算法最终解的 VC 值均为 0(即均为有效解),此时可进一步比较各个解满足用户偏好的程度.公式(4)定义的 ASF 值可反映解满足用户偏好的程度,该值越小越好.在计算 ASF 值时,需指定理想点 $\boldsymbol{z}^*$和天底点 $\boldsymbol{z}^{nad}$.对各特征模型,SATVaEA 算法[26]可搜索到近似 PF.在近似 PF 中,各目标函数值的最小值构成的向量可当作 $\boldsymbol{z}^*$.

相应地,各目标函数值的最大值构成的向量可当作 $\boldsymbol{z}^{nad}$.

Table 3 Settings of the parameters

表 3 参数设置

参数	取值	说明
$\boldsymbol{\omega}$	$(1/m,\ldots,1/m)$	在 PreEA 算法中,引入权向量的目的是表达用户的偏好信息.若用户认为各优化目标同等重要,则可取权向量为$(1/m,\ldots,1/m)$.因此,对 RealAmazon 和 RealDrupal(考虑 7 个优化目标),权向量为(1/7,…,1/7);而对剩下的特征模型,权向量则为(1/4,…,1/4)
N	100	文献[26]的建议取值
p_c	1.0	
p_m	$1/n_f$	
θ	0.9	根据文献[26]的建议取值,后文第 3.4 节将通过实验确定较优的参数值
maxFlips	4 000	在 ProbSAT 和快速 WalkSAT 中,*maxFlips* 的设置与文献[26]保持一致
c_b	2.164	在 ProbSAT 中,根据公式(7)计算$\sigma(x)$.两参数的取值与文献[34]保持一致. 实践表明,上述参数值具有良好的实际效果
ε	1	
p_n	0.567	在快速 WalkSAT 中,需设置噪声参数(p_n),其取值与文献[35]保持一致

3.3 实验结果与分析

表 4 给出了 PreEA、PreEA+WalkSAT、PreEA−RO、PreEA−SO 和 PreEA−RO−SO 这 5 种算法配置所有特征模型的 VC 指标值.

Table 4 Mean and standard deviation (in brackets) of the performance metric VC for each algorithm

表 4 各算法 VC 指标的均值和标准差(见括号)

特征模型	PreEA	PreEA+WalkSAT	PreEA−RO	PreEA−SO	PreEA−RO−SO
WebPortal	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
E-shop	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Drupal	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Amazon	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Random-10000	0 (0)	0 (0)	0 (0)	**3 (1)**	**1 580 (40)**
RealAmazon	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
RealDrupal	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
toybox	0 (0)	0 (0)	0 (0)	0 (0)	**1 (1)**
axTLS	0 (0)	0 (0)	0 (0)	0 (0)	**2 (1)**
freebsd-icse11	0 (0)	0 (0)	0 (0)	0 (0)	**98 (24)**
fiasco	0 (0)	0 (0)	0 (0)	**9 (2)**	**48 (7)**
uClinux	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
busybox-1.18.0	0 (0)	0 (0)	0 (0)	0 (0)	**989 (98)**
2.6.28.6-icse11	0 (0)	0 (0)	0 (0)	**3 (3)**	**2 514 (281)**
uClinux-config	0 (0)	0 (0)	0 (0)	**5 (1)**	**1 547 (178)**
buildroot	0 (0)	0 (0)	0 (0)	**34 (5)**	**2 810 (229)**
freetz	0 (0)	0 (0)	0 (0)	**31 (4)**	**8 516 (284)**
coreboot	0 (0)	0 (0)	0 (0)	**34 (6)**	**2 277 (290)**
embtoolkit	0 (0)	0 (0)	0 (0)	**338 (248)**	**7 935 (284)**
2.6.32-2var	0 (0)	0 (0)	0 (0)	**671 (47)**	**17 940 (515)**
2.6.33.3-2var	0 (0)	0 (0)	0 (0)	**707 (53)**	**18 880 (545)**

由表 4 可知,对所有特征模型,PreEA、PreEA−RO 和 PreEA+WalkSAT 这 3 种算法每次运行均可返回有效解(3 种算法 VC 指标的均值和标准差均为 0).但是对于 PreEA−SO 和 PreEA−RO−SO 这两种算法而言,情况有所不同.在大规模的特征模型上,如 Random-10000,fiasco 以及从 2.6.28.6-icse11 到 2.6.33.3-2var 的所有特征模型,PreEA−SO 的最终解并不总是可行的.例如对 2.6.33.3-2var 模型,最终解违反的约束个数均值为 707.显然,PreEA−SO 搜索到的软件配置还存在大量未满足的子句.类似地,PreEA−RO−SO 算法在更多的特征模型上无法确保总是找到可行解.与 PreEA−SO 相比,PreEA−RO−SO 最终解违反的约束个数更多.例如在 2.6.33.3-2var 特征模型上,PreEA−RO−SO 对应的 VC 均值高达 18 880.在实际中,这些软件配置并无意义.PreEA−SO 和 PreEA−RO−SO 算法(均移除了替换算子,即 DPLL/CDCL 类型 SAT 求解器)面临返回无效解的风险,而保留该求解器的所有算法(即 PreEA、PreEA−RO 和 PreEA+WalkSAT)每次均能成功搜索到有效解.由此可见,在返回有效解方面,DPLL/CDCL 类型 SAT 求解器起到极为重要的作用.事实上,DPLL/CDCL 类型 SAT 求解器系统地遍历

整个决策空间.算法返回时,或者已找到可行解或者该问题本身是不可满足的.由 Mendonca 等人[40]和 Liang 等人[39]的讨论知,由实际特征模型转换而来的 SAT 实例均是可满足的,而且 SAT 求解器可容易地找到问题的可行解.因此,DPLL/CDCL 类型 SAT 求解器的高性能表现并非偶然,而是由实际特征模型的特性所决定的.因此在配置软件产品线时,DPLL/CDCL 类型 SAT 求解器应当鼓励使用.此外,若在移除替换算子 SO 的基础上进一步移除修复算子,则算法的整体性能进一步退化.由此可见,在返回有效解方面,同时使用两种 SAT 求解器(或者至少使用其中之一)是非常有必要的.

