یازدهمین سمینار بین‌المللی
جبر خطی و کاربردهای آن
بهمن ۱۴۰۰

موازی سازی الگوریتم انتخاب موارد آزمون رگرسیون سیستم های نرم افزاری با استفاده از تجزیه مقادیر منفرد

مهدی موحدیان مقدم[۱] * و دکتر محمود فضلعلی[۱] †

[۱]گروه علوم کامپیوتر، دانشکده ریاضی، دانشگاه شهید بهشتی، تهران، ایران

چکیده

هنگام توسعه یک سیستم نرم افزاری تغییر در یک قسمت سیستم ممکن است منجر به ایجاد تغییرات ناخواسته در بخش های دیگر سیستم شود. این بخش های متاثر ممکن است عملکرد سیستم را دچار اختلال کند، بنابراین از آزمون رگرسیون جهت مقابله با این اختلالات استفاده می شود. این آزمون درصدد سنجش مجدد این بخش ها برآمده تا از این اختلالات جلوگیری کند، اما تشخیص این بخش ها جهت آزمون مجدد کار مشکلی است. تلاش ما بر این است که به کمک خوشه بندی تغییرات سیستم نرم افزاری خود براساس توابع سازنده سیستم به کمک تجزیه مقادیر منفرد ، بتوان برای تشخیص این بخش ها در طی یک تغییر جدید، جهت انجام مجدد آزمون، استفاده نمود. به جهت افزایش کارایی، محاسبات ما به صورت موازی روی سیستم های حافظه مشترک صورت گرفت تا با افزایش مقیاس سیستم های نرم افزاری ، بتوان پاسخی بهینه دریافت نمود.

واژه‌های کلیدی: تجزیه مقادیر منفرد ، آزمون نرم افزاری رگرسیون ، کاهش موارد آزمون موازی ، خوشه بندی موازی موارد آزمون

کد موضوع‌بندی ریاضی [۲۰۱۰]: 62P30

۱ مقدمه

امروزه در توسعه سیستم های نرم افزاری پس از یک دوره توسعه، ممکن است نیاز به ایجاد تغییرات در بخش های توسعه داده شده قبلی باشد. این تغییرات ممکن است موجب تاثیر بر قسمت های دیگر نرم افزار شود. این تاثیرات ممکن است موجب ایجاد اختلال جدیدی در عملکرد سیستم شود. جهت جلوگیری از این اختلالات ناخواسته نوعی از انواع آزمون های نرم افزاری به نام آزمون رگرسیون [۱] به کار گرفته می شود. تست رگرسیون از موارد آزمون [۲] موجود برای بررسی اینکه آیا تغییرات ایجاد شده خطای جدیدی ایجاد نکرده و خطاهای جانبی ناخواسته ای نداشته است، استفاده می کند. به عبارت دیگر، هدف این است که اطمینان حاصل شود که بخش‌هایی از یک سیستم که تغییر نکرده‌اند، همچنان مانند قبل از انجام تغییرات کار می‌کنند [۵]. آزمون رگرسیون درصدد اجرای مجدد مجموعه موارد آزمون برخواهد آمد که بتوان پس از اجرای موفقیت آمیز همه آن ها به طور قطعی نسبت به صحت عملکرد سیستم نظر داد. به دست آوردن این مجموعه موارد آزمون کار چندان آسانی نخواهد بود طوری که از روش های برنامه ریزی خطی صحیح نیز امروزه در جهت یافتن این مجموعه موارد آزمون بهره گرفته شده است [۲]. ما بر این تلاش هستیم تا این مجموعه موارد آزمون را به کمک خوشه بندی توسط تجزیه مقادیر منفرد [۳] بر روی ماتریسی از توابع [۴] به کار رفته در سیستم که از نظر تغییرات زمانی به یکدیگر نزدیک هستند

*سخنران. آدرس ایمیل: m.movahedian@mail.sbu.ac.ir
†سخنران. آدرس ایمیل: fazlali@sbu.ac.ir

[۱] Regression Test
[۲] Test cases
[۳] Singular value decomposition
[۴] Functions

