تعیین زیرمجموعه‌ای مناسب از موارد آزمون جهت افزایش نرخ کشف خطا در نرم‌افزارهای تغییر یافته

سعید پارسا[۱]، محمد عبداللهی ازگمی[۲]، علیرضا خلیلیان[۳]، یلدا فضل علیزاده[۴]

[۱]دانشکده مهندسی کامپیوتر دانشگاه علم و صنعت ایران
تهران، ایران
parsa@ iust.ac.ir
[۲] دانشکده مهندسی کامپیوتر دانشگاه علم و صنعت ایران
تهران، ایران
azgomi@iust.ac.ir
[۳]دانشکده مهندسی کامپیوتر دانشگاه علم و صنعت ایران
تهران، ایران
khalilian@comp.iust.ac.ir
[۴] دانشکده مهندسی کامپیوتر دانشگاه علم و صنعت ایران
تهران، ایران
ya_alizadeh@comp.iust.ac.ir

چکیده

آزمون نرم افزار، فرایندی است که پیوسته در چرخه توسعه و حیات نرم‌افزار انجام می‌شود. همگام با توسعه نرم‌افزار و عرضه نسخ متعدد از نرم‌افزار و افزودن موارد آزمون جدید برای آزمودن تغییرات نسخه، برخی از موارد آزمون افزونه می‌گردند. به این دلیل و نیز محدودیت زمان و منابع موجود برای آزمون مجدد، لازم است فنونی به‌کار گرفته شوند که با حذف دائمِ موارد آزمون افزونه از مجموعه آزمون، اندازه آن را در حد قابل قبولی حفظ نماید. این فرایند، کمینه‌سازی مجموعه آزمون نام دارد. حذف موارد آزمون افزونه که سبب کاهش زمان آزمون می‌شود، نباید با کاهش غیرقابل قبول در قدرت کشف خطای مجموعه حاصل نسبت به مجموعه آزمون اولیه همراه باشد. با این وجود، نتایج مطالعات تجربی بر روی فنون کاهش موجود، کاهش شدیدی را در این مورد نشان می‌دهد. برای حل مشکل هزینه از دست رفتن خطاها، راه‌حل جدیدی مبتنی بر افزونگی ارائه شده است. در این روش جدید، موارد آزمون افزونه نسبت به یک معیار پوشش خاص، اگر در سنجش با معیار پوشش دیگر، افزونه نباشند، از مجموعه آزمون حذف نمی‌شوند. نتیجه مطالعه تجربی، بهبود قابل توجهی را در قدرت کشف خطای مجموعه آزمون کاهش‌یافته در اثر ایجاد چنین افزونگی نشان می‌دهد.

کلمات کلیدی

آزمون رگرسیون نرم‌افزار[۱]، کاهش مجموعه آزمون[۲]، افزونگی کنترل‌شده در کاهش مجموعه آزمون‌ها.

[۱] software regression testing
[۲] test suite reduction

۱- مقدمه

آزمون، فرایند مهمی در چرخه توسعه و تولید نرم‌افزار است که به‌طور گسترده برای آشکارسازی خطاها در محیط‌های واقعی توسعه نرم‌افزار مورد استفاده قرار می‌گیرد [۳]. هزینه این فرایند، معمولاً بسیار زیاد است، چرا که زمان زیادی طول می‌کشد تا نرم‌افزار به ازای کل موارد آزمون اجرا گردد [۱۰]. به‌ویژه در زمان نگهداری نرم‌افزار، برای اعتبارسنجی تغییراتی که پیوسته در ساختار نرم‌افزار ایجاد می‌شود، مکرراً آزمون مجدد ضرورت می‌یابد.

همگام با افزودن موارد آزمون جدید، که برای آزمودن خصوصیات تغییر یافته نسخه جدید، به مجموعه آزمون اضافه شده‌اند، حجم زیادی از موارد آزمون، در هر مرتبه (هر ارائه نسخه تغییریافته از نرم‌افزار به همراه موارد آزمون جدید برای آزمودن این تغییرات)، افزونه می‌شوند [۹]. البته روشن است که هر چه تعداد موارد آزمون در فرایند آزمون بیشتر باشد، احتمال تأمین نیازمندی‌های آزمون[۱] بیشتر می‌گردد. اما با استمرار آزمون‌هایِ نگهداری نرم‌افزار روبه توسعه، که اصطلاحاً "آزمون رگرسیون نرم‌افزار" نامیده می‌شود [۱۰]، حجم مخزن آزمون[۲] (مجموعه آزمون) پس از مدتی به‌طور فزاینده‌ای زیاد می‌شود. در نتیجه به‌دلیل محدودیت‌های زمانی و منابع موجود برای آزمون، انجام آزمون جامع اغلب غیرعملی است. از این‌رو فنون گوناگونی برای حل مشکل هزینه زیاد آزمون رگرسیون نرم‌افزار ارائه شده است [۱۶،۱۵،۱۲،۱۱،۵،۴]. از جمله این راهکارها که از اهمیت زیادی برخوردار است، یافتن زیرمجموعه‌ای از موارد آزمون است. این زیرمجموعه باید بتواند به‌طور مؤثر و کارا نرم‌افزار را بیازماید و کماکان همه نیازمندی‌های آزمون را که مجموعه اولیه تأمین می‌کرد، برآورده نماید [۱۳]. یافتن چنین مجموعه‌ای می‌تواند در زمان تولید موارد آزمون، یا بعد از ایجاد مجموعه اولیه از موارد آزمون صورت گیرد. کاهش تعداد موارد آزمون، به‌گونه ای که همه نیازمندی‌های آزمون برآورده شود، سبب افزایش کارایی فرایند آزمون می‌شود. علت این مسئله این است که زمان اجرای موارد آزمون کمتر شده و نهایتاً منجر به کاهش زمان کل آزمون می‌شود. این مسئله به "کاهش یا کمینه‌سازی[۳] مجموعه آزمون" مشهور بوده و زیرمجموعه حاصل، "مجموعه نماینده[۴]" نامیده می‌شود [۲].