接下来,运用 ASF 指标比较各算法满足用户偏好的程度,实验结果见表 5,其中,最优值和次优值分别由深灰色和浅灰色背景标注.

Table 5 Mean and standard deviation (in brackets) of the performance metric ASF for each algorithm

表 5 各算法 ASF 指标的均值和标准差(见括号)

特征模型	PreEA	PreEA+WalkSAT	PreEA−RO	PreEA−SO	PreEA−RO−SO
WebPortal	1.187E−01 (4.13E−03)	1.241E−01 (5.00E−03)	1.229E−01 (7.67E−03)	1.252E−01 (4.61E−03)	1.202E−01 (5.99E−03)
E-shop	1.235E−01 (5.41E−03)	1.543E−01 (8.74E−03)	1.390E−01 (2.30E−02)	1.600E−01 (4.49E−03)	1.198E−01 (3.55E−03)
Drupal	1.223E−01 (7.12E−03)	1.200E−01 (5.84E−03)	1.224E−01 (3.42E−03)	1.187E−01 (2.94E−03)	1.193E−01 (4.39E−03)
Amazon	2.265E−01 (1.39E−02)	2.150E−01 (1.97E−02)	2.137E−01 (1.30E−02)	2.011E−01 (3.70E−02)	1.913E−01 (2.01E−02)
Random-10000	1.997E−01 (2.92E−02)	2.011E−01 (1.65E−02)	2.060E−01 (4.18E−02)	–	–
RealAmazon	1.231E−01 (3.34E−03)	1.225E−01 (4.39E−03)	1.240E−01 (9.66E−03)	1.127E−01 (1.03E−02)	1.128E−01 (1.36E−02)
RealDrupal	8.461E−02 (2.22E−03)	8.322E−02 (9.29E−04)	8.473E−02 (2.37E−03)	8.278E−02 (1.93E−03)	8.571E−02 (3.21E−03)
toybox	1.740E−01 (1.67E−02)	1.846E−01 (1.57E−02)	1.877E−01 (3.96E−02)	1.889E−01 (1.54E−02)	–
axTLS	1.671E−01 (1.05E−02)	1.925E−01 (2.94E−02)	1.690E−01 (3.72E−02)	1.949E−01 (1.12E−02)	–
freebsd-icse11	1.347E−01 (6.05E−03)	1.769E−01 (1.81E−02)	1.976E−01 (4.17E−02)	1.764E−01 (8.43E−03)	–
fiasco	1.725E−01 (2.44E−02)	2.013E−01 (2.03E−02)	1.815E−01 (2.96E−02)	–	–
uClinux	1.410E−01 (1.21E−02)	1.453E−01 (1.05E−02)	1.554E−01 (1.96E−02)	1.432E−01 (4.61E−03)	1.519E−01 (3.51E−03)
busybox-1.18.0	1.428E−01 (2.01E−02)	1.568E−01 (1.90E−02)	1.368E−01 (6.33E−03)	1.562E−01 (2.88E−03)	–
2.6.28.6-icse11	1.672E−01 (1.68E−02)	1.855E−01 (4.49E−02)	2.020E−01 (5.74E−02)	–	–
uClinux-config	1.503E−01 (3.42E−02)	1.744E−01 (4.45E−02)	1.439E−01 (1.59E−02)	–	–
buildroot	1.535E−01 (2.75E−02)	1.868E−01 (5.00E−02)	1.787E−01 (5.12E−02)	–	–
freetz	1.997E−01 (4.17E−02)	2.164E−01 (4.34E−02)	1.466E−01 (1.70E−02)	–	–
coreboot	2.417E−01 (4.21E−03)	2.381E−01 (2.40E−02)	1.312E−01 (1.08E−02)	–	–
embtoolkit	1.479E−01 (3.13E−02)	1.660E−01 (3.50E−02)	1.357E−01 (1.01E−02)	–	–
2.6.32-2var	1.850E−01 (5.05E−02)	1.754E−01 (5.54E−02)	2.221E−01 (4.58E−02)	–	–
2.6.33.3-2var	1.893E−01 (5.38E−02)	1.850E−01 (5.12E−02)	2.166E−01 (4.10E−02)	–	–

注意到,后两种算法在某些特征模型上并不能总是返回有效解(表中采用“/”进行标注).因为用户仅对有效软件产品感兴趣,上述情形下的 ASF 指标值可能无意义.因此,我们并未给出具体的 ASF 指标值.总体来看,

PreEA 表现最优,获得 9 个最优值和 7 个次优值.在某些大规模问题上,如 busybox-1.18.0、uClinux-config、freetz、coreboot 和 embtoolkit,PreEA–RO 算法的性能最优.事实上,该算法移除了修复算子,即 SLS 类型 SAT 求解器,但是保留了 DPLL/CDCL 类型 SAT 求解器.上述实验结果再一次说明,引入 DPLL/CDCL 类型 SAT 求解器的确有助于改善算法性能.与 PreEA 相比,移除替换算子后的 PreEA–SO 算法,其性能有较为明显的退化,尤其是对大规模的特征模型.这说明仅采用 SLS 类型 SAT 求解器不足以大幅度提高算法的性能.最后,比较 PreEA 和 PreEA+WalkSAT 两算法可以发现,ProSAT 求解器的性能优于 WalkSAT.具体地,在 14 个特征模型上,PreEA 的表现优于 PreEA+WalkSAT.根据上述讨论,在集成用户偏好的软件产品线配置方法中,DPLL/CDCL 类型 SAT 求解器结合 ProSAT 是一种行之有效的方法.