به دست آوریم. این اطلاعات به کمک ابزارهای کنترل منبع [۱] که امروزه در اکثر شرکت های تولید کننده سیستم های نرم افزاری به کار گرفته می شود، به سادگی به دست خواهد آمد. به دلیل وجود حجم بالای توابع در سیستم های نرم افزاری تجاری ما در طی این خوشه بندی از یک رویکرد موازی [۲] استفاده نمودیم تا به این طریق امکان استفاده از این رویکرد در سیستم های تجاری بزرگ نیز در کوتاه ترین زمان ممکن، فراهم شود.
این پژوهش در بخش دوم به معرفی روش های مورد استفاده می پردازد و در بخش سوم از این روش ها جهت انجام خوشه بندی اطلاعات و در پی آن استخراج این داده ها از یک سیستم نرم افزاری به صورت موازی و یافتن مجموعه مواردی که باید به ازای یک بردار تغییر خاص مورد آزمون مجدد قرار بگیرند استفاده شده است، در بخش چهارم نتایج تجربی این پژوهش و بخش پنجم جهت نتیجه گیری نهایی تدوین شده است.

۲ کارهای مرتبط

در این بخش به کارهای مرتبط انجام شده در حوزه انتخاب موارد آزمون رگرسیون و همچنین خوشه بندی کد منبع [۳] و تجزیه مقادیر منفرد پرداخته می شود.

۱.۲ انتخاب موارد آزمون رگرسیون

تست رگرسیون فرآیند تست مجدد نرم افزار است ، که اصلاح شده می باشد. سیستم‌های بزرگ معمولاً مجموعه‌های تست رگرسیون بزرگی دارند. در یک سیستم نرم‌افزاری، تغییرات کوچک در یک بخش از یک سیستم اغلب باعث ایجاد مشکلاتی در بخش‌های دوردست سیستم می‌شود. برای یافتن این نوع مشکلات از تست رگرسیون استفاده می شود [۱]. هدف از تکنیک‌های انتخاب آزمون رگرسیون ، جداسازی تست‌هایی است که به صورت احتمالی پس از اصلاح سیستم، خطاهای ایجاد شده را کشف می‌کنند. ساده‌ترین تکنیک انتخاب آزمون رگرسیون، استراتژی همه‌آزمایی مجدد است که در آن کل مجموعه موارد آزمایشی مورد بررسی قرار می گیرند [۴].

۲.۲ خوشه بندی کد منبع

امروزه خوشه بندی کد منبع به جهت کاربرد آن در سیستم های تشخیص سرقت ادبی اهمیت بخصوصی پیدا کرده است،از جمله این موارد می توان تلاش (Duracik و همکاران) را بدین صورت ذکر کرد که سعی بر خوشه بندی خط به خط کد منبع یک سیستم نرم افزاری، براساس الگوریتم خوشه بندی k-means داشته اند، که سپس با در پیش گرفتن یک رویکرد افزایشی جهت این خوشه بندی، عمل جستجو را به صورت برداری بر روی کد منبع به سرانجام رسانده اند [۳]. در این پژوهش، ما خوشه بندی تغییرات کد منبع را براساس تغییرات توابع توسعه داده ایم.

۳.۲ تجزیه مقادیر منفرد موازی

در سال های اخیر موازی سازی های متعددی بر روی الگوریتم تجزیه مقادیر منفرد صورت گرفته است، در تلاش (Xiao و همکاران) شاهد موازی سازی الگوریتم مذکور براساس موازی سازی بلوکی ژاکوبی بوده ایم [۶]. ما در این پژوهش علاوه بر موازی سازی الگوریتم تجزیه مقادیر منفرد خویش، استخراج اطلاعات خوشه های خود را به صورت موازی توسعه دادیم. این توسعه موازی برای استفاده توسط زیرساخت های شرکت های نرم افزاری تجاری بسیار مناسب تر می باشد، زیرا شرکت های تجاری بزرگ که دارای زیرساخت های سخت افزاری قوی تری می باشند عموما در حال توسعه سیستم های نرم افزاری هستند که به این نوع آزمون بسیار نیازمندند.