مسئله مهم در خصوص کاهش مجموعه آزمون‌ها این است که حذف دائمی تعدادی از موارد آزمون از مجموعه آزمونها، نباید منجربه کاهش غیرقابلِ قبول قدرت کشف خطای زیرمجموعه کاهش یافتهِ حاصل، نسبت به مجموعه آزمون اولیه گردد. با این حال در مطالعات و آزمایش‌هایی که توسط چند تن از محققان این رشته صورت گرفته است [۱]، میزان این کاهش حتی تا ٪۸۰ نیز رسیده است. لزوم توجه به میزان کاهش کشف خطا به این دلیل است که اثر این کاهش به‌تدریج با به‌کارگیری مجدد آن در نسخه‌های بعدی نرم‌افزار بیشتر شده تا جایی‌که حجم خطاهای از دست رفته زیاد شده و فرایند آزمون دچار اختلال و عدم کارایی می‌گردد.

در روشی که در این مقاله برای کاهش مجموعه آزمون ارائه شده است، هرگاه مورد آزمونی برحسب یک معیار پوشش کد نرم‌افزار (معیار پوشش کد در آزمون، تأمین‌کننده کفایت آزمون[۵] است) افزونه باشد، اگر نسبت به معیار پوشش دیگر افزونه نباشد، حذف نمی‌شود و در مجموعه آزمون باقی می‌ماند. در واقع چنین افزونگی که به‌طور کنترل‌شده در مجموعه آزمون بوجود می‌آید، ناشی از موارد آزمونی است، که در کشف خطاها اهمیت زیادی دارند و با حذف آن‌ها کاهش زیادی در قدرت کشف خطای آزمون اتفاق می‌افتد. نتایج آزمایش‌های تجربی که در ادامه مقاله آمده است، نشان می‌دهد که این روش قادر است به‌طور چشمگیری درصد کشف خطا را با کمی‌افزایش در اندازه مجموعه کاهش یافته بهبود بخشد.

ساختار ادامه مقاله به صورت زیر است: بخش ۲ به بیان تعریف دقیق مسئله کاهش موارد آزمون به عنوان یکی از فنون بهبود آزمون رگرسیون نرم‌افزار و نیز بیان مشکل کاهش قدرت کشف خطای مجموعه کاهش‌یافته، ناشی از حذف موارد آزمون می‌پردازد. همچنین در ادامه کارهای پیشین در این بخش، الگوریتم H شرح داده شده است. پس از آن، بخش ۳ شرح روش پیشنهادی با ذکر مثال توضیح داده شده است. در بخش ۴ به مطالعات تجربی و نتایج حاصل از پیاده‌سازی الگوریتم پرداخته شده است. بخش ۵ نیز به نتیجه‌گیری و کارهای آینده اختصاص دارد.

۲- کارهای پیشین

۲-۱- مسئله کاهش مجموعه آزمون

به دلیل این‌که معمولاً یک مورد آزمون، به‌تنهایی نمی‌تواند همه نیازمندی‌های آزمون را تأمین نماید، اغلب مجموعه‌ای از موارد آزمون مورد استفاده قرارمی‌گیرد تا بتوان نیازمندی‌های بیشتری را تأمین نمود.

اما با تولید و درج موارد آزمون جدید در مجموعه آزمون، ممکن است این حجم زیاد آزمونها بیش از حدِ نیاز برای تأمین نیازمندی‌های آزمون باشد. به عبارت دیگر، اگر تعدادی از موارد آزمون از مجموعه آزمون حذف شود، بازهم این مجموعه همه نیازمندی‌های آزمون را که مجموعه اولیه تأمین می‌کرد، برآورده می‌نماید.

نخستین تعریف رسمی از مسئله کاهش مجموعه آزمون در سال ۱۹۹۳ توسط هارولد و گروهش [۲] ارائه گردید:

فرض: یک مجموعه آزمون T از موارد آزمون$\{t_1, t_2,..., t_m\}$ یک مجموعه از نیازمندی‌های آزمون $\{r_1, r_2 ,..., r_n\}$ که باید برآورده شوند، تا پوشش مطلوب از برنامه حاصل شود و زیرمجموعه‌های $\{T_1,T_2 ,...,T_n\}$ از T که هرکدام به یکی از r_iها منتسب است، به‌قسمی‌که هر یک از موارد آزمون t_j متعلق به T_i ، r_i را تأمین نماید.

مسئله: زیر مجموعه کمینه‌ای از T پیدا کنید که همه r_i هایی را که مجموعه غیر کمینه T تأمین می‌کند برآورده نماید.

عموماً کاهش مجموعه آزمون یک مسئله بهینه‌سازی "تک هدفه[۶]" است، اما در برخی بررسی های اخیر [۳،۲] کاهش مجموعه آزمون چند هدفه نیز مورد توجه قرار گرفته است. این اهداف معمولاً شامل پوشش کد، تاریخچه کشف خطا و هزینه اجرایی خواهند بود [۳].

در حالت کلی، مسئله پیدا کردن زیرمجموعه کمینه‌ای از T که قادر به پوشش همه نیازمندی‌های T باشد، یک مسئله NP کامل است [۳]؛ زیرا مسئله پیدا کردن پوشش کمینه مجموعه در زمان نمایی را می‌توان به مسئله کمینه‌سازی مجموعه آزمون کاهش داد. بنابراین استفاده از روشهای ابتکاری برای حل این مسئله اهمیت پیدا می‌کند. یک روش ابتکاری-حریصانه [۴،۳] کلاسیک برای حل مسئله پوشش کمینه مجموعه آزمون به شرح زیر است : مورد آزمونی را انتخاب کنید که بیشترین نیازمندی‌ها را می‌پوشاند، همه نیازمندی‌هایی که توسط مورد آزمون منتخب پوشیده می‌شود را حذف نمایید و این فرایند را تا آنجا که همه نیازمندیها پوشیده شوند، تکرار نمایید. روش ابتکاری دیگر توسط هارولد در [۲] ارائه شده است که روش کار آن به این صورت است که بطور حریصانه موارد آزمونی انتخاب می‌شوند که بیشترین نیازمندی‌هایی را که توسط کمترین تعداد آزمونها برآورده می‌شوند، تأمین نماید.