观察表 5 还可以发现,修复算子和替换算子分别适用于不同规模的特征模型.对多数小规模模型(如表 5 中除 Random-10000 之外的前 7 个特征模型),修复算子所起的作用比替换算子更明显.这是因为仅保留修复算子的 PreEA–SO 算法能够在 Drupal、RealAmazon 和 RealDrupal 这 3 个模型上取得最优 ASF 指标值(这 3 个模型的最大特征数仅为 79,最大约束数仅为 250).为何修复算子在小规模问题上表现优异?这是因为修复算子由 SLS 类型 SAT 求解器实现,这类求解器能够快速地搜索小规模问题的有效解.然而对大规模问题,SLS 类型 SAT 求解器不保证总能搜索到有效解.与之相反,DPLL/CDCL 类型 SAT 求解器每次运行总能搜索到有效解(由文献[39]知,由实际特征模型转换而来的 SAT 实例均是可满足的且往往有众多可行解).因此,仅保留替换算子的 PreEA–RO 在有些大规模特征模型上也表现出色(参考 PreEA–RO 在 busybox-1.18.0、uClinux-config、freetz、coreboot 和 embtoolkit 模型的 ASF 值).当然,两类 SAT 求解器的有机结合能够提高算法在不同规模特征模型上的整体性能表现.从表 5 的最后一列可发现,同时移除两种算子的 PreEA–RO–SO 算法在小规模特征模型上同样有较为出色的性能表现.其主要原因在于,这些小规模特征模型非常容易配置,仅采用演化算法也能获得不错的效果.但是当问题规模略微增大时,PreEA–RO–SO 算法则无法保证总能找到有效解(见表 4 的 VC 指标值).

由表 2 可知,前 7 个特征模型的终止条件为 *MaxFEs*=50000,而其他模型的终止条件为最大运行时间 *MaxRT*.因此,主要运用前 7 个特征模型比较各算法的实际运行时间.以 E-shop 和 Random-10000 为例,将各算法的实际运行时间绘制于图 3.

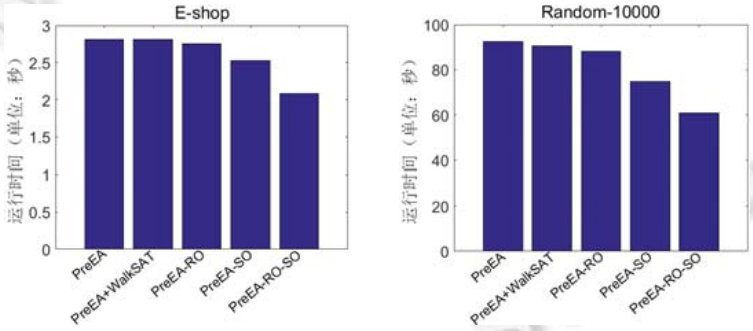

Fig.3 Runtime comparison among the algorithms

图 3 各算法实际运行时间的比较

如图 3 所示,PreEA–RO–SO 算法所需的运行时间最少,其次为 PreEA–SO,再次为 PreEA–RO.最后,PreEA 和 PreEA+WalkSAT 的运行时间相差不大.事实上,PreEA–RO–SO 未采用任何 SAT 求解器,故运行时间比其他算法都少;PreEA–SO 和 PreEA–RO 分别只采用了 SLS 类型和 DPLL/CDCL 类型 SAT 求解器,因此会比 PreEA 和 PreEA+WalkSAT 更省时(二者均使用了两种求解器).鉴于 SLS 类型 SAT 求解器一般比 DPLL/CDCL 类型 SAT 求解器更为高效,所以 PreEA–SO 的速度比 PreEA–RO 更快.在 PreEA 和 PreEA+WalkSAT 这两种算法中,DPLL/CDCL 类型 SAT 求解器均为 Sat4j,差别仅在于 DPLL/CDCL 类型 SAT 求解器.前者使用的是 ProbSAT,二者使用的则是快速 WalkSAT.此外,ProbSAT 和快速 WalkSAT 均采用同样的快速算法计算变量的 *break* 值.因此,PreEA 和 PreEA+WalkSAT 的运行速度相差不大.

3.4 参数θ的灵敏度分析

参数θ是 PreEA 的控制参数.不同参数取值会显著影响算法性能吗?为回答上述问题,拟将参数θ分别取值0.0,0.1,0.3,0.5,0.7,0.9 和 1.0,然后通过比较 VC 和 ASF 指标值得出结论.注意,当θ=0.0 时,由算法 1 知,PreEA 仅使用替换算子;当θ=1.0 时,PreEA 仅使用修复算子.以上两种特殊情况将在下文中重点讨论.对所有特征模型,不同θ对应的 VC 指标值见表 6.