۳ انتخاب موازی موارد آزمون رگرسیون براساس تجزیه مقادیر منفرد

این پژوهش الگوریتمی را ارائه می کند که براساس تغییرات توابع در یک کد منبع سیستم نرم افزاری، بتوان مجموعه توابعی که روی یکدیگر تاثیر بیشتری دارند را شناسایی کرد و به کمک تجزیه مقادیر منفرد، به نوعی خوشه بندی دست یافت که از طریق این خوشه

[۱] Source control
[۲] Parallel
[۳] Source code

بندی بتوان مجموعه خلاصه تری از مجموعه موارد آزمون رگرسیون را تولید نمود. تمامی الگوریتم عنوان شده به صورت موازی روی سیستم های حافظه مشترک جهت افزایش کارایی توسعه داده شده است.الگوریتم فوق ۱.۳ به این شرح می باشد.

الگوریتم ۱.۳. انتخاب موازی موارد آزمون رگرسیون براساس تجزیه مقادیر منفرد

۱. ایجاد ماتریس F براساس تغییرات توابع نسبت به یکدیگر

۲. ایجاد ماتریس T براساس هر مورد آزمون نسبت به توابع مربوطه به خود

۳. ایجاد ماتریس R جهت نگهداری خوشه ها

۴. ایجاد بردار X جهت نگهداری توابع تغییر یافته

۵. $[U, S, V] = PSVD(F)$.

۶. برای $i = ۱, \ldots, size(U)$ به صورت موازی انجام دهید:

۷. برای $j = ۱, \ldots, size(U)$ به صورت موازی انجام دهید:

۸. اگر $|U[j, i]| > threshold$ انجام دهید:

۹. شروع ناحیه بحرانی

۱۰. افزودن داده های مثبت خوشه $U[i]$ به ماتریس R

۱۱. افزودن داده های منفی خوشه $U[i]$ به ماتریس R

۱۲. پایان ناحیه بحرانی

۱۳. پایان حلقه

۱۴. پایان حلقه

۱۵. $P = T * R$.

۱۶. $Y = R * P^T$.

۱۷. $S = Y * X$.

۱.۳ داده های اولیه

در سیستم های کنترل منبع، تمامی تغییرات کد منبع یک سیستم نرم افزاری در حین توسعه توسط سیستم نرم افزاری، ثبت می شود. این تغییرات در قالب یک تاریخچه در سطوح متفاوتی از قبیل فایل ها و توابع و یا حتی خطوط تغییر یافته، قابل رویت می باشد. ما این تغییرات را در سطح توابع رصد نموده و اجرای موارد آزمون سیستم را به توابع محدود نمودیم.این کار به دو دلیل صورت گرفت که مورد اول را می توان اینگونه بیان کرد که اگر در سطح فایل تغییرات را بررسی کنیم ممکن است به دلیل وجود خطوط زیادی از کدهای منبع موجود در هر فایل، دقت لازم حاصل نشود و دلیل دوم اینکه تغییرات کد منبع را در سطح خطوط تغییر یافته محدود نساختیم زیرا ابعاد ماتریس های تغییرات ما به صورت سرسام آوری افزایش می یافت و باعث تنکی زیاد آن ها در مسئله می شد. به دلایل ذکر شده تصمیم بر این شد که تغییرات کد منبع در سطح تغییرات توابع مورد بررسی قرار گیرد.

جهت آماده سازی داده های مورد نیاز، یک ماتریس مربعی از مرتبه تعداد توابع مورد بررسی که هر درایه آن نشان دهنده وجود تغییرات دو تابع با یکدیگر می باشد را از طریق این سیستم کنترل منبع می سازیم. این ماتریس، F نامیده می شود. این ارتباط به صورت دوطرفه می باشد یعنی تغییرات تابع i و j ام اگر صورت گرفته باشد هر دو درایه مرتبط با F(i,j) و F(j,i) باید مقداردهی شوند، پس ماتریس F حالت متقارن دارد. این ماتریس دارای ابعاد M * M خواهد بود که M به عنوان تعداد توابع مورد بررسی می باشد.