هدف معیار آزمون، (مثلاً پوشش انشعاب، پوشش مسیر یا پوشش دستورات) تأمین کفایت مجموعه آزمون از لحاظ پوشش و نیز بررسی و کنترل کیفیت آن می‌باشد. با داشتن معیار آزمون C که توسط مجموعه آزمون T تأمین می‌شود، مورد آزمون t در T با توجه به C افزونه است، اگر مجموعه کوچکتر T-{t} نیز C را بپوشاند [۵]. بنابراین، فرایند حذفِ موارد آزمونی که با توجه به یک معیار آزمون افزونه هستند، "کفایت مجموعه را با توجه به آن معیار خاص" حفظ می‌نماید. مطالعات تجربی قبلی [۵،۳] از معیار پوشش کد برای کمینه‌سازی استفاده کرده‌اند. در آزمایش‌هایی که وونگ انجام داده است [۶]، مجموعه آزمون کمینه ۹ تا ۶۸ درصد از نظر اندازه کاهش یافته است و بطور همزمان بین ۰،۱۹ تا ۶،۵۵ درصد فقدان کشف خطا داشته است. از طرف دیگر، در آزمایش‌هایی که رودرمل انجام داد [۱۷]، مجموعه‌های کاهش یافته، بطور میانگین، حدود ۸۰ درصد کاهش یافته‌اند و نیز بطور متوسط حدود ۴۸ درصد فقدان کشف خطا وجود داشته است. این نتایج نشان می‌دهد که هرچه درصد کاهش اندازه مجموعه آزمون بیشتر باشد، درصد فقدان کشف خطا نیز بالاتر می‌رود.

در مطالعات گوناگون انجام گرفته در حوزه کاهش مجموعه آزمون، معیارهای گوناگونی مورد توجه قرار گرفته‌اند که بعضی ریز دانه‌تر[۷] (قوی‌تر) از بقیه هستند [۷]. برای شناسایی موارد آزمونی که عناصر ساختاری و عملیاتی متفاوت برنامه را مورد آزمون قرار می‌دهند، معیارهای آزمون مختلف می‌تواند مفید باشد و می‌توان چندین معیار را آشکار می‌نمایند، به‌کار برد.

۲-۲- الگوریتم H

یکی از روش‌های ابتکاری برای حل مسئله کاهش مجموعه آزمون، روشی است (روش H) که توسط هارولد و گروهش در [۲] ارائه شده است. پیش از بیان روش پیشنهادی، لازم است ابتدا روش H معرفی شده و نشان داده شود که چگونه توسعه و بهبود این روش، ایده روش پیشنهادی را به‌دست می‌دهد.

عملکرد این الگوریتم به این ترتیب است: با داشتن مجموعه آزمون T و نیازمندی‌های آزمون r_1 تا r_n که بایستی پوشیده شوند تا پوشش مطلوب آزمون حاصل شود، الگوریتم H زیر مجموعه های T_1 تا T_n را از T در نظر می‌گیرد به‌طوری‌که همه موارد آزمون موجود در T_i را می‌توان برای برآورده کردن r_i بکار گرفت. در مرحله اول تمام موارد آزمونی که در T_iهای با کاردینالیتی[۸] یک هستند، انتخاب شده و در مجموعه کاهش یافته درج می‌شوند و T_i های حاوی آن‌ها علامت‌گذاری می‌شوند. سپس T_i های با کاردینالیتی دو مورد بررسی قرار می‌گیرند. به‌طور تکراری مورد آزمونی که در بیشترین تعداد T_iهای با کاردینالیتی دو قرار دارد انتخاب شده و به مجموعه کاهش یافته افزوده می‌شود و همه T_iهای حاوی چنین موارد آزمونی علامت‌گذاری می‌شوند. این فرایند برای T_iهای با کاردینالیتی سه، چهار تا max ادامه می‌یابد که max حداکثر کاردینالیتی T_iها است. هرگاه در انتخاب موارد آزمون از T_iهای از کاردینالیتی m تساوی ایجاد شود، مورد آزمونی که در بیشترین تعداد T_i علامت‌نخورده از کاردینالیتی m+۱ باشد، انتخاب می‌گردد. اگر نتوانیم مورد آزمون مناسب را انتخاب نماییم، T_iهای کاردینالیتی بالاتر مورد بررسی قرار می‌گیرند و درنهایت یکی از موارد آزمون به تصادف انتخاب می‌شود.

کاربرد الگوریتم فوق برای کاهش مجموعه آزمون در شکل ۱ به‌صورت زیر است. در ابتدا چون انشعابات B_1^T و B_4^F تنها به‌وسیله موارد آزمون t_1 و t_2 تأمین می‌شوند، هر دوی آن‌ها در مجموعه کمینه درج می‌شوند. سپس همه انشعاباتی که توسط این دو پوشیده می‌شوند، علامت‌گذاری می‌گردند. در نتیجه مورد آزمون t_3 با توجه به معیار پوشش انشعاب افزونه می‌شود؛ زیرا تمام انشعاباتی که می‌پوشاند هم اکنون علامت‌گذاری شده‌اند. در ادامه تنها انتخاب یکی از موارد آزمون t_4 یا t_5 کافیست تا انشعاب باقیمانده B_4^T را برآورده سازد.

فرض می‌کنیم t_4 انتخاب شود. انشعاب B_4^F را علامت‌گذاری می‌نماییم (در نتیجه t_5 با توجه به معیار انشعاب افزونه می‌گردد). چون همه نیازمندی‌های آزمون پوشش یافته است، الگوریتم خاتمه می‌یابد و مجموعه کاهش یافته عبارتست از {t_1, t_2, t_4}. باید توجه داشت که مورد آزمون t_3 که خطای تقسیم بر صفر را در خط ۱۳ آشکار می‌نمود، انتخاب نشد و کارایی مجموعه کمینه در کشف خطا به‌خاطر کمینه‌سازی مجموعه آزمون کاهش یافت.

```
۱:    read(a,b,c,d);
B۱:   if (a > ۰)
۲:      x = ۲;
۳:    else
۴:      x = ۵;
۵:    endif
B۲:   if (b > ۰)
۶:      y = x + ۱;
۷:    endif
B۳:   if (c > ۰)
B۴:     if (d > ۰)
۸:        output(x);
۹:      else
۱۰:       output(۱۰);
۱۱:     endif
۱۲:   else
۱۳:     output(۱/(y-۶));
۱۴:   endif
```

دنباله آزمون T، کافی برای پوشش انشعاب

t۱: (a = ۱,b = ۱, c = -۱, d = ۰)

t۲: (a = -۱,b = -۱, c = ۱, d = -۱)

t۳: (a = -۱,b = ۱, c = -۱, d = ۰)

t۴: (a = -۱,b = ۱, c = ۱, d = ۱)

t۵: (a = -۱,b = -۱, c = ۱, d = ۱)