Table 6 Mean of the performance metric VC for different values of θ

表 6 不同θ值对应的 VC 指标均值

特征模型	θ=0.0	θ=0.1	θ=0.3	θ=0.5	θ=0.7	θ=0.9	θ=1.0
WebPortal	0	0	0	0	0	0	0
E-shop	0	0	0	0	0	0	0
Drupal	0	0	0	0	0	0	0
Amazon	0	0	0	0	0	0	0
Random-10000	0	0	0	0	0	0	0
RealAmazon	0	0	0	0	0	0	0
RealDrupal	0	0	0	0	0	0	0
toybox	0	0	0	0	0	0	0
axTLS	0	0	0	0	0	0	0
freebsd-icse11	0	0	0	0	0	0	0
fiasco	0	0	0	0	0	0	0
uClinux	0	0	0	0	0	0	0
busybox-1.18.0	0	0	0	0	0	0	0
2.6.28.6-icse11	0	0	0	0	0	0	0
uClinux-config	0	0	0	0	0	0	0
buildroot	0	0	0	0	0	0	**1**
freetz	0	0	0	0	0	0	**3**
coreboot	0	0	0	0	0	0	**2**
embtoolkit	0	0	0	0	0	0	**5**
2.6.32-2var	0	0	0	0	0	0	**2**
2.6.33.3-2var	0	0	0	0	0	0	**4**

由表可知,除了θ=1.0 之外,其他所有的θ取值都能使得 PreEA 算法每次运行均返回有效解.对于最后 6 个大规模特征模型,仅运用修复算子(即θ=1.0)不能确保返回的最终解总是可行的.由相应的 VC 指标均值可知,尚存在一些约束最终解并未满足.表 7 给出了不同θ值对应的 ASF 指标均值.

Table 7 Mean of the performance metric ASF for different values of θ

表 7 不同θ值对应的 ASF 指标均值

特征模型	θ=0.0	θ=0.1	θ=0.3	θ=0.5	θ=0.7	θ=0.9	θ=1.0
WebPortal	1.392E−01	1.260E−01	1.174E−01	1.164E−01	1.136E−01	1.187E−01	1.146E−01
E-shop	1.373E−01	1.265E−01	1.194E−01	1.219E−01	1.200E−01	1.235E−01	1.242E−01
Drupal	1.277E−01	1.209E−01	1.201E−01	1.263E−01	1.219E−01	1.223E−01	1.203E−01
Amazon	1.629E−01	1.672E−01	2.005E−01	1.743E−01	1.745E−01	2.265E−01	2.051E−01
Random-10000	1.293E−01	1.357E−01	1.293E−01	1.468E−01	1.549E−01	1.997E−01	1.254E−01
RealAmazon	1.164E−01	1.216E−01	1.145E−01	1.144E−01	1.207E−01	1.231E−01	1.247E−01
RealDrupal	8.766E−02	8.429E−02	8.813E−02	8.621E−02	8.597E−02	8.461E−02	8.602E−02
toybox	1.369E−01	1.417E−01	1.481E−01	1.623E−01	1.492E−01	1.740E−01	1.766E−01
axTLS	1.501E−01	1.381E−01	1.349E−01	1.327E−01	1.435E−01	1.671E−01	1.584E−01
freebsd-icse11	1.783E−01	1.775E−01	1.260E−01	1.319E−01	1.276E−01	1.347E−01	1.356E−01
fiasco	1.577E−01	1.496E−01	1.597E−01	1.456E−01	1.664E−01	1.725E−01	1.810E−01
uClinux	1.261E−01	1.253E−01	1.296E−01	1.424E−01	1.333E−01	1.410E−01	1.402E−01
busybox-1.18.0	1.336E−01	1.555E−01	1.422E−01	1.547E−01	1.301E−01	1.428E−01	1.326E−01
2.6.28.6-icse11	1.679E−01	1.273E−01	1.646E−01	1.649E−01	1.672E−01	1.672E−01	1.587E−01
uClinux-config	1.430E−01	1.454E−01	1.810E−01	1.450E−01	1.897E−01	1.503E−01	1.447E−01
buildroot	1.724E−01	1.861E−01	2.186E−01	1.836E−01	1.547E−01	1.535E−01	–
freetz	1.292E−01	1.616E−01	1.629E−01	1.504E−01	1.365E−01	1.997E−01	–
coreboot	2.458E−01	2.457E−01	2.463E−01	2.445E−01	2.425E−01	2.417E−01	–
embtoolkit	1.296E−01	1.372E−01	1.309E−01	1.366E−01	1.367E−01	1.479E−01	–
2.6.32-2var	1.294E−01	1.272E−01	1.279E−01	1.291E−01	1.366E−01	1.850E−01	–
2.6.33.3-2var	1.297E−01	1.322E−01	1.306E−01	1.352E−01	1.885E−01	1.893E−01	–
表现最优个数	**6**	**4**	**3**	**3**	**2**	**2**	**1**

可以看出,不同θ值均可在特定的某个(某些)特征模型上取得最优指标值(最优值由深灰色背景标注).整体而言,θ=0.0 和θ=0.1 表现较优,分别获得 6 个和 4 个最优 ASF 指标值.当θ取值为 0.2~0.9 之间时,算法所获得的最优值个数相差不大.就 ASF 指标值而言,随着θ的增大,PreEA 的性能有逐步衰退的趋势;但是若考虑运行时间,则情况正好相反.图 4 给出了各θ值在 E-shop、RealAmazon 和 RealDrupal 这 3 个特征模型上,运行时间的变化趋势.如图所示,随着θ值的增大,算法所需的运行时间呈下降趋势.

