دانشگاه شهید بهشتی
دانشکده مهندسی و علوم کامپیوتر

گزارش پروژه کارشناسی رشته مهندسی کامپیوتر

پیاده‌سازی و ارزیابی روش‌های پیشنهادکننده‌ی بازبین کد

نگارش
شیـوا ضیـمران

نام استاد راهنما
دکتر صادق علی‌اکبری

نام اساتید داور

بهمن ماه ۱۴۰۰

دانشگاه شهید بهشتی
دانشکده مهندسی و علوم کامپیوتر

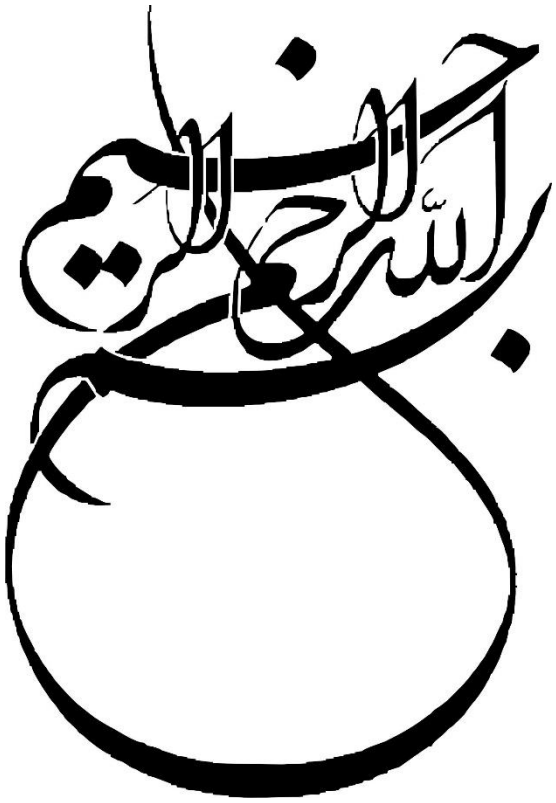
رشته مهندسی کامپیوتر

عنوان: پیاده‌سازی و ارزیابی روش‌های پیشنهادکننده‌ی بازبین کد

نگارش: شیوا ضیمران

استاد راهنما: دکتر صادق علی‌اکبری

تاریخ و امضاء


بسم الله الرحمن الرحيم

سپاسگزاری

چکیده

بازبینی[۱] کدهای نرم‌افزاری یکی از عملیات‌های حیاتی برای نگهداری، گسترش و تکامل پروژه‌های نرم‌افزاری است. بازبینی، بررسی تغییرات کد توسط یک توسعه‌دهنده[۲] است که این شخص، مستقل از توسعه‌دهندگان اصلی کد می‌باشد. هدف بازبینی، شناسایی و اصلاح کاستی‌های کد پیش از یکپارچه‌سازی آن با سایر اجزای پروژه است. انجام عمل بازبینی به طور مناسب، موجب افزایش کیفیت نرم‌افزار شده و با کشف زودرس خطاها، به صرفه‌جویی در زمان و هزینه می‌انجامد. در بازبینی مدرن[۳]، این عمل با کمک ابزارها انجام می‌شود که در سال‌های اخیر به طور گسترده مورد استفاده قرار می‌گیرد. پیدا کردن بازبین[۴] مناسب برای هر کد، از گام‌های مهم بازبینی است. یافتن بازبین مناسب روند بازبینی را سرعت و کیفیت می‌بخشد.

باتوجه به اهمیت ذکر شده برای عملیات بازبینی و انتخاب بازبین مناسب، در این پروژه به بررسی جدیدترین و مهم‌ترین روش‌های پیشنهادکننده‌ی بازبین کد[۵] در مقالات اخیر پرداخته‌ایم. روش‌های پیشنهادکننده‌ی بازبین کد، شامل روش‌های اکتشافی[۶] و روش‌های مبتنی بر هوش مصنوعی[۷] و یادگیری ماشین[۸] است. در این‌جا بر مقالات مبتنی بر روش‌های هوش مصنوعی تمرکز کرده و سعی در پیاده‌سازی دو مورد از این مقالات داشتیم.