مورد آزمون	B_1^T	B_1^F	B_2^T	B_2^F	B_3^T	B_3^F	B_4^T	B_4^F
t_1	X		X			X		
t_2		X		X	X			X
t_3		X	X			X		
t_4		X	X		X		X	
t_5		X		X	X		X	

شکل ۱: یک برنامه نمونه با دنباله آزمون کافی در پوشش انشعاب

اکنون با استفاده از مثال فوق، نشان داده می‌شود که روش جدید چگونه بعضی از موارد آزمون را که نسبت به پوشش انشعاب افزونه هستند، در مجموعه کاهش یافته نگه می‌دارد. اطلاعات پوشش تعریف-استفاده برای همه موارد آزمون در جدول ۱ نشان داده شده است. الگوریتم H را به‌گونه‌ای تغییر می‌دهیم، که پس از انتخاب هر مورد آزمون t_i عملیات زیر را انجام دهد: اگر به‌واسطه انتخاب t_i، مورد آزمون دیگری مثل t_j با توجه به معیار پوشش انشعاب افزونه گردد، و نیز اگر t_j با توجه به معیار پوشش زوج تعریف و استفاده افزونه نباشد، t_j نیز در مجموعه کاهش یافته درج می‌گردد. در مثال اخیر پس از انتخاب t_1 و t_2 توسط الگوریتم H ، t_3 با توجه به معیار پوشش انشعاب افزونه می‌گردد. با این وجود، t_3 زوج تعریف و استفاده x(۴،۶) را می‌پوشاند که t_1 و t_2 آن را نمی‌پوشانند. بنابراین، t_3 انتخاب می‌شود. در ادامه، یکی از موارد آزمون t_4 یا t_5 باید انتخاب شود تا B_4^T را بپوشاند. فرض می‌کنیم t_4 انتخاب شود. در این مرحله t_5 با توجه به هر دو معیار پوشش انشعاب و تعریف-استفاده افزونه شده و انتخاب نمی‌شود و چون همه انشعاب‌ها و زوج‌های تعریف-استفاده علامت‌گذاری شده‌اند، الگوریتم خاتمه می‌یابد. مجموعه حاصل، عبارت است از $\{t_1, t_2, t_3, t_4\}$ که خطای تقسیم برصفر را در خط ۱۳ آشکار می‌نماید.

جدول ۱: اطلاعات زوج پوشش تعریف-استفاده

مورد آزمون	x(۲،۶)	x(۴،۶)	x(۴،۸)	y(۶،۱۳)	a(۱,B_1)	b(۱,B_2)	c(۱,B_2)	d(۱,B_4)
t_1	X			X	X	X	X	
t_2					X	X	X	X
t_3		X		X	X	X	X	
t_4		X	X		X	X	X	X
t_5			X		X	X	X	X

اگر الگوریتم H را برای کاهش مجموعه آزمون، با توجه به معیار زوج تعریف-استفاده بکار بگیریم، مجموعه کاهش یافته عبارت است از $\{t_1,t_4\}$. باید توجه داشت که این مجموعه از لحاظ پوشش انشعاب کامل نیست، چون دستورات B_2^F و B_4^F را نمی‌پوشاند و نیز خطای تقسیم بر صفر در خط ۱۳ را آشکار نمی‌نماید. علاوه بر این، اگر اجتماع مجموعه انشعاب‌ها و زوج‌های تعریف-استفاده را در نظر گرفته و الگوریتم H را روی این مجموعه مرکب (اجتماع دو نیازمندی آزمون) اعمال نماییم، مجموعه کاهش یافته $\{t_1, t_2, t_4\}$ خواهد بود، که باز هم خطای تقسیم برصفر در خط ۱۳ آشکار نخواهد شد. نکته مهم این است که کاربرد هر تعیین‌کننده[۹] (دستورات شرطی و حلقه‌ها در زمره تعیین‌کننده‌ها هستند.) در جدول دو نیازمندی آزمون تولید می‌کند، یکی برای خروجی "درست" و دیگری برای خروجی "نادرست".

در کل، مثال ارائه شده نشان می‌دهد که روش پیشنهادی (روش H تغییر یافته)، برای کاهش مجموعه آزمون، که به طور کنترل‌شده بعضی از موارد آزمون را در مجموعه کمینه درج می‌کند، نسبت به روش‌های موجود، که یک معیار را برای کاهش در نظر می‌گیرند، عملکرد بهتری خواهد داشت. در تمام مثال‌های اخیر، که با روش H انجام شده است، t_3 بواسطه انتخاب سایر موارد آزمون افزونه شده و هرگز انتخاب نمی‌شود. اما روش پیشنهادی، t_3 را در مجموعه کاهش یافته قرار داده و موجب می‌شود تا خطایی که توسط سایر موارد آزمون آشکار نمی‌شود، آشکار گردد. علت این امر این است که t_3 ترکیب متفاوتی از خروجی‌های انشعاب و زوج تعریف-استفاده را نسبت به سایر موارد آزمون اجرا می‌نماید و با این حال، مجموعه آزمون اصلی کاهش یافته است.

۳- روش پیشنهادی

در این مقاله یک روش جدید برای کاهش مجموعه آزمونها ارائه شده است که از چندین معیار آزمون استفاده می‌نماید. ایده کلیدی روش ارائه شده، به این‌صورت است که هر بار یک مورد آزمون t که یک نیازمندی آزمون را با توجه به معیار پوشش C_i تأمین می‌کند، برای مجموعه کاهش‌یافته انتخاب می‌شود، عملیات زیر انجام می‌شود: از میان تمام موارد آزمون باقی مانده که به علت انتخاب t و با توجه به معیار C_i افزونه می‌شوند، موارد آزمونی که بیشترین نیازمندی‌ها را با توجه به معیار C_j ($C_j \neq C_i$) دیگر برآورده می‌کنند، در مجموعه کاهش یافته انتخاب و درج می‌نماییم. بنابراین این روش به‌طور کنترل‌شده موارد آزمونی را که با توجه به یک معیار آزمون افزونه بوده، ولی نسبت به معیار دیگر افزونه نیستند، در مجموعه کاهش یافته نگه می‌دارد.