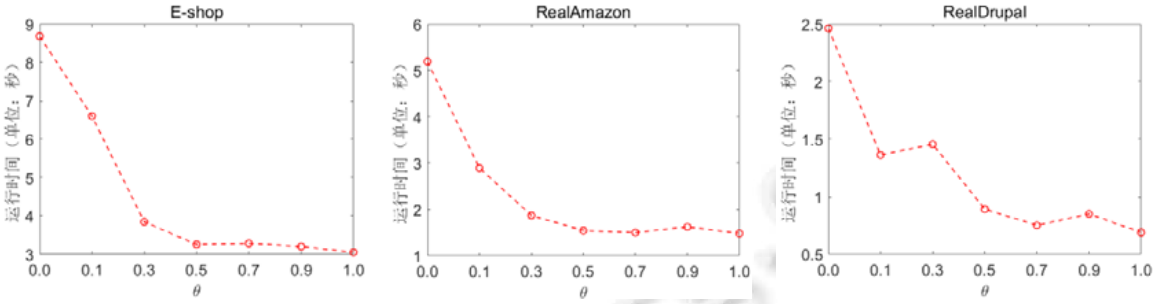

Fig.4 Runtime required for different values of θ

图 4 不同θ值所需的运行时间

下面对上述现象进行解释.如第 3.3 节所述,DPLL/CDCL 类型 SAT 求解器每次运行均能确保找到可行解,而 SLS 类型 SAT 求解器仅能改善当前不可行解(无法保证返回解总是可行的).因此,若θ值越小,则 PreEA 会越多地调用 DPLL/CDCL 类型 SAT 求解器,可探索越多可行解,有助于找到其中的较优解;相反,若θ值越大,则 PreEA 会越多地调用 SLS 类型 SAT 求解器.相对而言,这不利于算法探索更多可行解.特别地,若仅使用修复算子(即 SLS 类型 SAT 求解器),则在某些大规模特征模型上,最终返回解甚至是不可行的(参见表 6 的最后 1 列).一般而言,DPLL/CDCL 类型 SAT 求解器比 SLS 类型 SAT 求解器要费时.因此,若总的函数评估次数有限,则较大的θ值有利于节约计算时间.

关于参数θ对算法性能的影响,有以下结论.

- 当θ=1.0 时,意味着 PreEA 算法仅使用修复算子.此时,最终返回解不总是可行的.因此,实际应用时,不建议将θ取值为 1.0.
- 当θ=0.0 时,意味着 PreEA 算法仅使用替换算子.就 ASF 指标而言,此时算法的性能最佳,但是算法的运行速度则是最慢的.
- 当$\theta\in$(0.0,1.0)时,可折衷算法的效果和效率.用户可根据实际情况选择合适的θ值.例如,若最终解的质量比算法的效率更重要,则θ可取较小值;反之,则可取较大值.若无特别的偏好,可将θ设置为 0.5 左右.

3.5 进一步比较

在 PreEA 算法中,替换算子和修复算子分别由 DPLL/CDCL 类型和 SLS 类型 SAT 求解器实现.交换两类求解器后,算法的性能将如何变化?为此,设计了变体算法 PreEA*.在该算法中,替换算子由 SLS 类型 SAT 求解器(即 ProbSAT)实现.具体地,执行替换操作时,首先产生一个随机解,然后调用 ProbSAT 改善上述随机解,最后运用改善后的解替换混合种群中的某个随机不可行解;修复算子则由 DPLL/CDCL 类型 SAT 求解器(即 Sat4j[36])实现.此外,PreEA 以演化算法作为搜索引擎.将演化算法替换为爬山(hill climbing,简称 HC)算法[42,43],得到一个新算法 PreHC.爬山算法属于一种局部搜索算法,它从某个随机解出发,若当前解的邻域中存在比当前解更优的解,则用该解替换当前解.重复以上操作,直到满足终止条件.

PreHC 算法的流程如算法 4 所示.化简特征模型后,随机初始化最优解 $\boldsymbol{x}_{best}$,随后构造 $\boldsymbol{x}_{best}$ 的邻域 L.若邻域中存在无效解,则执行 PreEA 中相同的修复和替换算子.最后,将邻域 L 中的最优个体 $\bar{\boldsymbol{x}}$ 与 $\boldsymbol{x}_{best}$ 进行比较.在$\prec_{ASF}$关系下,若 $\bar{\boldsymbol{x}}$ 比 $\boldsymbol{x}_{best}$ 更优,则用 $\bar{\boldsymbol{x}}$ 替换 $\boldsymbol{x}_{best}$.在 PreHC 算法中,采用与文献[42]类似的方法构造邻域个体.具体地,

随机选择 $\boldsymbol{x}_{best}$ 的某个基因位,然后对该基因位的基因进行翻转.重复上述过程,可生产多个邻域个体.

算法 4. PreHC 算法框架.

输入:特征模型(FM)、权向量($\boldsymbol{\omega}$)、邻域规模(N)、控制两类 SAT 求解器调用频率的参数(θ).

输出:最优个体($\boldsymbol{x}_{best}$).

