测试用例集启发式约简算法分析与评价

游 亮 卢炎生
(华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 在软件开发和维护过程中,为了提高对源程序变更部分的信心并且保证源程序变更部分没有对未变更部分造成负面影响,需要对软件系统进行回归测试。回归测试是一个昂贵的测试过程。测试用例集约简算法是在仍然满足测试准则的前提下,通过删除所有冗余测试用例得到测试用例集的最小约简测试用例集,用以优化回归测试过程。综述了文献中主要的测试用例集启发式约简算法,通过统一的框架和术语定义了这些算法,分析和比较了这些算法的效率和优劣,指出了未来进一步研究的方向。

关键词 软件测试,软件维护,回归测试,测试用例集约简,测试用例集最小化
中图法分类号 TP311.5 **文献标识码** A

Analysis and Evaluation of Heuristic Algorithms for Test Suite Reduction

YOU Liang LU Yan-sheng
(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract During the development and maintenance of software, regression testing is used to enhance confidence to the modified parts of software and guarantee no side effect to the existing parts of software. Regression testing is an expensive process. Test suite reduction algorithm removes all redundant test cases among the test suite to get a minimal subset of test suite that still satisfy test criterion. This paper surveyed the most important heuristic algorithms for test suite reduction in the literature and used a unify framework and terminologies to define and analyze different algorithms. Typical heuristic algorithms for test suite reduction were analyzed and compared. The future work was presented.

Keywords Software testing, Software maintenance, Regression testing, Test suite reduction, Test suite minimization

在软件开发和维护过程中,修正软件的错误、增强软件的功能以及软件需求变更等会导致对源程序的变更。为了保证代码质量,提高对源程序变更部分的信心并且保证源程序变更部分没有对未变更部分造成负面的影响,需要对软件系统进行回归测试。回归测试是软件测试过程中花费最大的一个环节。它花费了整个测试过程80%的费用,消耗了整个软件维护过程50%的费用[1,2]。

传统开发流程中,每次构建整个源程序或者发布软件新版本前都需要进行回归测试。而在敏捷编程中,软件开发每次保存和编译时都需要对软件系统进行回归测试。新的开发模型中回归测试的频率增加,导致回归测试的花费进一步增大,使得对回归测试的优化变得更加重要。

软件测试的目标是针对特定的测试准则 C,设计和运行一个软件测试用例集 T 来达到测试准则 C。测试准则 C 可以用一个软件测试需求集 R 来表示,测试用例集 T 需要满足软件测试需求集 R 中的所有需求。例如语句覆盖准则下,程序的每一条语句就是一个软件测试需求。测试用例集 T 必须满足测试需求集 R,也就是测试用例集需要覆盖程序的每一条语句。在分支覆盖准则下,程序的所有分支构成了测试需求集,则测试用例集必须覆盖程序的所有分支。

随着程序的进化,新的测试用例被加入到回归测试用例集中,造成测试用例集膨胀,使得回归测试的花费越来越大。去除所有冗余的测试用例,对测试用例集进行约简,是优化回归测试的重要技术[3-5]。测试用例集 T 满足测试准则 C,一个测试用例 t 是冗余的当且仅当测试用例集的子集 $T-\{t\}$ 仍然能满足测试准则 C。

1 测试用例集约简问题

定义1 已知软件测试用例集 T 中有 m 个测试用例 $T=\{t_1,t_2,t_3,\cdots,t_m\}$,软件测试需求集 R 中有 n 个测试需求 $R=\{r_1,r_2,r_3,\cdots,r_n\}$。测试用例 T 与测试需求 R 之间的映射用满足关系 S 来定义。$S=\{(t,r)\in T\times R \mid t \text{ 满足 } r\}$。

定义2 满足关系 S 可以用满足关系矩阵来表示。满足关系矩阵是一个 $m\times n$ 的矩阵,其行表示测试用例,列表示测试需求。满足关系矩阵的第 i 行第 j 列的数值 $S_{ij}=1$ 表示第 i 个测试用例满足第 j 个测试需求;反之 $S_{ij}=0$ 表示第 i 个测试用例没有满足第 j 个测试需求。