در حقیقت روش پیشنهادی بعد از انتخاب هر مورد آزمون در مجموعه کاهش یافته براساس معیار C_i، از معیارهای دیگری استفاده می‌کند تا موارد آزمون اضافی را انتخاب کند. این موارد آزمون اضافی، نسبت به C_i افزونه هستند، ولی با توجه به سایر معیارها افزونه نیستند. شبه کد روش پیشنهادی، در شکل ۲ مشاهده می‌شود. ورودی این الگوریتم، مجموعه آزمون T و دو مجموعه از دنباله آزمون است که برای هر یک از دو معیار، هر مورد آزمون را به نیازمندی‌هایی که می‌پوشاند، نگاشت می‌دهد. تکرار الگوریتم، با توجه به کاردینالیتی فعلی است که از ۱ شروع شده و به max cardinality ختم می‌شود. برای هر معیار آزمون، نیازمندی‌های علامت‌نخورده که توسط مورد آزمون‌های منتخب تأمین می‌شوند، علامت‌گذاری می‌شوند و مورد آزمون‌هایی که با توجه به معیار اول، افزونه شده‌اند به مجموعه موارد آزمونِ افزونه اضافه می‌گردند. خروجی الگوریتم، مجموعه کاهش یافته‌ای است که از لحاظ پوشش نیازمندی‌های آزمون با مجموعه اصلی برابری می‌کند. در ابتدا، مجموعه کاهش یافتهِ RS خالی است و تمام نیازمندی‌های آزمون به ازای همه معیارها علامت‌نخورده هستند.

گام اول: در این مرحله، مقداردهی‌های اولیه انجام می‌شود. مثلاً برای هر معیار تعداد مجموعه‌هایی که هر مورد آزمون t در آن‌ها وجود دارد، محاسبه می‌شود. پس از آن، الگوریتم کمینه‌سازی H اجرا می‌شود و تا زمانی که در مجموعه T، مورد آزمونی وجود داشته باشد، گام دوم تا چهارم تکرار می‌شوند.

گام دوم: در این مرحله، مورد آزمون بعدی بر اساس معیار اول آزمون، C_1، انتخاب شده و به مجموعه کاهش یافته اضافه می‌شود. این مورد آزمون به‌گونه‌ای انتخاب می‌شود که بیشترین نیازمندی‌های علامت‌نخورده را بپوشاند. پیاده‌سازی این مرحله براساس الگوریتم کمینه‌سازی H انجام شده است. سپس مورد آزمون انتخاب شده در مجموعه کاهش یافته درج شده و نیازمندی‌هایی از آزمون که با انتخاب مورد آزمون اخیر، تأمین می‌شوند، علامت‌گذاری می‌گردند. به‌علاوه مورد آزمون انتخاب شده از مجموعه موارد آزمون حذف می‌شود.

گام سوم: در این مرحله، موارد آزمونی که به‌واسطه انتخاب مورد آزمون کاندید، و با در نظر گرفتن معیار اول افزونه می‌شوند، ذخیره می‌گردند. در ادامه این موارد آزمون نیز از مجموعه کل موارد آزمون حذف می‌شوند.

گام چهارم: این مرحله از موارد آزمونی که با توجه به معیار C_1 افزونه هستند، موارد آزمونی را انتخاب می‌کند. انتخاب در این مرحله بر اساس اطلاعات پوشش موارد آزمون نسبت به معیار دوم، C_2، صورت می‌گیرد. موارد آزمون منتخب بیشترین پوشش اضافی را نسبت به معیار C_2 دارند. در خاتمه این مرحله کنترل به گام دوم برمی‌گردد و این کار تکرار می‌شود، تا جایی‌که همه نیازمندی‌ها علامت‌گذاری شوند.

باید توجه داشت که روش پیشنهادی، مستقل از هر نوع معیار آزمون است و می‌توان آن‌را با معیارهای مختلف جریان کنترلی، جریان داده‌ای و معیارهای آزمون جعبه سفید و جعبه سیاه به‌کار برد. همچنین روش پیشنهادی را می‌توان با هر الگوریتم کمینه‌سازی (مثل H یا حریصانه کلاسیک) که لیستی از موارد آزمون را ایجاد کرده و موارد آزمون را یکی پس از دیگری در مجموعه کاهش یافته درج می‌کند، به‌کار برد.

۴- مطالعات تجربی

مطالعات تجربی انجام شده در این مقاله، مشابه مطالعات سایر محققان [۱] بر روی برنامه‌های زیمنس[۱۰] و Space که به زبان C نوشته شده‌اند، انجام شده است. مجموعه آزمون زیمنس و برنامه Space، به‌همراه موارد آزمون و نسخ خطادار در [۱۸] در دسترس هستند.

مطالعات روی برنامه‌های زیمنس، آزمایش‌های کنترل شده هستند، زیرا خطاها به‌طور دستی در آن‌ها کاشته[۱۱] شده است، و اساساً این دسته از برنامه‌ها واقعی نیستند. اما مطالعه روی برنامه Space، یک مطالعه موردی است، چون این برنامه حاوی خطاهای واقعی است. هر برنامه هدف در مجموعه زیمنس، یک مخزن آزمون دارد که حاوی موارد آزمونی است که برای اهداف مختلف آزمون جعبه سفید و جعبه سیاه تولید شده‌اند. همچنین هر برنامه هدف، مجموعه‌ای از نسخ خطادار دارد و هر نسخه خطادار حاوی یک خطای به‌خصوص است، که در آن کاشته شده است. نسخ خطادار به‌گونه‌ای ایجاد شده‌اند که خطای آن‌ها، توسط حداقل ۳ و حداکثر ۳۵۰ مورد آزمون در مخزن آزمون قابل کشف باشد. در این مطالعه تجربی نیز همانند سایر مطالعات [۱]، دنباله آزمون‌های کافی در پوشش لبه[۱۲] ایجاد شده است. یک دنباله آزمون کافی در پوشش لبه است، اگر پوشش لبه آن، همانند پوشش مخزن آزمون‌ها باشد. معیار پوشش لبه، پوشش انشعاب نیز نامیده می‌شود و روی گراف‌های جریان کنترلی تعریف می‌شود. نتیجه نهایی هر شرط، یک لبه در نظر گرفته می‌شود. همچنین، ورودی هر تابع (علاوه بر main)، نیز یک لبه مجزا در نظر گرفته می‌شود.

برای محاسبه پوشش انشعاب نیز، کد منبع برنامه‌ها تجهیز[۱۳] شده‌اند، تا اطلاعات پوشش انشعاب هریک از موارد آزمون را به‌دست دهند.