اولین پیاده‌سازی با کمک الگوریتم ژنتیک، بازبین(بازبین‌های) مناسب برای بازبینی یک درخواست یکپارچه‌سازی[۹] را پیشنهاد می‌دهد. این کار با کمک استخراج ویژگی‌هایی از کاندیداهای بازبینی صورت می‌گیرد. این ویژگی‌ها شامل میزان تخصص هر فرد برای بازبینی آن کد، میزان دسترس‌پذیری وی و تاریخچه‌ی همکاری فرد با توسعه‌دهنده‌ی اصلی کد می‌باشد. سیستم توصیه‌گر در این مقاله، یک مساله‌ی جستجو را حل می‌کند که هر سه ویژگی ذکر شده را به طور همزمان در نظر می‌گیرد؛ به عبارت دیگر، ایجاد توازن بین تاثیر این سه پارامتر بر تصمیم‌گیری برای انتخاب یا عدم انتخاب یک بازبین، با حل یک الگوریتم ژنتیک چند منظوره[۱۰] صورت می‌گیرد.

در پیاده‌سازی دوم، ....

واژگان کلیدی: توسعه‌دهنده، بازبین، پیشنهادکننده‌ی بازبین کد، سیستم توصیه‌گر، یادگیری ماشین، هوش مصنوعی

---

[۱] Review
[۲] Developer
[۳] Modern Code Review
[۴] Reviewer
[۵] Code Reviewer Recommendation
[۶] Heuristic
[۷] Artificial Intelligence
[۸] Machine Learning
[۹] Pull Request
[۱۰] Multi-objective Genetic Algorithm

فهرست مطالب

عنوان **صفحه**

فهرست شکل‌ها

فهرست جدول‌ها

فهرست علائم اختصاری

درخواست یکپارچه‌سازی	Pull Request	PR
ماتریس جواب (کروموزوم‌ها)	Solution	S
ماتریس تخصص بازبین (ارتباطات فایل‌ها و بازبین‌ها)	File-Reviewer Matrix	FR
ماتریس همکاری بازبین–توسعه‌دهنده	Developer-Reviewer Matrix	DR
ماتریس دسترسی	Availability Matrix	A
ماتریس ارتباطات فایل‌ها و توسعه‌دهندگان	File-Developer Matrix	FD
تعداد کل فایل‌های درخواست ارسال شده		P
تعداد کل بازبین‌های پروژه		M
تعداد کل توسعه‌دهندگان پروژه		N

فصل اول – مقدمات

۱-۱. مقدمه

پروژه‌های نرم‌افزاری همواره نیاز به ارتقا، رفع مشکلات و به‌روزرسانی دارند. این تغییرات می‌تواند در حین توسعه یا پس از آن توسط فرد یا افرادی صورت بگیرد. هر شخص می‌تواند یک تغییر یا به‌روزرسانی را برای یک پروژه نرم‌افزاری تعریف کرده و آن را برای ادغام[۱۱] با پروژه اصلی ارسال کند[۱۲]. سپس افرادی به عنوان بازبین به بررسی این تغییرات می‌پردازند. انتخاب بازبین مناسب برای هر درخواست ارسال شده مزایایی داشته و باتوجه به فاکتورهایی صورت می‌گیرد که در ادامه به آن‌ها خواهیم پرداخت.

۱-۲. تعریف مساله

توسعه‌دهندگان همواره می‌توانند برای هر پروژه نرم‌افزاری درخواست‌هایی شامل تغییراتی در هر جزء پروژه، برای توسعه‌دهندگان اصلی آن پروژه ارسال کنند تا آن‌ها برای ادغام یا رد این درخواست‌ها تصمیم‌گیری کنند. این درخواست می‌تواند در ابزارهایی مثل Github، Gerrit یا ... ثبت و ارسال شود. در این حین، افراد دیگری (که می‌تواند شامل توسعه‌دهندگان اصلی پروژه باشد یا نباشد)، این درخواست را بازبینی کرده و نظر[۱۳] خود را راجع به آن اعلام یا تغییراتی در کد نوشته شده ایجاد می‌کنند (مانند حذف یا افزودن خطوطی به کد). نظرات بازبین‌ها می‌تواند توسط ارسال‌کننده‌ی تغییر اعمال شده و مجددا همان بازبین‌ها یا اشخاص دیگری مورد جدید را بازبینی کنند. این روند می‌تواند ادامه یابد تا جایی که یکی از توسعه‌دهندگان اصلی پروژه تصمیم بگیرد که این درخواست را ببندد یا آن را با پروژه‌ی اصلی ادغام کند.