定义3 对于任何 $t\in T$,所有 t 满足的测试需求集合记为 $Require(t)=\{r\in R \mid (t,r)\in S\}$。

定义4 对于任何 $T_1\subseteq T$,所有 T_1 满足的测试需求记为

到稿日期:2011-01-12 返修日期:2011-05-05 本文受国家部委预研基金项目(513150601)资助。
游 亮(1982—),男,博士生,主要研究方向为软件测试等,E-mail:youliang@live.com;卢炎生(1949—),男,教授,博士生导师,主要研究方向为数据库系统和软件工程等。

$Require(T_1)=\bigcup_{t\in T_1} Require(t)$。

定义 5 对于任何 $r\in R$，所有满足 r 的测试用例集记为 $Test(r)=\{t\in T|(t,r)\in S\}$。

定义 6 对于任何 $R_1\subseteq R$，所有满足 R_1 的测试用例集记为 $Test(R_1)=\bigcup_{r\in R_1} Test(r)$。

定义 7 若测试用例集 $T'\subset T$，并且 $Require(T')=Require(T)$，则称 T' 为测试用例集 T 的约简测试用例集。若不存在满足 $|T''|<|T'|$ 的约简测试用例集 T''，则称 T' 为测试用例集 T 的最小约简测试用例集，记为 $T_{\min}$。

定义 8 约简率 $k=\dfrac{|T|-|T_{\min}|}{|T|}\times 100\%$。其中 $|T|$ 表示集合 T 的元素个数，$|T_{\min}|$ 表示集合 $T_{\min}$ 的元素个数。

测试用例集约简问题就是求测试用例集的最小约简测试用例集。回归测试时利用最小约简测试用例集代替原来的测试用例集，可以在保证满足测试准则的前提下节省回归测试的花费。

如表 1 所列，测试用例集有 7 个测试用例 $T=\{t_1,t_2,t_3,t_4,t_5,t_6,t_7\}$，测试需求集有 8 个测试需求 $R=\{r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8\}$。其中 $Require(t_1)=\{r_1,r_3,r_5,r_6\}$ 表示测试需求 t_1 满足 r_1,r_3,r_5 和 r_6 这 4 个测试需求。$T_1=\{t_1,t_3,t_5\}$，则 $Require(T_1)=\{r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8\}=R$。根据满足关系矩阵还可知 $Test(r_2)=\{t_5\}$，这表示测试需求 r_2 只能被唯一的测试用例 t_5 满足。若 $R_1=\{r_1,r_4\}$，则 $Test(R_1)=\{t_1,t_3,t_5,t_6\}$。通过后文介绍的算法可得测试用例集 T 的最小约简测试用例集 $T_{\min}=\{t_1,t_3,t_5\}$。

表 1 满足关系矩阵

S	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
t_1	1	0	1	0	1	1	0	0
t_2	0	0	1	0	0	0	0	1
t_3	0	0	1	1	0	0	1	1
t_4	0	0	0	0	1	0	1	1
t_5	1	1	0	0	0	0	0	0
t_6	0	0	0	1	0	1	0	0
t_7	0	0	0	0	0	0	1	1

2 测试用例集启发式约简算法

测试用例集约简问题等价于最小集合覆盖问题，所以测试用例集约简是 NP 完全问题[6]。既然没有多项式时间的算法，那么各种启发式算法能保证得到足够好的近似解。

2.1 贪婪算法

Chvatal 提出的贪婪算法是一个简单高效的启发式算法[7]，简称 G 算法，即每次选择满足最多测试需求的测试用例，每迭代一次就把已经满足的测试需求从测试需求集中删除。如果满足最多测试需求的测试用例存在多个，那么就随机选择一个。测试需求全部被满足时，算法结束。