ماتریس دیگری که مورد نیاز است ماتریس T بوده که هر درایه آن نشان گر ارتباط بین هر مورد آزمون با توابع آن سیستم نرم افزاری می باشد. این ماتریس لزوما متقارن نیست چون لزوما تعداد موارد آزمون با تعداد توابع یکسان نخواهد بود و ممکن است یک مورد آزمون با تعدادی تابع ارتباط داشته باشد یا بالعکس.این ماتریس نیز دارای ابعاد TC * M خواهد بود که TC بیان گر تعداد موارد آزمون می باشد.

۲.۳ خوشه بندی موازی توسط تجزیه مقادیر منفرد

همانگونه که ذکر شد، ماتریس F به دلیل متقارن و مثبت معین بودن دارای تجزیه مقادیر منفردی با مقادیر U و V یکسان می باشد. پس از اجرای تجزیه مقادیر منفرد بر روی ماتریس F خروجی مطلوب سه ماتریس $V،U،\Sigma$ است.

$$F = \begin{pmatrix} ۳ & ۱ & ۳ & ۴ & ۲ \\ ۱ & ۰ & ۰ & ۴ & ۰ \\ ۳ & ۰ & ۱ & ۱ & ۰ \\ ۴ & ۴ & ۱ & ۲ & ۰ \\ ۲ & ۰ & ۰ & ۰ & ۱ \end{pmatrix}, T = \begin{pmatrix} ۱ & ۰ & ۰ & ۰ & ۰ \\ ۰ & ۱ & ۰ & ۰ & ۰ \\ ۰ & ۰ & ۱ & ۰ & ۰ \\ ۰ & ۰ & ۱ & ۱ & ۰ \\ ۰ & ۰ & ۱ & ۱ & ۱ \\ ۰ & ۰ & ۰ & ۱ & ۱ \end{pmatrix} \quad (۱)$$

پس از تجزیه مقادیر منفرد و اجرای موازی دو حلقه موجود در الگوریتم جهت تولید خوشه های مد نظر به ماتریسی دست خواهیم یافت که در ابعاد M * C و بدین صورت خواهد بود. جهت جلوگیری از نوشتن همزمان مقادیر توسط دو واحد اجرایی موازی(مانند نخ یا فرآیند) از نواحی بحرانی استفاده نمودیم تا از نوشته شدن مقادیر نامعتبر در یک درایه از ماتریس R جلوگیری شود.

$$R = \begin{pmatrix} -۰/۶۴ & ۰/۴ & ۰/۰۰ & ۰/۰۰ & -۰/۳۶ & ۰/۰۰ & -۰/۵۳ & ۰/۰۰ & ۰/۰۰ \\ -۰/۳۳ & ۰/۵۹ & ۰/۰۰ & ۰/۵۳ & ۰/۰۰ & ۰/۴۷ & ۰/۰۰ & ۰/۱۵ & ۰/۰۰ \\ -۰/۳۱ & ۰/۰۰ & -۰/۱۱ & ۰/۰۰ & -۰/۴۱ & ۰/۵۸ & ۰/۰۰ & ۰/۰۰ & ۰/۶ \\ -۰/۵۹ & ۰/۰۰ & -۰/۶۶ & ۰/۴۴ & ۰/۰۰ & ۰/۰۰ & -۰/۰۹ & ۰/۰۳ & ۰/۰۰ \\ -۰/۱۵ & ۰/۰۰ & -۰/۱۶ & ۰/۰۰ & -۰/۴۵ & ۰/۳۶ & ۰/۰۰ & ۰/۷۷ & ۰/۰۰ \end{pmatrix} \quad (۲)$$

هر ستون این ماتریس یا به عبارت دیگر هر خوشه از این مجموعه بیان گر توابعی هستند که نسبت به هم وابستگی تغییرات بیشتری داشته اند. تنظیم این حد از وابستگی بین توابع به تنظیم مقدار آستانه در الگوریتم وابسته است و به نوعی با آن رابطه مستقیم دارد، طوری که هر چقدر آستانه را افزایش دهیم خوشه هایی خواهیم داشت که دارای عناصری با وابستگی های بیشتر هستند.