هر فردی می‌تواند تمام درخواست‌های ارسال شده را بازبینی کرده و نظرات خود را اعلام کند اما انتخاب بازبین (بازبین‌های) مناسب برای هر درخواست، باتوجه به محتوای کد ارسال شده و ویژگی‌های دیگری از توسعه‌دهندگان موجود، مزایای خاصی دارد. یکی از این مزایا، بهبود کیفیت بازبینی‌ها خواهد بود؛ باتوجه به این که بازبین‌ها مطابق تخصصی که دارند برای بازبینی انتخاب می‌شوند، بازبینی‌ها به صورت دقیق‌تر و تخصصی‌تر صورت می‌گیرند. از مزایای دیگر آن، سرعت بخشیدن به عمل بازبینی است؛ زیرا در دسترس بودن یک فرد برای انتخاب وی به عنوان بازبین، معمولاً یک فاکتور تاثیرگذار است. بنابراین نیاز به روش‌هایی برای انتخاب و پیشنهاد بازبین‌های مناسب برای هر درخواست ارسال شده داریم.

پیشنهاد بازبین مناسب و پیاده‌سازی یک سیستم توصیه‌گر برای این منظور، هدف اصلی این پژوهش می‌باشد. یک سیستم توصیه‌گر، باتوجه به پیشینه‌ی درخواست‌های ارسال شده توسط هر توسعه‌دهنده و بازبین‌هایی که آن درخواست را بررسی کرده‌اند، عمل می‌کند. فردی که تغییری را ایجاد و با هدف ادغام با پروژه‌ی اصلی از طریق یکی از ابزارهای موجود برای توسعه‌دهندگان اصلی پروژه ارسال می‌کند، مالک[۱۴] آن درخواست نامیده می‌شود. سیستم توصیه‌گر، ساختار، محتوا، عنوان،

---

[۱۱] Merge

[۱۲] Submitter of a PR

[۱۳] Comment

[۱۴] Owner

تاریخ و سایر مشخصات این درخواست را استخراج و باتوجه به تاریخچه‌ی موجود از بازبینی‌های گذشته، بازبین(های) مناسب را پیشنهاد می‌کند.

افرادی که برای بازبینی توسط سیستم پیشنهادکننده اعلام می‌شوند به ترتیب از مناسب‌ترین فرد تا شخصی که کم‌تر مناسب بازبینی این درخواست است، مرتب شده‌اند. این که یک فرد مناسب‌تر از دیگری باشد، براساس ویژگی‌هایی تعریف می‌شود که وابسته به ساختار سیستم توصیه‌گر است. در هر روش پارامتر یا پارامترهایی برای این منظور تعریف شده که عملکرد سیستم پیشنهادکننده تا حد زیادی به تعریف این پارامترها وابسته است.

یک سیستم پیشنهادگر مناسب در این مساله سیستمی است که بتواند بازبین‌های حقیقیِ کدهای ارسالی را به درستی پیش‌بینی کند. به عبارت دیگر، همه یا تعدادی از بازبین‌های حقیقی در لیست پیشنهادی سیستم یافت شود. واضح است که هرچه بازبین‌های حقیقی در اولویت‌های بالاتری از لیست پیشنهادی باشند، یعنی سیستم به درستی این افراد را به عنوان مناسب‌ترین بازبین‌ها شناسایی کرده و دقت سیستم توصیه‌گر بالاتر خواهد بود.

۱-۳. انگیزش ناشی از پایان‌نامه

سیستم‌های پیشنهادکننده‌ی متنوعی در سال‌های اخیر ارائه شده که مبتنی بر روش‌های اکتشافی یا هوش مصنوعی است. در این پایان‌نامه قصد داریم چند مورد از جدیدترین و بهترین روش‌های پیاده شده در حوزه‌ی هوش مصنوعی را معرفی و دو مورد را اجرا کنیم. سعی بر پیاده‌سازی دقیق مقاله‌های منتخب و رسیدن به یک سیستم توصیه‌گر با دقت مناسب، از انگیزه‌های این پروژه است.