贪婪算法的最坏时间复杂度是 $O(mn\cdot\min(m,n))$。

t_1 和 t_3 同时都最多满足 4 个测试需求，随机选择一个，假设选择了 t_1，则 $Require(t_1)=\{r_1,r_3,r_5,r_6\}$ 有 4 个测试需求被满足，所以 $U=\{r_2,r_4,r_7,r_8\}$。第二轮中能满足最多测试需求的是测试用例 t_3，因为 $Require(t_3)=\{r_3,r_4,r_7,r_8\}$，所以 $U=\{r_2\}$。最后只有 t_5 能满足 r_2，所以第三轮 t_5 被选中。

G 算法求解得到最小约简测试用例集 $T_{min}=\{t_1,t_3,t_5\}$。

2.2 HGS 算法

Harrold 和同事们一起提出了 HGS 算法[8]。HGS 算法首先把 n 个测试需求根据 $|Test(r)|$ 大小不同分成 $R_1,R_2,\cdots,R_{\max}$ 共 MAX 个集合，$R_i=\{r\in R|Test(r)=i\}$。

迭代过程从 R_1 集合开始，首先因为这个集合中所有的测试需求只能被唯一的测试用例所满足，所以这些测试用例被加入到 $T_{\min}$ 中，然后把这些测试用例所能满足的测试需求标记为已经被满足。

接着考虑 R_2 集合，选择能满足 R_2 中最多测试需求的测试用例 t，把 t 加入到 T_{min} 中，并且把 $Require(t)$ 中的所有测试需求标记为已经被满足。如果能满足最多测试需求的测试用例不止一个，那么看这些测试用例谁能满足 R_3 中的最多测试需求。如果还是不止一个，接着看谁能满足 R_4 中的最多测试需求。如果直到 $R_{\max}$ 还是无法区分，就随机选择一个加入到 $T_{\min}$。

继续这个迭代过程，直到 R_2 集合中的所有测试需求被满足，就接着考虑 R_3 集合直到 $R_{\max}$，当所有的测试需求被满足时程序结束。

HGS 算法的最坏时间复杂度是 $O(n\cdot(n+m)\cdot MAX)$，其中 MAX 表示满足测试需求的测试用例集的基数的最大值。

利用 HGS 算法约简表 1，初始化时 $R_1=\{r_2\}$，$R_2=\{r_1,r_4,r_5,r_6\}$，$R_3=\{r_3,r_7\}$，$R_4=\{r_8\}$，MAX=4。

因为 $|Test(R_1)|=t_5$，所以选择测试用例 t_5。因为 $Require(t_5)=\{r_1,r_2\}$，所以 $R_2=\{r_4,r_5,r_6\}$。

Loop 循环中开始考虑 R_2，在用例测试集 $|Test(R_2)|$ 中 t_1 和 t_6 都出现了两次，所以接着考虑 R_3。因为在 $Test(R_3)$ 中 t_1 出现了一次，而 t_6 出现了零次，所以选择测试用例 t_1。因为 $Require(t_1)=\{r_1,r_3,r_5,r_6\}$，则 $R_2=\{r_4\}$，$R_3=\{r_7\}$。

R_2 还不为空，因为 t_3 和 t_6 都刚好出现一次，所以接着考虑 $Test(r_7)$。其中 t_3 出现了一次，而 t_6 出现了零次，所以选择测试用例 t_3。因为 $Require(t_3)=\{r_3,r_4,r_7,r_8\}$，所以 R_2，R_3，R_4 此时都为空，程序结束。

HGS 算法求解得到最小约简测试用例集为 $T_{\min}=\{t_1,t_3,t_5\}$。

2.3 GRE 算法和 GE 算法

Chen 提出了 GRE 算法和 GE 算法[9]。其中 GRE 算法使用 3 个策略对测试用例集进行约简。第一个策略是必选策略，即若某测试需求 r 只能被唯一测试用例所满足，则该测试用例必属于最小约简测试用例集。第二个策略是删除重复测试用例策略，即两个测试用例 t_i 和 t_j，如果 $Require(t_j)\sqsubseteq Require(t_i)$，则 t_j 相对于 t_i 来说是冗余的，所以可以安全地删除测试用例 t_j。第三个策略就是贪婪策略，选择满足当前测试需求最多的测试用例。