در این مطالعه از معیارهای آزمون جعبه سفید مانند پوشش انشعاب[۱۴] و پوشش همه استفاده‌ها[۱۵] به‌ترتیب به‌عنوان معیار اول و دوم کاهش استفاده شده است. معیار پوشش انشعاب، یک معیار مبتنی بر جریان کنترلی[۱۶] است و معیار همه استفاده‌ها، یک معیار مبتنی بر جریان داده‌ای[۱۷] می‌باشد.

```
input : // ورودی
        Existing minimization technique H      //H تکنیک کمینه‌سازی
        A  test suite T      // T دنباله آزمون
        Primary/secondary coverage information for all tests  in T
        اطلاعات پوشش همه موارد آزمون برای معیار اول و دوم//
output://خروجی
        RS : a representative set of tests from T
        مجموعه کاهش یافته، مجموعه نماینده //
algorithm ReduceWithRedundancy
begin
        RS := {}; // مجموعه کاهش یافته در ابتدا تهی است
        start running method H; //اجرای الگوریتم H آغاز می‌شود
        while  T is not empty do // تا زمانیکه مورد آزمون در T وجود دارد
            nextTest :=  the next test selected by H for inclusion in RS
            مورد آزمونی که توسط الگوریتم H انتخاب می‌شود تا در مجموعه کاهش‌یافته قرار گیرد//
            RS := RS ∪ {nextTest}; //مورد آزمون انتخاب شده در مجموعه کاهش یافته قرار می‌گیرد
            T := T – {nextTest};    //مورد آزمون انتخاب شده از مجموعه کل موارد آزمون حذف می‌شود
            redundant  := other tests from T that, given the updated RS, have just
                          become redundant with reference to  the primary coverage criterion;
            سایر موارد آزمونی که با انتخاب nextTest نسبت به معیار پوشش اول افزونه شده‌اند //
            T := T – {redundant}
            while ∃ test t in redundant such that not redundant with reference to secondary
                  criterion do
                تا زمانی‌که مورد آزمونی در redundant وجود دارد که نسبت به معیار پوشش دوم افزونه نیست //
                toAdd := the test in redundant contributing max additional secondary
                         coverage  to RS;
                از بین موارد آزمون افزونه، مورد آزمونی که بیشترین پوشش را نسبت به معیار پوشش دوم دارد،//
                انتخاب می‌شود                                                    //
                RS := RS ∪ toAdd;
                مورد آزمون افزونه که  پوشش بیشتری نسبت به معیار دوم دارد، در مجموعه کاهش یافته قرار می‌گیرد//
            endwhile
            Redundant = {};
        endwhile
        return RS;
end ReduceWithRedundancy
```

شکل ۲: شبه کد روش پیشنهادی برای کاهش دنباله آزمون‌ها

پوشش همه استفاده‌ها برای برنامه‌های C، با ابزار ATAC محاسبه شده است [۱۴]. پیاده‌سازی الگوریتم کاهش نیز با زبان جاوا انجام شده است.

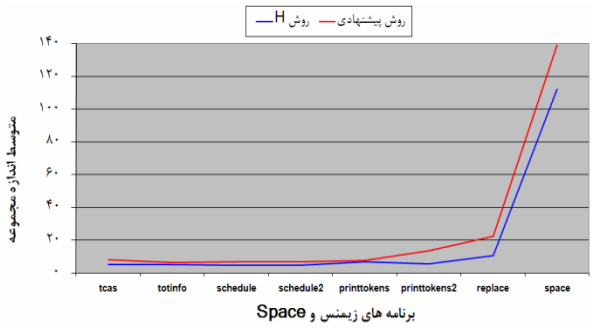
همانند سایر مطالعات تجربی [۱]، ۱۰۰۰ دنباله آزمون کافی در پوشش انشعاب تولید شده است. برای تولید هر دنباله آزمون، ابتدا تعداد ۰٫۵ × LOC مورد آزمون، به‌طور تصادفی از مخزن آزمون انتخاب و در دنباله آزمون، درج شده است (LOC تعداد خطوط کد هر برنامه است). سپس تا مادامی‌که دنباله آزمون کافی در پوشش انشعاب گردد، به‌طور تصادفی مورد آزمون به آن اضافه شده است. با این روش، دنباله‌های آزمون تولید شده، گونه‌های متعددی داشته و در آن‌ها افزونگی بسیاری در تعداد موارد آزمونی که انشعاب‌ها را می‌پوشانند، وجود دارد. برای ارزیابی روش پیشنهادی، روش H نیز پیاده‌سازی شده است و آزمایش‌های مشابهی روی روش H و روش پیشنهادی صورت گرفته است. نتایج این مطالعات در جدول ۲ و شکل ۳ تا ۶ خلاصه شده است. با دقت در شکل ۳ و ۶، مشاهده می‌شود که اندازه مجموعه کاهش‌یافته توسط روش پیشنهادی اندکی بیشتر از روش H است (به‌خاطر ایجاد افزونگی کنترل شده). اما همان‌طور که در شکل ۴ و ۵ دیده می‌شود، متوسط تعداد خطاهای آشکار شده و میانگین درصد فقدان کشف خطا در روش پیشنهادی بالاتر از روش H است. ضمن این‌که شکل ۵ بهبود قابل ملاحظه در میانگین درصد فقدان کشف خطا را نشان می‌دهد. به‌عبارت دیگر، اندکی افزایش در اندازه مجموعه

کاهش‌یافته، باعث می‌شود درصد از دست رفتن خطاها (فقدان کشف آن‌ها) بسیار کاهش یابد.