۳.۳ تولید مجموعه کاهش یافته موارد آزمون

پس از به دست آمدن ماتریس R به کمک ماتریس T باید نوعی ارتباط میان موارد آزمون رگرسیون و این خوشه های حاصل، ایجاد نمود. پس به کمک ترانهاده کردن ضرب آن دو ماتریس به ماتریسی دست می یابیم که به صورت خوشه بندی شده موارد آزمونی که به ازای هر تغییر تابع باید مورد بررسی قرار گیرند، را نشان خواهد داد. این ماتریس حاصل را Y می نامیم.

$$Y = R * (T * R)^T = \begin{pmatrix} -۰/۶۴ & ۰/۴ & ۰/۰۰ & ۰/۰۰ & -۰/۳۶ & ۰/۰۰ & -۰/۵۳ & ۰/۰۰ & ۰/۰۰ \\ -۰/۳۳ & ۰/۵۹ & ۰/۰۰ & ۰/۵۳ & ۰/۰۰ & ۰/۴۷ & ۰/۰۰ & ۰/۱۵ & ۰/۰۰ \\ -۰/۳۱ & ۰/۰۰ & -۰/۱۱ & ۰/۰۰ & -۰/۴۱ & ۰/۵۸ & ۰/۰۰ & ۰/۰۰ & ۰/۶ \\ -۰/۵۹ & ۰/۰۰ & -۰/۶۶ & ۰/۴۴ & ۰/۰۰ & ۰/۰۰ & -۰/۰۹ & ۰/۰۳ & ۰/۰۰ \\ -۰/۱۵ & ۰/۰۰ & -۰/۱۶ & ۰/۰۰ & -۰/۴۵ & ۰/۳۶ & ۰/۰۰ & ۰/۷۷ & ۰/۰۰ \end{pmatrix} *$$

$$\begin{pmatrix} -۰/۶۴ & -۰/۳۳ & -۰/۳۱ & -۰/۹۰ & -۱/۰۵ & -۱/۰۵ \\ ۰/۴۰ & ۰/۵۹ & ۰/۰۰ & ۰/۰۰ & ۰/۰۰ & ۰/۰۰ \\ ۰/۰۰ & ۰/۰۰ & -۰/۱۱ & -۰/۷۷ & -۰/۹۳ & -۰/۹۳ \\ ۰/۰۰ & ۰/۵۳ & ۰/۰۰ & ۰/۴۴ & ۰/۴۴ & ۰/۴۴ \\ -۰/۳۶ & ۰/۰۰ & -۰/۴۱ & -۰/۴۱ & -۰/۸۶ & -۰/۸۶ \\ ۰/۰۰ & ۰/۴۷ & ۰/۵۸ & ۰/۵۸ & ۰/۹۴ & ۰/۹۴ \\ -۰/۵۳ & ۰/۰۰ & ۰/۰۰ & -۰/۰۹ & -۰/۰۹ & -۰/۰۹ \\ ۰/۰۰ & ۰/۱۵ & ۰/۰۰ & ۰/۰۳ & ۰/۰۸ & ۰/۰۸ \\ ۰/۰۰ & ۰/۰۰ & ۰/۶۰ & ۰/۶۰ & ۰/۶۰ & ۰/۶۰ \end{pmatrix} = \begin{pmatrix} ۰/۹۸ & ۰/۴۴ & ۰/۳۴ & ۰/۷۷ & ۱/۰۲ & ۱/۰۲ \\ ۰/۴۴ & ۰/۹۸ & ۰/۳۷ & ۰/۸۰ & ۱/۱۴ & ۱/۱۴ \\ ۰/۳۴ & ۰/۳۷ & ۰/۹۷ & ۱/۲۲ & ۱/۶۸ & ۱/۶۸ \\ ۰/۴۲ & ۰/۴۳ & ۰/۲۵ & ۱/۲۴ & ۱/۴۵ & ۱/۴۵ \\ ۰/۲۵ & ۰/۳۳ & ۰/۴۵ & ۰/۶۷ & ۱/۶۴ & ۱/۶۴ \end{pmatrix} \quad (۳)$$