GRE 算法反复使用必选策略和删除重复测试用例策略，直到这两个策略无法使用时才使用贪婪策略。GRE 算法的最坏时间复杂度是 $O((n+m^2k)\cdot\min(m,n))$，其中参数 $k=\max_{1\leqslant i\leqslant m}(|Require(t_i)|)$。

GE 算法与 GRE 算法的唯一不同点在于不使用删除重复测试用例策略，也就是说 GE 算法只使用必选策略和贪婪策略。GE 算法的最坏时间复杂度与 G 算法相同，为 $O(mn\cdot\min(m,n))$。

一般说来，因为 GRE 算法比 GE 算法多使用了一个删除重复测试用例策略，所以 GRE 算法的结果要优于或者等于

GE算法的结果。但是Chen也举出了反例，在一些特殊情况下，GE算法会优于GRE算法。

利用GRE算法对表1进行约简，因为测试需求 r_2 只能被唯一的测试用例 t_5 满足，所以使用必选策略把 t_5 加入 T_{min}，接着把 t_5 和已经被 t_5 满足的测试需求 r_1 和 r_2 删除。此时，因为 $Require(t_2)\sqsubseteq Require(t_3)$ 并且 $Require(t_7)\sqsubseteq Require(t_4)$，所以可以使用删除重复测试用例策略安全地删除测试用例 t_2 和 t_7。这时必选策略和删除重复测试用例策略都无法使用了，所以采用贪婪策略把满足测试需求最多的测试用例 t_3 加入 T_{min}。接着使用必选策略把测试用例 t_1 加入 T_{min}，此时所有的测试需求都被满足，程序结束。

GRE算法得出最小约简测试用例集 $T_{min}=\{t_1,t_3,t_5\}$。

利用GE算法对表1进行约简，与GRE算法类似地使用必选策略将 t_5 加入 T_{min}，但因为不使用删除重复测试用例策略，所以使用贪婪策略将满足当前测试需求最多的测试用例 t_3 加入 T_{min}，最后还是使用贪婪策略将测试用例 t_1 加入 T_{min}。

GE算法得出最小约简测试用例集 $T_{min}=\{t_1,t_3,t_5\}$。

2.4 基于需求约简的算法

针对语句覆盖测试准则和分支覆盖测试准则，Agrawal提出了基于测试需求约简的算法[10,11]。其基本思想是通过分析测试需求之间的相互关系求解出一个约简测试需求集，然后针对约简的测试需求集设计和运行测试用例集。

定义9 对于测试需求集 R，若存在测试需求集的真子集 $R'\subset R$，并且 $Test(R')=Test(R)$，则称 R' 为测试需求集 R 的约简测试需求集。若不存在满足 $|R''|<|R'|$ 的约简测试需求集 R''，则称 R' 为测试需求集 R 的最小约简测试需求集，记为 R_{min}。

Marré进一步推广了Agrawal的思想，给出了基于控制流和数据流的各种常用测试准则的最小约简测试需求集的求解方法，并通过实验证明了这种方法的有效性[12]。

2.5 延迟贪婪算法

Tallam提出了延迟贪婪算法(简称DG算法)，将形式概念分析引入测试用例集约简算法中[13]。测试用例、测试需求、满足关系和满足关系矩阵分别对应于形式概念分析中的对象、属性、语境和语境表。

延迟贪婪算法使用4个约简策略对语境表进行约简，当语境表为空时，程序结束。它们分别是对象约简策略、属性约简策略、所有者约简策略和贪婪策略。延迟贪婪算法是将测试用例集约简问题变换到概念格上，然后在概念格上定义这4个约简策略的。延迟贪婪算法反复使用对象约简策略、属性约简策略和所有者约简策略进行约简，只有当这3个策略无法使用时才使用贪婪策略进行约简。整个算法执行过程中如果完全没有使用过贪婪策略，那么最小约简测试用例集就是优化解；如果算法执行过程中使用了贪婪策略，那么得到的是近似解。