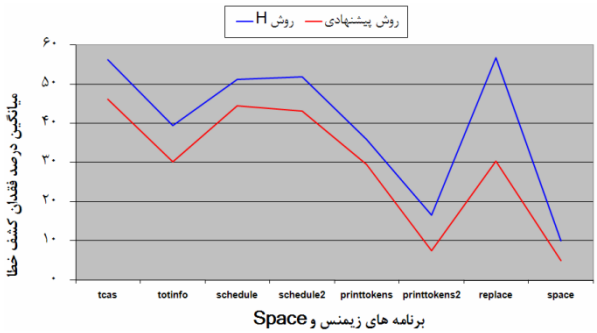
باید توجه داشت که درصد میانگینِ فقدان کشف خطا، برای برنامه Space، به مراتب از مجموعه آزمون زیمنس کمتر است، زیرا موارد آزمون برای برنامه Space، برای رسیدن به پوشش انشعاب، به‌طور تصادفی تولید شده‌اند، در حالی‌که مخزن آزمون، برای مجموعه زیمنس

جدول۲: نتایج مطالعات تجربی برای روش پیشنهادی و روش H روی مجموعه زیمنس و برنامه Space

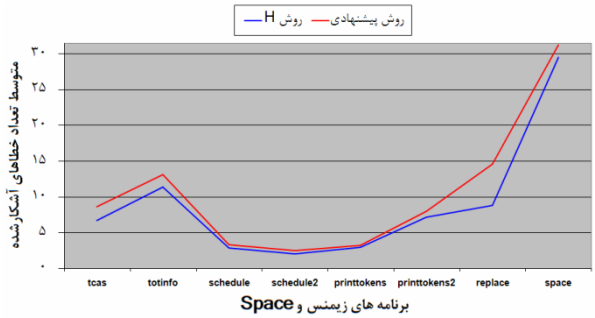
نام برنامه	متوسط اندازه مجموعه آزمون	متوسط تعداد خطاهای آشکارشده	متوسط اندازه مجموعه کاهش‌یافته		متوسط تعداد خطاهای آشکارشده توسط مجموعه کاهش‌یافته		متوسط درصد کاهش		میانگین درصد فقدان کشف خطا	
			روش H	روش پیشنهادی	روش H	روش پیشنهادی	روش H	روش پیشنهادی	روش H	روش پیشنهادی
tcas	۳۵,۶۳	۱۷,۷۶	۵,۰۰	۷,۹۱	۶,۶۷	۸,۵۹	۷۶,۷۷	۶۷,۵۷	۵۶,۲۳	۴۶,۱۳
totinfo	۸۷,۷۳	۱۹,۱۶	۵,۰۲	۶,۴۶	۱۱,۳۴	۱۳,۱۵	۸۸,۹۶	۸۶,۶۲	۳۹,۴۲	۳۰,۱۵
schedule	۷۴,۹۴	۵,۹۶	۴,۷۴	۶,۶۱	۲,۸۸	۳,۲۷	۸۷,۹۱	۸۴,۵۱	۵۱,۱۸	۴۴,۴۶
schedule۲	۷۶,۳۴	۴,۷۳	۴,۷۴	۶,۷۱	۲,۰۲	۲,۴۴	۸۸,۴۵	۸۵,۳۶	۵۱,۸۷	۴۳,۱۵
printtokens	۱۰۱,۸۷	۴,۷۵	۶,۵۸	۷,۷۳	۲,۸۹	۳,۲۲	۸۸,۷۷	۸۷,۳۸	۳۶,۰۲	۲۹,۴۶
printtokens۲	۱۲۱,۷۳	۸,۶۰	۵,۴۹	۱۳,۵۱	۷,۱۳	۷,۹۴	۹۰,۱۹	۸۰,۷۱	۱۶,۷۲	۷,۵۲
replace	۱۳۴,۵۹	۲۱,۴۳	۱۰,۶۶	۲۲,۳۹	۸,۷۷	۱۴,۵۳	۸۶,۷۰	۷۶,۱۰	۵۶,۷۷	۳۰,۳۸
space	۱۵۵۹,۳۱	۳۲,۹۳	۱۱۲,۰۹	۱۳۸,۸۴	۲۹,۵۵	۳۱,۲۶	۸۶,۱۶	۸۳,۸۹	۱۰,۱۳	۵,۰۰



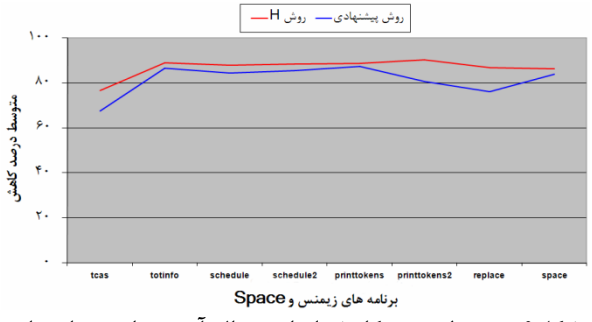
شکل ۳: متوسط اندازه مجموعه کاهش‌یافته برای هر برنامه



شکل ۵: میانگین درصد فقدان کشف خطا توسط دنباله آزمون‌های کاهش‌یافته



شکل ۴: متوسط تعداد خطاهای آشکار شده توسط دنباله آزمون‌های کاهش‌یافته



شکل۶: متوسط درصد کاهش اندازه دنباله آزمون‌ها در برنامه‌های محک

به‌گونه‌ای تولید شده که طیف وسیعی از نیازمندی‌های آزمون جعبه سفید و جعبه سیاه را بپوشاند. بنابراین، حذف یک مورد آزمون از مجموعه زیمنس، با احتمال بیشتری قابلیت کشف خطای مجموعه را کاهش می‌دهد.

۵- نتیجه‌گیری و کارهای آینده

در این مقاله روش جدیدی برای کاهش مجموعه آزمون‌ها، مبتنی بر افزونگی موارد آزمون ارائه شد، که هدف آن افزایشِ قابل ملاحظه قدرت کشف خطای مجموعه آزمون در اثر این افزونگی کنترل‌شده است. به عنوان کارهای آتی می‌توان به‌دنبال انواع دیگر نیازمندی‌های پوششی بود، که برای هر مورد آزمون قابل محاسبه است و با این‌کار، مجموعه بزرگتری از معیارهای پوششی جهت استفاده در افزونگی کنترل‌شده و آزمایشات تجربی متناظر با آن‌ها خواهیم داشت.

در مطالعات تجربی آینده می توان از مجموعه‌های مختلفی از نسخ خطادار (تعداد مختلف نسخه‌های خطادار، انواع مختلف خطاهای کاشته شده و سطوح مختلف دشواری در کشف خطا) به منظور درک بهتر تأثیر نسخه‌های خطادار بر نتایج تجربیِ کاهش مجموعه‌ها استفاده نمود. علاوه براین، می‌توان آزمایشات تجربی بیشتری ترتیب داد و روش جدید را روی برنامه‌های بزرگتر با خطاهای واقعی آزمود.