1: 化简特征模型 FM
2: 随机初始化 $\boldsymbol{x}_{best}$
3: **while** 终止条件未满足 **do**
4: L=*Neighbors*($\boldsymbol{x}_{best}$); //构造 $\boldsymbol{x}_{best}$ 的邻域 L,其规模为 N
5: **if** L 中存在无效解 **then**
6: 随机选择 L 中的某个无效解 $\boldsymbol{x}_{invalid}$,并执行修复或替换算子 //参见算法 1 的第 10 行~第 14 行
7: **end if**
8: 对每个 $\boldsymbol{x}\in L\cup\{\boldsymbol{x}_{best}\}$,计算其目标向量 $\boldsymbol{F}(\boldsymbol{x})$和 $ASF(\boldsymbol{F}(\boldsymbol{x}),\boldsymbol{\omega})$
9: 根据$\prec_{ASF}$关系确定 L 中的最优个体 $\bar{\boldsymbol{x}}$
10: **if** $\bar{\boldsymbol{x}}\prec_{ASF}\boldsymbol{x}_{best}$ **then**
11: $\boldsymbol{x}_{best}\leftarrow\bar{\boldsymbol{x}}$ //更新 $\boldsymbol{x}_{best}$
12: **end if**
13: **end while**
14: **return** $\boldsymbol{x}_{best}$

表 8 给出了 PreEA、PreEA*和 PreHC 等算法的 ASF 指标均值.为公平起见,上述算法中的参数θ均取为 0.9.在 PreHC 中,我们考虑邻域规模 N 为 50 和 100 两种情形.如表 7 所示,在 7 个特征模型上,PreEA 比 PreEA*更优;而在 14 个模型上,PreEA 比 PreEA*差.总体而言,交换替换和修复算子对应的 SAT 求解器之后,算法的性能有所提升.事实上,PreEA*算法以 0.9 的概率调用 DPLL/CDCL 类型 SAT 求解器,而以 0.1 的概率调用 SLS 类型 SAT 求解器.正如第 3.4 节的讨论,更多地调用 DPLL/CDCL 类型 SAT 求解器有利于算法性能的提升,但是同时会带来算法效率的降低.如表 9 所示,PreEA*比 PreEA 需要更多的运行时间.

Table 8 Mean of the performance metric ASF for PreEA, PreEA* and PreHC

表 8 PreEA、PreEA*和 PreHC 等算法的 ASF 指标均值

特征模型	PreEA	PreEA*	PreHC(N=50)	PreHC(N=100)
WebPortal	1.187E−01	1.158E−01	1.789E−01	1.702E−01
E-shop	1.235E−01	1.314E−01	1.518E−01	1.703E−01
Drupal	1.223E−01	1.319E−01	1.452E−01	1.275E−01
Amazon	2.265E−01	1.944E−01	1.439E−01	1.607E−01
Random-10000	1.997E−01	1.308E−01	2.466E−01	2.473E−01
RealAmazon	1.231E−01	1.137E−01	1.237E−01	1.062E−01
RealDrupal	8.461E−02	8.952E−02	9.366E−02	9.894E−02
toybox	1.740E−01	1.416E−01	1.967E−01	2.274E−01
axTLS	1.671E−01	1.269E−01	2.115E−01	1.890E−01
freebsd-icse11	1.347E−01	1.276E−01	2.375E−01	2.103E−01
fiasco	1.725E−01	1.550E−01	2.131E−01	1.914E−01
uClinux	1.410E−01	1.259E−01	1.873E−01	1.689E−01
busybox-1.18.0	1.428E−01	1.421E−01	2.374E−01	2.169E−01
2.6.28.6-icse11	1.672E−01	1.878E−01	2.473E−01	2.479E−01
uClinux-config	1.503E−01	1.818E−01	2.413E−01	2.417E−01
buildroot	1.535E−01	2.364E−01	2.440E−01	2.383E−01
freetz	1.997E−01	1.486E−01	2.490E−01	2.473E−01
coreboot	2.417E−01	2.466E−01	2.496E−01	2.471E−01
embtoolkit	1.479E−01	1.466E−01	2.290E−01	2.247E−01
2.6.32-2var	1.850E−01	1.278E−01	2.450E−01	2.451E−01
2.6.33.3-2var	1.893E−01	1.297E−01	2.440E−01	2.438E−01
PreEA 表现更优的个数		**7**	**20**	**19**
PreEA 表现更差的个数		**14**	**1**	**2**

Table 9 Comparison of the runtime between PreEA and PreEA* (s)

表 9 PreEA 和 PreEA*实际运行时间的比较 (秒)

特征模型	PreEA	PreEA*	比例(PreEA*的时间/PreEA 的时间)
WebPortal	0.68	0.87	1.29
E-shop	2.82	3.20	1.14
Drupal	0.70	0.83	1.19
Amazon	1.32	1.49	1.13
Random-10000	92.51	111.00	1.20
RealAmazon	1.20	1.54	1.29
RealDrupal	0.62	0.89	1.44

从表 8 的最后两列可以看出,PreEA 的性能明显优于 PreHC.具体地,PreEA 的表现比 PreHC(*N*=50)和 PreHC (*N*=100)更优的特征模型个数分别为 20 和 19.针对带偏好的高维多目标最优软件产品选择问题,演化算法比本节采用的爬山算法更具优势.作为一种局部搜索算法,爬山算法无法确保找到最优解,且所采用的贪心策略使得算法易于陷入局部最优[43].若在 PreHC(*N*=100)算法中同时移除修复算子和替换算子(该变体算法记为 PreHC–RO–SO),则算法的性能有明显的退化.如表 10 所示,PreHC–RO–SO 仅在 4 个特征模型上能够保证每次运行均返回有效解.对于其他特征模型,该算法的最终解违反了大量的约束,进而使得这些解没有实际应用价值.上述事实再一次说明,采用 SAT 求解器作为修复和替换算子对提升搜索算法的性能意义重大.