بنابراین به ازای هر بردار تغییر X دارای ابعادی به صورت M * 1 که فقط نشان دهنده توابع تغییر یافته می باشد. با اعمال عمل ضرب در ترانهاده ماتریس Y به دست آمده، می توان مجموعه موارد آزمونی که باید مورد بررسی مجدد در این نسخه از نرم افزار قرار

گیرند مشخص شود.

$$X = \begin{pmatrix} ۰ \\ ۱ \\ ۱ \\ ۰ \\ ۱ \end{pmatrix}, Y^T * X = \begin{pmatrix} ۱٫۰۵ \\ ۱٫۶۹ \\ ۱٫۸۰ \\ ۲٫۷۱ \\ ۴٫۴۷ \\ ۴٫۴۷ \end{pmatrix} \quad (۴)$$

بردار حاصل نشان دهنده مجموعه موارد آزمونی است که باید مجدد در ازای این تغییرات مورد بررسی قرار گیرند. این نتیجه نشان دهنده آن است که موارد آزمون ۵ و ۶ دارای اولویت بالاتری جهت اجرای مجدد نسبت به اجرای موارد آزمون دیگر می باشد.

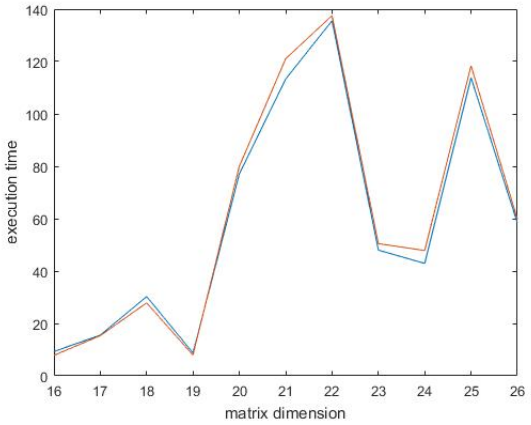
۴ نتایج تجربی

در این بخش بررسی نمودیم چنانچه به جای تغییرات فایل های یک سیستم نرم افزاری، جهت افزایش دقت، تغییرات توابع سیستم را بررسی نماییم با چه چالش هایی رو به رو خواهیم بود.
در بررسی تغییرات توابع با یکدیگر با ابعاد ماتریس های ورودی خیلی بزرگتری رو به رو خواهیم بود ، زیرا تعداد توابع یک سیستم بسیار زیادتر از تعداد فایل های آن سیستم است،این رویکرد برای توسعه دهندگان یک سیستم نرم افزاری چالش محاسباتی ایجاد می کند. بدین جهت رویکردی موازی برای الگوریتم فوق به جای رویکردی ترتیبی ارائه دادیم که در این بخش سعی بر مقایسه آن دو و تحلیل نقاطی داریم که برای اجرای موازی الگوریتم مناسب تر می باشد.
جهت انجام آزمایش خود، از یک سیستم چند پردازنده حافظه مشترک با ۴ هسته پردازشی نسل سوم با سرعت ساعت 2.9 گیگاهرتز

جدول ۱: مقایسه زمان اجرای الگوریتم به صورت موازی و ترتیبی بر حسب ثانیه.