هدف غایی در مقوله کاهش مجموعه آزمون، کاهش قابل ملاحظه اندازه یک مجموعه است، به‌طوری‌که هیچ تأثیری در میزان کشف خطا نداشته باشد یا میزان تأثیر آن ناچیز باشد. اما این‌که آیا در حالت کلی می توان به این هدف رسید یا خیر هنوز به صورت سؤال باقی است ونیاز به کار بیشتر دارد.

مراجع

[۱] Rothermel, G., Harrold, M. J., Ostrin, J., Hong, C., *"An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites,"* Proc. Int'l Conf. Software Maintenance, pp. ۳۴-۴۳, Nov. ۱۹۹۸.

[۲] Harrold, M. J., Gupta, R., Soffa, M. L., *"A Methodology for Controlling the Size of a Test Suite,"* ACM Trans. Software Eng. And Methodology, vol. ۲, no. ۳, pp. ۲۷۰-۲۸۵, July ۱۹۹۳.

[۳] Zhong, H., Zhang, L., Mei, M., *"An experimental study of four typical test suite reduction techniques,"* Inf. Softw. Technol., vol. ۵۰, pp. ۵۳۴-۵۴۶, ۲۰۰۸.

[٤] Chen, T. Y., Lau, M. F., *"Heuristics toward the Optimization of the Size of a Test Suite,"* Proc. ۳rd Int'l Conf. on Softw. Quality Management. Vol. ۲, ۴۱۵-۴۲۴, Seville, Spain, April ۱۹۹۵.

[٥] Jones, J. A., Harrold, M. J., *"Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage,"* IEEE Transactions on Software Engineering. ۲۹(۳):۱۹۵-۲۰۹, March ۲۰۰۳.

[٦] Offutt, A. J., *"Investigations of the Software Testing Coupling Effect,"* ACM Trans. on Softw. Eng. Methodology. ۱(۱):۳-۱۸, January ۱۹۹۲.

[۷] DeMillo, R. A., Mathur, A. P., *"On the Use of Software Artifacts to Evaluate the Effectiveness of Mutation Analysis for Detecting Errors in Production Software,"* Technical Report SERC-TR-۹۲-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, August ۱۹, ۱۹۹۴.

[۸] Wong, W. E., Horgan, J. R., London, S., Mathur, A. P., *"Effect of Test Set Minimization on Fault Detection Effectiveness,"* Software Practice and Experience, vol. ۲۸, no. ۴, pp. ۳۴۷-۳۶۹, Apr. ۱۹۹۸.

[۹] Chen, T. Y., Lau, M., *"new heuristic for test suite reduction,"* Information and Software Technology ۴۰ (۵-۶),۱۹۹۸.

[۱۰] Rothermel, G., Harrold, M., *A Safe, Efficient Regression Test Selection Technique.* ACM Transactions on Software Engineering & Methodology, ۶(۲), pp. ۱۷۳-۲۱۰, ۱۹۹۷.

[۱۱] Rothermel, G., Harrold, M. J., von Ronne, J., Hong, C., *"Empirical Studies of Test-Suite Reduction,"* Journal of Software Testing, Verification, and Reliability. ۱۲(۴):۲۱۹-۲۴۹, ۲۰۰۲.

[۱۲] Rothermel, G., Elbaum, S., Malishevsky, A. G., Kallakuri, P., Qiu, X., *"On Test Suite Composition and Cost-effective Regression Testing,"* ACM Trans. Softw. Eng. Methodol., vol. ۱۳, pp. ۲۷۷-۳۳۱, ۲۰۰۴.

[۱۳] Chen, Z., Xu, B., Zhang, X., Nie, C., *"A novel approach for test suite reduction based on requirement relation contraction,"* in Proceedings of the ۲۰۰۸ ACM symposium on Applied computing Fortaleza, Ceara, Brazil: ACM, ۲۰۰۸.

[۱٤] Horgan, J. R., London, S. A., *"ATAC: A Data Flow Coverage Testing Tool for C,"* Proceedings of Symposium on Assessment of Quality Software Development Tools. ۲-۱۰, New Orleans, Louisiana, May ۱۹۹۲.

[۱٥] McMaster, S., Memon, A., *"Call-Stack Coverage for GUI Test Suite Reduction,"* IEEE Trans. Softw. Eng., vol. ۳۴, pp. ۹۹-۱۱۵, ۲۰۰۸.

[۱٦] Tallam, S., Gupta, N., *"A concept analysis inspired greedy algorithm for test suite minimization,"* in Proceedings of the ۶th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering Lisbon, Portugal: ACM, ۲۰۰۵.

[۱۷] http://paul.rutgers.edu/~rhoads/Code/easter.c.

[۱۸] Rothermel, G., Elbaum, S., Kinneer, A., Do, H., Software-artifact infrastructure repository. http://www.cse.unl.edu/~galileo/sir.

---

[۱] معمولاً پیش از آن‌که نرم‌افزار مورد آزمون قرار گیرد، اهداف آزمایش تعریف می‌شوند تا بر این اساس، میزان کارایی آزمون در بررسی اجزای موجود در نرم‌افزار تعیین شود[۹]. بعنوان مثال، پوشش کلیه دستورات شرطی برنامه، می‌تواند یکی از اهداف آزمون باشد. این اهداف ممکن است، تفاوت‌های زیادی به‌ویژه در میزان دانه بندی اجزاءِ مورد بررسی با هم داشته باشند. اهداف آزمون را می‌توان با دانه بندی ریز، نظیر پوشش دستورات یا دانه بندی درشت، نظیر پوشش نیازمندی‌های کاربر برای سیستم نرم‌افزاری تعریف نمود. معمولاً هر نیازمندیِ کاربر، به‌صورت چندین ماژول پیاده‌سازی می‌شود که از دهها و یا حتی صدها خط کد تشکیل می‌شود. در حوزه کاهش مجموعه آزمون‌ها، از اهداف آزمون به نام "نیازمندی‌های آزمون" نیز یاد می‌شود.

[۲] test pool

[۳] minimization
[٤] representative set
[٥] test adequacy
[٦] single objective
[۷] fine-granularity
[۸] تعداد اعضای مجموعه
[۹] predicate
[۱۰] Siemens suite
[۱۱] seed
[۱۲] edge coverage
[۱۳] instrument
[۱٤] branch coverage
[۱٥] all-uses
[۱٦] control-flow based
[۱۷] data-flow based