虽然延迟贪婪算法的4个约简策略是在概念格上定义的，但是也可以直接在满足关系矩阵上定义和使用。它们分别与定义在关系矩阵上的删除重复测试用例策略、删除重复测试需求策略、必选策略和贪婪策略等价。

删除重复测试用例策略是指两个测试用例 t_i 和 t_j，若 $Require(t_j)\subseteq Require(t_i)$，则 t_j 相对于 t_i 来说是冗余的，可以安全地删除测试用例 t_j。删除重复测试用例策略与对象约简策略等价，即如果 $o_i\Rightarrow o_j$，则在语境表中删除 o_j 对应的行。

删除重复测试需求策略，即两个测试需求 r_i 和 r_j，如果 $Test(r_i)\subseteq Test(r_j)$，则测试需求 r_i 被满足，一定使得测试需求 r_j 被满足，所以 r_j 相对于 r_i 来说是冗余的，可以安全地删除测试需求 r_j。删除重复测试需求策略与属性约简策略等价，即如果 $r_i\Rightarrow r_j$，则在语境表中删除 r_j 对应的列。

必选策略是指，若某测试需求 r 只能被唯一测试用例 t 所满足，则该测试用例 t 必属于 T_{min}。必选策略与所有者约简策略等价，即对属性 r_k 的所有者约简需要先在语境表中删除唯一满足 r_k 的测试用例 t 所对应的行以及 t 满足的所有测试需求对应的列。

贪婪策略，选择满足当前测试需求最多的测试用例 t。在语境表中删除 t 所对应的行以及 t 满足的所有测试需求所对应的列。

```
Input:Context table for given test suite T.
Output:Set of test cases in minimized suite Tmin.
procedure Delayed-Greedy(Context table)
  Tmin=empty
  while(Context table ! = empty) do
    fInter=false;detectInter=0;
    while not(fInter) and (Context table ! = empty) do
        fInter=true;
  Step1:
    For each object implication oi⇒oj do
        Remove row for test case oj from Context table;
        fInter=false;
    endfor
  Step2:
    For each attribute implication ri⇒rj do
      Remove column for requirement rj from Context table;
      fInter=false;
    endfor
  Step3:
    For each attribute rk resulting in an owner reduction do
      Remove row for test case t that covers requirement rk;
      Remove columns for attributes covered by test case t;
      Tmin=Tmin∪{t};
      fInter=false;
    endfor
    endwhile
  Step4:
    if(Context table ! = empty) then
      Let t be test case picked using greedy coverage heuristic.
      Remove row for test case t from the Context table;
      Remove columns for attributes covered by test case t;
        Tmin=Tmin∪{t};detectInter=1;
    endif
  endwhile
    if(detectInter=0) then
        report minimized test suite Tmin is of optimal size.
    else
      report interference encountered.
    endif
      return(Tmin)
  endprocedure
```

DG算法中Step1表示对象约简策略，Step2表示属性约简策略，Step3表示所有者约简策略，Step4表示贪婪策略。标记变量detectInter初值为0。如果程序结束时detectInter

为1，则表示算法执行中使用了贪婪策略，得到的是近似解。反之，则表示程序执行过程中没有使用贪婪策略，得到的是优化解。

利用DG算法对表1进行约简，因为 $t_3 \Rightarrow t_2$ 和 $t_3 \Rightarrow t_7$，并且 $r_2 \Rightarrow r_1$ 和 $r_7 \Rightarrow r_8$，得到第一步约简。因为必选策略，所以选择测试用例 t_5，得到第二步约简。因为此时只能使用贪婪策略，又因为 t_1 和 t_3 都满足3个测试需求，所以随机选择测试用例 t_1，得到第三步约简。因为 $t_3 \Rightarrow t_4$ 和 $t_3 \Rightarrow t_6$，得到第四步约简。最后选择测试用例 t_3，约简语境表为空，程序结束。