پس از آن به بهبود سیستم پیاده شده خواهیم پرداخت و با ارائه‌ی ایده‌هایی نو تلاش می‌کنیم دقت پیشنهادکننده را افزایش دهیم.

۱-۴. ساختار پایان‌نامه

حال که با مقدمات موضوع در این فصل آشنا شدیم و به فهم دقیق‌تری از مساله رسیدیم، در ادامه به توضیح دقیق‌تر روش‌های پیاده شده خواهیم پرداخت. در فصل دوم به توضیح بیش‌تر مقالاتی که در این حوزه ارائه شده‌اند می‌پردازیم و آن‌ها را مقایسه خواهیم کرد. سپس در فصل سوم روش‌های منتخب برای پیاده‌سازی را به تفصیل معرفی کرده و به توضیح جزئیات کارهای انجام شده و چالش‌های پشت‌سر گذاشته خواهیم پرداخت. همچنین در انتهای این فصل نتایج به دست آمده را گزارش و تحلیل می‌کنیم. در انتها، نتایج به دست آمده و دستاوردهای پروژه را در فصل چهارم معرفی کرده و به ارائه‌ی راه‌حل‌هایی برای ادامه‌ی تحقیقات خواهیم پرداخت.

فصل دوم – مروری بر کارهای انجام شده

۲-۱. مقدمه

مقالات متعددی در طی سال‌های ۲۰۰۹ تا ۲۰۲۰ میلادی برای طراحی یک سیستم پیشنهادکننده‌ی بازبین کد ارائه شده‌است. روش‌های مورد استفاده در این تحقیقات عموماً در یکی از دسته‌های کلی فوق قرار می‌گیرند: روش‌های اکتشافی، روش‌های مبتنی بر یادگیری ماشین یا ترکیبی[۱۵] از این دو. در چند سال اخیر نیز سیستم‌هایی پیشنهاد شدند که از یادگیری عمیق برای طراحی سیستم توصیه‌گر بهره می‌برند. در این فصل به توصیف و تحلیل مختصر کارهای انجام شده خواهیم پرداخت تا چهارچوب دقیق‌تری از حوزه‌ی پژوهش را تبیین کنیم.

۲-۲. مرور بر کارهای انجام شده

برای آغاز این پژوهش ابتدا به بررسی دقیق مقالاتی پرداختیم که در این زمینه مطرح شده بودند. اکثر مقالات بررسی شده را از میان آن‌هایی برگزیدیم که از روش‌های هوش مصنوعی و یادگیری ماشین استفاده کرده بودند. البته روش‌های اکتشافی مشهوری مانند روش REVFINDER [۱] یا از بین روش‌های ترکیبی، متد RevRec [۲] نیز به دقت مطالعه شدند. اما بررسی‌ها نشان داد که برخی روش‌های یادگیری ماشین با گزارش نتایج خود و مقایسه‌ی خود با روش‌های معروف اکتشافی (مانند REVFINDER، cHRev [۳] و Review Bot [۴]) یا هیبریدهای معروفی چون RevRec و TIE [۵] توانسته‌بودند به دقت‌های بالاتری دست یابند. بنابراین بر آن شدیم تا به بررسی بیش‌تر روش‌های مبتنی بر یادگیری ماشین بپردازیم و دو مورد را به عنوان نمونه‌های مناسبی از آن‌ها پیاده‌سازی کنیم.

اولین مقاله‌ی منتخب[۶]، سیستمی را معرفی می‌کند که با کمک الگوریتم ژنتیک چندمنظوره عمل می‌کند. نزدیک‌ترین کار مشابه که پیش از آن و در سال ۲۰۱۸ میلادی انجام شده، روش RevRec است. این روش نیز از الگوریتم ژنتیک استفاده می‌کند و برای پیشنهاد بازبین‌های مناسب برای یک بازبینی، از تخصص و تاریخچه‌ی همکاری‌ها بهره می‌برد. این مقاله یک مساله‌ی بهینه‌سازی یک منظوره[۱۶] را حل می‌کند؛ براساس تخصص هر بازبین بر هر یک از فایل‌های موجود در درخواست ارسال شده و همچنین تعاملات گذشته میان بازبین و مالک درخواست، بازبین‌های مناسب شناسایی و پیشنهاد می‌شوند. مقاله‌ی انتخابی ما که با نام AEC (NSGA-II) شناخته می‌شود، از چند منظر با مقاله‌ی ذکر شده تفاوت دارد؛ از جمله آن که در این روش، علاوه بر دو فاکتور تخصص و همکاری‌ها، دسترس‌پذیری بازبین‌ها نیز در نظر گرفته می‌شود. درواقع بهترین بازبین با کمک دو پارامتر اول، اگر مشغول کارهای دیگری باشد (دسترس‌پذیری کم‌تر)، به عنوان بازبین مناسب در اولویت‌های پایین‌تری پیشنهاد می‌شود. این امر موجب می‌شود تا تاخیر سیستم برای بازبینی فایل‌ها کاهش یابد و بازبینی‌ها با سرعت بیش‌تری انجام شوند. بنابراین روش جدید می‌بایست یک مساله‌ی چندمنظوره را حل کند. در گزارش‌های نهایی مقاله، نتایج روش با روش‌های RevRec، cHRev، Review Bot و REVFINDER با معیارهای ارزیابی متفاوتی مقایسه شده که