بعد	۱۶	۱۷	۱۸	۱۹	۲۰	۲۱	۲۲	۲۳	۲۴	۲۵	۲۶
موازی	9.27	15.55	30.25	8.71	77.08	113.32	135.53	48.00	42.91	113.80	58.32
ترتیبی	7.81	15.35	27.84	7.88	79.98	121.077	137.53	50.45	47.84	118.35	59.77

ساخته شرکت اینتل، به همراه پیاده سازی دو نسخه موازی و غیرموازی از الگوریتم فوق، استفاده نمودیم. جهت جمع آوری داده های مربوط به تغییرات سیستم نرم افزاری از سیستم های کنترل منبع استفاده می نماییم. این سیستم ها در هر بازه زمانی قادر به ثبت تغییرات محصول نرم افزاری به ازای هر تغییر ایجاد شده توسط هر یک از اعضای تیم توسعه نرم افزار می باشند.
همان طور که در جدول ۱ مشاهده می کنید برای ابعاد ماتریس کمتر از ۲۰ روش موازی شده دارای زمان اجرای بیشتری نسبت به



شکل ۱: مقایسه زمان اجرای الگوریتم به صورت موازی و ترتیبی بر حسب ثانیه.

روش ترتیبی می باشد، اما برای ابعاد بزرگتر از این مقدار روش موازی شده دارای سرعت اجرای کمتری می باشد. بدین جهت که

پژوهش ما به دلیل بررسی تغییرات توابع با یکدیگر دارای ماتریس هایی با ابعاد ورودی بسیار بالا می باشد، ما استفاده از رویکرد موازی را پیشنهاد می نماییم.
با توجه به شکل ۱ این موازی سازی انجام شده بر روی ۱۱ مورد از ابعاد ماتریس های متفاوت حاصل از تغییرات یک سیستم نرم افزاری مورد بررسی قرار گرفته است که اختلالات ناشی از آن به دلیل مقدار تنک بودن هر ماتریس می باشد که این اختلالات، تاثیری روی مقایسه دو رویکرد موازی و ترتیبی و نتیجه حاصل از این دو رویکرد نداشته است. این تغییرات ناشی از این می باشد که در یک نسخه از نرم افزار تغییرات کمتری نسبت به نسخه های دیگر وجود داشته است.

۵ نتیجه‌گیری

در رویکرد معرفی شده توسط (Sherriff و همکاران) به دلیل بررسی تغییرات فایل ها و در پی آن پایین بودن اندازه مسئله، نیازی به موازی سازی نبود [۴]. اما در رویکرد معرفی شده توسط ما به دلیل افزایش دقت و بررسی توابع تغییر یافته در فایل ها، با افزایش اندازه مسئله رو به رو شدیم که با رویکرد موازی ارائه شده توانستیم به این چالش محاسباتی غلبه نماییم. در نتیجه؛ این پژوهش که به کمک رویکرد موازی خوشه بندی کد منبع براساس تغییرات توابع توسعه یافته است، نسبت به پژوهش هایی که براساس تغییرات فایل های کد منبع انجام شده بود، دقت عمل بسیار بالاتری به دست آمده است.

تشکر و قدردانی

در آخر از دکتر کوروش پرند، هیئت علمی دانشکده ریاضی دانشگاه شهید بهشتی، که اینجانب را در این مسیر بسیار یاری نموده اند، کمال تشکر را دارم.

مراجع

[1] P. Ammann, J. Offutt, *Introduction to Software Testing*, Cambridge University Press, Cambridge, 2017.

[2] C. Chi-Lun ,H. Chin-Yu ,C. Chang-Yu ,C. Kai-Wen and L. Chen-Hua, Analysis and assessment of weighted combinatorial criterion for test suite reduction, *Quality and Reliability Engineering International*, (2021), 1–31.

[3] M. Duracik,E. Krsak and P. Hrkut, Searching source code fragments using incremental clustering, *Concurrency and Computation Practice and Experience*, 32 (2020), No. 13

[4] M. Sherriff, M. Lake and L. Williams, Prioritization of Regression Tests using Singular Value Decomposition with Empirical Change Records, The 18th IEEE International Symposium on Software Reliability (ISSRE '07), (2007).

[5] A. Spillner, T. Linz,*Software Testing Foundations A Study Guide for the Certified Tester Exam*, Rocky Nook, 2021.

[6] P. Xiao, Z. Wang and S. Rajasekaran, Novel Speedup Techniques for Parallel Singular Value Decomposition, 20th International Conference on High Performance Computing and Communications, 2018.