DG算法得出最小约简测试用例集 $T_{min}=\{t_1, t_3, t_5\}$。

3 算法对比分析

G算法、HGS算法、GE算法、GRE算法和DG算法针对表1的约简都得到了相同的结果。但是在一般情况下，它们的约简率是不同的。使用上述5种算法分别对表2、表3和表4进行约简，其5种算法针对3个满足关系矩阵的约简结果分别记录在表5中。

表2 满足关系矩阵

S	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}
t_1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0
t_2	1	1	1	1	1	1	0	1	0	1	1	0	1	1	0	0	0	0	0
t_3	1	1	1	1	0	1	0	0	0	0	1	1	1	1	1	0	1	1	0
t_4	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	1	1	1	1
t_5	1	1	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
t_6	1	1	1	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1
t_7	1	1	1	1	1	0	1	0	1	1	0	0	0	0	1	0	0	0	0

表3 满足关系矩阵

S	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}
t_1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0
t_2	1	1	1	1	1	1	0	1	0	1	1	0	1	1	0	0	0	0	0
t_3	1	1	1	1	0	1	0	0	0	0	1	1	1	1	1	0	1	1	0
t_4	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	1	1	1	1
t_8	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	1	0	0	1
t_9	1	1	1	1	1	1	0	1	0	1	0	1	0	0	1	0	1	1	0

表4 满足关系矩阵

S	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}
t_1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0
t_3	1	1	1	1	0	1	0	0	0	0	1	1	1	1	1	0	1	1	0
t_4	1	1	1	1	0	1	0	0	0	0	0	1	0	0	1	1	1	1	1
t_5	1	1	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
t_6	1	1	1	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1
t_8	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	1	0	0	1
t_{10}	1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	0	0	0	0
t_{11}	1	1	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
t_{12}	1	1	1	1	1	1	1	0	1	1	0	1	0	0	1	0	1	1	0

表5 5种算法的运算结果

	Table 2 $T=\{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$	Table 3 $T=\{t_1, t_2, t_3, t_4, t_8, t_9\}$	Table 4 $T=\{t_1, t_3, t_4, t_5, t_6, t_8, t_{10}, t_{11}, t_{12}\}$
G	$\{t_1, t_2, t_3, t_4\}$	$\{t_1, t_2, t_3, t_4\}$	$\{t_5, t_8, t_{12}\}$
HGS	$\{t_1, t_2, t_3, t_4\}$	$\{t_1, t_2, t_4\}$	$\{t_3, t_5, t_8, t_{12}\}$
GE	$\{t_1, t_2, t_3, t_4\}$	$\{t_1, t_2, t_3, t_4\}$	$\{t_5, t_8, t_{12}\}$
GRE	$\{t_1, t_2, t_4\}$	$\{t_1, t_2, t_3, t_4\}$	$\{t_3, t_5, t_8, t_{12}\}$
DG	$\{t_1, t_2, t_4\}$	$\{t_1, t_2, t_4\}$	$\{t_5, t_8, t_{12}\}$

仔细分析表5的结果可知，针对表2的约简中，GRE算法和DG算法的结果都是$\{t_1, t_2, t_4\}$，优于其他算法的约简结果$\{t_1, t_2, t_3, t_4\}$。针对表3的约简中，HGS算法和DG算法的结果$\{t_1, t_2, t_4\}$优于其他算法的约简结果$\{t_1, t_2, t_3, t_4\}$。而针对表4的约简中，G算法、GE算法和DG算法的约简结果$\{t_5, t_8, t_{12}\}$优于HGS算法和GRE算法的约简结果$\{t_3, t_5, t_8, t_{12}\}$。综合可知，DG算法始终优于其他4个算法，而其它4个算法各有特点，不存在确定的包含关系。