---

[۱۵] Hybrid

[۱۶] Single Objective Optimization

روش AEC (NSGA-II) بهترین نتایج را داشت. مقاله بر روی ۹ پروژه متفاوت از گیت‌هاب تست شده که به طور میانگین، پیشنهادات بهتری نسبت به بهترین روش موجود[۱۷] در این حوزه تا سال ۲۰۲۰م. داشته‌است.

دومین مقاله‌ی منتخب، ....

۲-۳. جمع‌بندی و نتیجه‌گیری

مطابق توضیحات داده شده، روش AEC (NSGA-II) که در سال ۲۰۲۰ میلادی منتشر شده، از بسیاری روش‌های معروف این حوزه عملکرد بهتری دارد. بنابراین با پیاده‌سازی آن یک سیستم توصیه‌گر خواهیم داشت که با دقت بالایی عمل می‌کند. روش دوم ...

در فصل بعد جزئیات پیاده‌سازی هر کدام به تفصیل شرح داده شده و نتایج عملکرد هریک را گزارش می‌کنیم.

[۱۷] State of the art

فصل سوم – موضوع پیشنهادی پایان‌نامه

۳-۱. مقدمه

در این فصل به تعریف دقیق روش‌های پیاده شده در این تحقیق و چالش‌ها و نوآوری‌های هر یک در دو زیربخش مجزا خواهیم پرداخت. در انتهای هر روش نیز نتایج را آورده و تفسیر می‌کنیم.

۳-۲. روش AEC (NSGA-II) [۱۸]

۳-۲-۱. تعریف دقیق مساله

برای پیش‌بینی بازبین یا بازبین‌های مناسب برای یک درخواست ارسال شده، ابتدا باید اطلاعاتی از این درخواست استخراج کنیم و با کمک آن‌ها فاکتورهای تاثیرگذار برای تصمیم‌گیری را محاسبه کنیم. این فاکتورهای مهم عبارتند از: میزان تخصص هر بازبین برای بازبینی هر فایل موجود در درخواست، تاریخچه همکاری‌های بین بازبین‌ها و توسعه‌دهندگان و میزان در دسترس بودن هر بازبین.

پس از محاسبه‌ی سه مقدار قبل برای هر درخواست و با کمک درخواست‌هایی که در تاریخچه موجود است، به سراغ حل یک مساله‌ی بهینه‌سازی می‌رویم که روش انتخاب شده الگوریتم NSGA-II می‌باشد. در این مساله، سه هدف برای بهینه سازی داریم و به همین دلیل آن را چند منظوره می‌نامیم:

- پارامتر تخصص که می‌بایست ***بیشینه*** شود؛ چرا که تخصص بیش‌تر فرد بر فایل‌های موجود در درخواست، موجب بالا بردن کیفیت بازبینی‌ها می‌شود.
- پارامتر دسترس‌پذیری که باید ***بیشینه*** شود. همان‌طور که پیش از این اشاره کردیم، بازبین‌هایی اولویت بالاتری دارند که علاوه بر تخصص، دسترس‌پذیری مناسبی نیز داشته‌باشند تا بتوانند به موقع بازبینی‌ها را به انجام رسانده و تاخیر کلی سیستم را برای پاسخگویی کاهش دهند.
- پارامتر همکاری که باید ***کمینه*** گردد؛ این فاکتور برای جلوگیری از متمایل شدن[۱۹] پیشنهادات سیستم به سمت بازبین‌هایی است که پیش از این با مالک درخواست همکاری‌هایی داشته‌اند.