Table 10 Mean of the performance metric VC after removing the repair and substitution operators in PreHC

表 10 移除 PreHC 的修复和替换算子之后的 VC 指标均值

特征模型	PreHC (*N*=100)	PreHC–RO–SO
WebPortal	0	0
E-shop	0	**4**
Drupal	0	0
Amazon	0	**2**
Random-10000	0	**1 547**
RealAmazon	0	**2**
RealDrupal	0	0
toybox	0	**3**
axTLS	0	**6**
freebsd-icse11	0	**212**
fiasco	0	**58**
uClinux	0	0
busybox-1.18.0	0	**1 085**
2.6.28.6-icse11	0	**3 884**
uClinux-config	0	**1 550**
buildroot	0	**2 602**
freetz	0	**10 790**
coreboot	0	**2 164**
embtoolkit	0	**8 450**
2.6.32-2var	0	**19 780**
2.6.33.3-2var	0	**20 010**

4 结论与工作展望

本文提出了集成用户偏好信息的高维多目标最优软件产品选择算法 PreEA.在 PreEA 中,用户偏好由权向量给定,算法运用成就标量函数将高维多目标最优软件产品选择问题转化成单目标优化问题.为了增强算法的搜索能力,采用两类 SAT 求解器分别实现修复算子和替换算子.实验结果表明,DPLL/CDCL 类型 SAT 求解器在返回有效解方面起到了极为重要的作用;而在提高偏好满足程度方面,SLS 类型的 SAT 求解器具有积极表现;比较不同的 SLS 类型 SAT 求解器可以发现,ProSAT 的整体性能表现比快速 WalkSAT 更为出色;算法中的唯一控制参数θ可折衷算法的效果和效率;在算法框架中,演化算法的表现优于爬山算法.本文提出的 PreEA 实际上是一个算法框架,两类 SAT 求解器的具体实现分别为 Sat4j 和 ProSAT.根据实际情况,也可采用其他 SAT 求解器.本文对带偏好的软件产品线配置问题进行了初步探讨,未来可着眼于以下方面对该问题进行深入研究:(1) 探

讨集成用户偏好信息的其他可行方法.用户偏好信息的准确表达是算法设计的重要步骤,关系到最终解是否真的能满足实际需求;(2) 运用更多实际特征模型进行算法的性能测试,或者运用算法解决工业界真实的软件产品线配置问题;(3) 目前,本文采用的框架是演化算法,今后也可尝试其他算法,如粒子群优化算法[44]、分布式估计算法[45]等.

References:

[1] Harman M, Jones BF. Search-based software engineering. Information and Software Technology, 2001,43(14):833−839. [doi: 10.1016/S0950-5849(01)00189-6]

[2] Harman M, Mansouri SA, Zhang Y. Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys, 2012,45(1):Article No.11.

[3] Holland JH. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Cambridge: MIT Press, 1992. 20−26.

[4] Yao X, Chen GL. Simulated annealing algorithm and its applications. Journal of Computer Research and Development, 1990,27(7): 1−6 (in Chinese with English abstract).

[5] Li Z, Gong DW, Nie CH, Jiang H. Preface of the *Special Issue of Research on Search Based Software Engineering*. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):769−770 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4976.htm [doi: 10.13328/j.cnki.jos.004976]

[6] Greer D, Ruhe G. Software release planning: An evolutionary and iterative approach. Information and Software Technology, 2004, 46(4):243−253.

[7] Clark JA, Jacob JL. Protocols are programs too: The meta-heuristic search for security protocols. Information and Software Technology, 2001,43(14):891−904.

[8] Alba E, Chicano JF. Software project management with GAs. Information Sciences, 2007,177(11):2380−2401.

[9] Antoniol G, Penta MD, Harman M. Search-based techniques applied to optimization of project planning for a massive maintenance projec. In: Proc. of the 21st IEEE Int'l Conf. on Software Maintenance. Washington: IEEE Press, 2005. 240−249.

[10] Mcminn P. Search-based software test data generation: A survey. Software Testing, Verification & Reliability, 2004,14(2): 105−156. [doi: 10.1002/stvr.294]

[11] Long CA. Software product lines: Practices and patterns [Book review]. IEEE Software, 2002,19(4):131−132.

[12] Batory D. Feature models, grammars, and propositional formulas. In: Obbink H, Pohl K, eds. Proc. of the Int'l Conf. on Software Product Lines. Berlin, Heidelberg: Springer-Verlag, 2005. 7−20.

[13] Hierons RM, Li M, Liu X, Segura S, Zheng W. SIP: Optimal product selection from feature models using many-objective evolutionary optimization. ACM Trans. on Software Engineering and Methodology, 2016,25(2):Article No.17.

[14] Sayyad AS, Menzies T, Ammar H. On the value of user preferences in search-based software engineering: A case study in software product lines. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). San Francisco: IEEE, 2013. 492−501.

[15] Sayyad AS, Ingram J, Menzies T, Ammar H. Scalable product line configuration: A straw to break the camel's back. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Silicon Valley: IEEE Press. 2013. 465−474.

[16] Henard C, Papadakis M, Harman M, Traon YL. Combining multi-objective search and constraint solving for configuring large software product lines. In: Proc. of the 37th Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 517−528.

[17] Yeh JY, Wu TH. Solutions for product configuration management: An empirical study. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2005,19(1):39−47.

[18] White J, Dougherty B, Schmidt DC. Selecting highly optimal architectural feature sets with filtered Cartesian flattening. The Journal of Systems and Software, 2009,82(8):1268−1284.

[19] Müller J. Value-based portfolio optimization for software product lines. In: Proc. of the 15th Int'l Software Product Line Conf. Munich: IEEE, 2011. 15−24.

[20] Wang Y, Pang J. Ant colony optimization for feature selection in software product lines. Journal of Shanghai Jiaotong University (Science), 2014,19(1):50−58.

[21] Sayyad AS, Ammar H. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In: Proc. of the Int'l Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. San Francisco: IEEE, 2013. 21−27.