DG算法的4个约简策略中，G算法只考虑了贪婪策略，GE算法只考虑了必选策略和贪婪策略。HGS算法和GRE算法都只是考虑了对象约简策略、必选策略和贪婪策略。只有DG算法另外还考虑了Agrawal首先提出的属性约简策略。延迟贪婪算法综合了以上算法的各种优点，使得它可以约简出更多的优化解而不是近似解，并且可以得到更高的约简率。

结束语 测试用例集约简是一种重要的回归测试优化技术，启发式算法是解决测试用例集约简问题的最重要方法。本文通过统一的术语与框架来深入分析比较G算法、HGS算法、GE算法、GRE算法和DG算法的优劣。结论是：(1)G算法是最简单并且引用率也最高的启发式算法。(2)DG算法是目前已知的最有效的启发式约简算法，其理论约简率是5种约简算法中最高的。(3)HGS算法、GE算法和GRE算法3者之间并不存在着绝对的优劣对比，3种算法的约简率各有优劣。

测试用例集约简算法仍然存在着需要进一步研究的方面：

1)已知约简算法的有效性验证和比较都是在中小型程序中进行的，迫切需要在工业级别的程序上验证和比较各种已知约简算法的有效性和效率。

2)启发式约简算法求解出的最小约简测试用例集仍然能满足原先的测试准则，但是发现程序错误的能力相比于原有的测试用例集可能会有很大的损失，这同样需要在工业级别的程序上进行验证。

3)针对单个测试准则的测试用例约简问题，启发式约简算法是非常简单和有效的。但是针对多测试准则的测试用例集约简问题，一般无法直接使用启发式约简方法，需要研究遗传算法和整数规划方法在多测试准则测试用例集约简问题下的应用。

参考文献

[1] Harrold M J, Orso A. Retesting software during development and maintenance [C]// Frontiers of Software Maintenance. Beijing, China: IEEE, 2008: 99-108

[2] Harrold M J. Reduce, reuse, recycle, recover: Techniques for improved regression testing [C]// IEEE International Conference on Software Maintenance(ICSM 2009). Alberta, Canada: IEEE, 2009: 5-5

[3] Rothermel G, Elbaum S, Malishevsky A G, et al. On test suite composition and cost-effective regression testing [J]. ACM Trans. Softw. Eng. Methodol, 2004, 13(3): 277-331

[4] Jeffrey D, Gupta N. Improving Fault Detection Capability by Selectively Retaining Test Cases During Test Suite Reduction[J]. IEEE Transactions on Software Engineering, 2007, 33(2): 108-123

[5] Li Z, Harman M, Hierons R M. Search Algorithms for Regression Test Case Prioritization[J]. IEEE Transactions on Software Engineering, 2007, 33(4): 225-237

(下转第177页)

5(a)可以看出，算法执行时间随着数据量的增大呈近似线性地增长，总体上4种算法的执行时间相似，LCCG和MDDCF算法的执行效率略好。由图5(b)可知，算法的执行时间都随着多敏感属性列数而变化，但幅度并不是很大。主要是由于L取值的变化并不影响MBF、MSDCF和MDDCF算法中桶的结构，因而影响并不是很大。但LCCG算法受L取值的重要影响是L取值越大，导致分组的成功率下降，剩余记录增多，进而影响了两次处理剩余记录的时间，导致后来执行时间的增大。图5(c)反映了随着多敏感属性列数C的增加，4种算法的时间也都随之增大。原因在于C值的增大使得多维桶的个数也随之增多，算法MBF、MSDCF和MDDCF对每个桶计算选择度的执行时间就增大。而算法LCCG执行时间的增加主要是因为随着C值的增大，一次分组成功率下降，剩余记录增加，进而导致了处理剩余记录的时间增加。

结束语 隐私信息的安全性是数据发布与共享环境中面临的重要问题。由于现有的隐私数据发布技术通常只针对具有单一敏感属性的数据或是没有考虑敏感属性的敏感度问题，而且在匿名化过程中将所有敏感属性值都同等对待，因此对于现实中大量存在的多敏感属性数据却无法保证其中隐私信息的安全。针对这一问题，本文提出了一种基于有损连接和相同敏感属性集的L-覆盖性聚类分组方法，并给出了方法的具体实现算法LCCG。在实际数据集上进行了大量的实验，结果表明在保证多敏感属性数据中隐私信息安全性及算法LCCG在不隐匿任何记录的前提下，其附加信息损失度和基于多维桶的方法相当。虽存在一定的平均概率泄漏度，但其值较小，且任一分组中的记录都满足L-覆盖性，可以保证发布数据的安全性，具有较高的数据发布质量。

参 考 文 献

[1] Zhao Y, Du M, Le L, et al. A Survey on Privacy Preserving Approaches in Data Publishing[C]// International IEEE Workshop on Database Technology and Applications. 2009: 128-131

[2] 周水庚，李丰，陶宇飞，等. 面向数据库应用的隐私保护研究综述[J]. 计算机学报，2009，32(5)：847-860

[3] Fung B C M, Wang Ke, Chen Rui, et al. Privacy-preserving data publishing: a survey on recent developments[J]. ACM Computing Surveys, 2010, 42(4): 1-55

[4] 魏琼. 数据发布中的隐私保护方法的研究[D]. 武汉：华中科技大学，2008

[5] 杨晓春，王雅哲，王斌，等. 数据发布中面向多敏感属性的隐私保护方法[J]. 计算机学报，2008，31(4)：574-587

[6] Sweeney L. K-anonymity: A model for protecting privacy[J]. International Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems, 2002, 10(5): 557-570

[7] Meyerson A, Williams R. On the complexity of optimal k-anonymity[C]// Proceedings of the 23rd ACM SIGACT-SIG-MOD-SIGART Symposium on Principles of Database Systems. Paris, France, 2004: 223-228

[8] Sweeney L. Achieving k-anonymity privacy protection using generalization and suppression[J]. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 2002, 10(5): 571-588

[9] Machanavajjhala A, Gehrke J, Kifer D, et al. l-diversity: Privacy beyond k-anonymity[C]// Proceedings of the 22nd International Conference on Data Engineering (ICDE). 2006: 24-36

[10] Truta T M, Vinay B. Privacy protection: p-sensitive k-anonymity property[C]// 2nd International Workshop on Privacy Data Management. 2006: 94-99

[11] Wong R C-W, Li Jiu-yong, Fu A W-C, et al. (α, k)-anonymous data publishing[J]. Journal of Intelligent Information Systems, 2009, 33(2): 209-234

[12] Li N, Li T, Venkatasubramanian S. T-closeness: privacy beyond k-anonymity and l-diversity[C]// Proceedings of the IEEE ICDE. 2007: 44-56

(上接第150页)

[6] Garey M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-Completeness [M]. New York: W. H. Freeman & Co, 1979

[7] Chvatal V. A Greedy Heuristic for the Set-Covering Problem [J]. Mathematics of Operations Research, 1979, 4(3): 233-235

[8] Harrold M J, Gupta R, Soffa M L. A methodology for controlling the size of a test suite[J]. ACM Trans. Softw. Eng. Methodol, 1993, 2(3): 270-285

[9] Chen T Y, Lau M F. A new heuristic for test suite reduction[J]. Information and Software Technology, 1998, 40(5/6): 347-354

[10] Agrawal H. Dominators, super blocks, and program coverage [C]// Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming anguages. Portland, Oregon, United States: ACM, 1994: 25-34

[11] Agrawal H. Efficient coverage testing using global dominator graphs[C]// Proceedings of the 1999 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. Toulouse, France: ACM, 1999: 11-20

[12] Marré M, Bertolino A. Using Spanning Sets for Coverage Testing[J]. IEEE Trans. Softw. Eng, 2003, 29(11): 974-984

[13] Tallam S, Gupta N. A concept analysis inspired greedy algorithm for test suite minimization[C]// Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. Lisbon, Portugal: ACM, 2005: 35-42