[22] Li B, Li J, Tang K, Yao X. Many-objective evolutionary algorithms: A survey. ACM Computing Surveys, 2015,48(1):1−35.

[23] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2002,6(2):182−197.

[24] Zitzler E, Künzli S. Indicator-based selection in multiobjective search. In: Yao X, *et al*., eds. Proc. of the Parallel Problem Solving from Nature—PPSN VIII (PPSN 2004). Berlin, Heidelberg: Springer-Verlag, 2004. 832−842.

[25] Xiang Y, Zhou Y, Li M, Chen Z. A vector angle-based evolutionary algorithm for unconstrained many-objective optimization. IEEE Trans. on Evolutionary Computation, 2017,21(1):131−152.

[26] Xiang Y, Zhou Y, Zheng Z, Li M. Configuring software product lines by combining many-objective optimization and SAT solvers. ACM Trans. on Software Engineering and Methodology, 2018,26(4):Article No.14.

[27] Davis M, Logemann G, Loveland D. A machine program for theorem-proving. Communications of the ACM, 1962,5(7):394−397.

[28] Marques-Silva JP, Sakallah KA. GRASP: A search algorithm for propositional satisfiability. IEEE Trans. on Computers, 1999, 48(5):506−521.

[29] Saborido R, Ruiz AB, Luque M. Global WASF-GA: An evolutionary algorithm in multiobjective optimization to approximate the whole Pareto optimal front. Evolutionary Computation, 2016,25(2):309−349.

[30] Knauber P, Bermejo J, Böckle G, *et al*. Quantifying product line benefits. In: van der Linden F, ed. Proc. of the Software Product-Family Engineering. LNCS 2290, Berlin, Heidelberg: Springer-Verlag, 2002. 155−163.

[31] Thum T, Batory D, Kastner C. Reasoning about edits to feature models. In: Proc. of the 31st Int'l Conf. on Software Engineering. Vancouver: IEEE, 2009. 254−264.

[32] Zabih R, Mcallester D. A rearrangement search strategy for determining propositional satisfiability. In: Proc. of the National Conf. on Artificial Intelligence. 1988. 155−160.

[33] Chicano F, Whitley D, Alba E, Exact computation of the expectation surfaces for uniform crossover along with bit-flip mutation. Theoretical Computer Science, 2014,545:76−93.

[34] Balint A, Schöning U. Choosing probability distributions for stochastic local search and the role of make versus break. In: Proc. of the Int'l Conf. on Theory and Applications of Satisfiability Testing. Berlin, Heidelberg: Springer-Verlag, 2012. 16−29.

[35] Cai S, Luo C, Su K. Improving WalkSAT by effective tie-breaking and efficient implementation. The Computer Journal, 2015, 58(11):2864−2875.

[36] Berre DL, Parrain A. The Sat4j library, release 2.2, system description. Journal on Satisfiability, Boolean Modeling and Computation, 2010,7(2-3):59−64.

[37] Eén N, Sörensson N. An extensible SAT-solver. In: Giunchiglia E, Tacchella A, eds. Proc. of the Theory and Applications of Satisfiability Testing. LNCS 2919, Berlin, Heidelberg: Springer-Verlag, 2004. 502−518.

[38] Xue Y, Zhong J, Tan TH, Liu Y, Cai W, Chen M, Sun J. IBED: Combining IBEA and DE for optimal feature selection in software product line engineering. Applied Soft Computing, 2016,49:1215−1231.

[39] Liang JH, Ganesh V, Czarnecki K, Raman V. SAT-based analysis of large real-world feature models is easy. In: Proc. of the 19th Int'l Conf. on Software Product Line. Nashville: ACM, 2015. 91−100.

[40] Mendonca M, Wasowski A, Czarnecki K. SAT-based analysis of feature models is easy. In: Proc. of the 13th Int'l Software Product Line Conf. San Francisco: Carnegie Mellon University, 2009. 231−240.

[41] Berger T, She S, Lotufo R, Wąsowski A. Variability modeling in the real: A perspective from the operating systems domain. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. Antwerp: ACM, 2010. 73−82.

[42] Mitchell M, Holland JH, Forrest S. When will a genetic algorithm outperform hill climbing? In: Cowan JD, Tesauro G, Alspector J, eds. Proc. of the Advances in Neural Information Processing Systems 6. San Mateo: Morgan Kaufmann Publishers, 1994. 1−9.

[43] Prügel-Bennett A. When a genetic algorithm outperforms hill-climbing. Theoretical Computer Science, 2004,320(1):135−153.

[44] Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. In: Proc. of the 6th Int'l Symp. on Micro Machine and Human Science. Nagoya: IEEE. 1995. 39−43. [doi: 10.1109/MHS.1995.494215]

[45] Baluja S. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report, No.CMU-CS-94-163, Pittsburgh: Carnegie Mellon University, 1994. 1−41.

附中文参考文献:

[4] 姚新,陈国良.模拟退火算法及其应用.计算机研究与发展,1990,27(7):1−6.

[5] 李征,巩敦卫,聂长海,江贺.基于搜索的软件工程研究专题前言.软件学报,2016,27(4):769−770. http://www.jos.org.cn/1000-9825/4976.htm [doi: 10.13328/j.cnki.jos.004976]

向毅(1986－),男,湖北恩施人,博士,CCF专业会员,主要研究领域为计算智能,多目标优化,基于搜索的软件工程.

周育人(1965－),男,博士,教授,博士生导师,主要研究领域为计算智能,多目标优化,算法设计与分析.

蔡少伟(1986－),男,博士,研究员,博士生导师,CCF 专业会员,主要研究领域为人工智能,复杂问题的有效解法.