با حل این مساله‌ی بهینه‌سازی و یافتن پاسخی که هر سه هدف بالا را به طور همزمان برآورده کند، به مجموعه جوابی خواهیم رسید که در هر جواب، به ازای هر فایل موجود در درخواست، بازبین‌های پیشنهادی به صورت مرتب شده گزارش می‌شوند.

[۱۸] https://github.com/shivaZeymaran/Multi-objective-code-reviewer-recommendations.git

[۱۹] Bias

۳-۲-۲. بیان حل مساله و روش پیاده‌سازی

در تعریف دقیق روش گفتیم که روند کلی عملکرد سیستم برای پیشنهاد بازبین‌ها از ابتدای ارسال یک درخواست تا پیش‌بینی افراد مناسب به چه صورت است. در این قسمت به توضیح جزء به جزء اجزا خواهیم پرداخت.

۳-۲-۲-۱. استخراج مجموعه داده مناسب

اولین گام در حل چنین مسائلی، انتخاب مجموعه دادگان[۲۰] مناسب می‌باشد. در این مقاله، سیستم بر روی ۹ پروژه‌ی متفاوت تست شده که شامل پروژه‌های متن‌باز[۲۱]ی مانند OpenStack، Android و Qt می‌باشد. ما نیز در این پژوهش از داده‌های پروژه‌ی OpenStack [۲۲] استفاده کرده‌ایم. این مجموعه دادگان یک فایل json. است که شامل اطلاعاتی از درخواست‌های مختلف ارسال شده در بازه‌ی زمانی ۲۰۱۱/۷ الی ۲۰۱۲/۵ می‌باشد. اطلاعات نگهداری شده برای هر درخواست شامل نام زیرپروژه[۲۳]، شناسه‌ی تغییر[۲۴]، شماره‌ی تغییر[۲۵]، مهر زمانی[۲۶]، لیستی از شناسه و سایر اطلاعات بازبین‌های درخواست، شناسه‌ی ارسال‌کننده‌ی درخواست و لیستی از مسیر[۲۷] فایل‌های موجود در درخواست می‌باشد.

از میان اطلاعات موجود برای هر درخواست، تنها شناسه‌ی تغییر، مهر زمانی، لیست شناسه‌ی افراد بازبین، شناسه‌ی مالک درخواست و لیست مسیر فایل‌ها را نگه می‌داریم. از آن‌جا که ترتیب زمانی درخواست‌ها برای ادامه‌ی کار و فرآیند یادگیری اهمیت فراوانی دارد، درخواست‌ها را بر اساس زمان ایجاد از قدیم‌تر به جدیدتر مرتب می‌کنیم.

۳-۲-۲-۲. استخراج اجزای اصلی رویکرد

اولین گام با داشتن دادگان مناسب، استخراج اجزای اصلی رویکرد است:

- مدل تخصص بازبین: این مدل به شکل یک ماتریس است که FR نام دارد و بیانگر ارتباطات بازبین‌ها و فایل‌های موجود در درخواست می‌باشد. فایل‌های *هر درخواست* در سطرها و بازبین‌های *کل پروژه* در ستون‌های این ماتریس جای می‌گیرند. درایه‌ی سطر iام و ستون jام ماتریس یک عدد حسابی است که بیان می‌کند فایل iام چند بار توسط بازبین jام بازبینی شده.

---

[۲۰] Dataset

[۲۱] Open Source

[۲۲] https://github.com/XLipcak/rev-rec/tree/master/rev-rec-data

[۲۳] Sub-project

[۲۴] Change ID

[۲۵] Change Number

[۲۶] Timestamp

[۲۷] Path

- مدل همکاری بازبین–توسعه‌دهنده: بیانگر ماتریس DR است که کل توسعه‌دهندگان پروژه در سطرها و کل بازبین‌ها در ستون‌ها قرار می‌گیرند. هر درایه یک عدد حسابی و بیانگر تعداد فایل‌هایی است که هر جفت توسعه‌دهنده و بازبین در آن همکاری داشته‌اند.
- مدل دسترسی: یک آرایه با نام A است که تعداد فایل‌هایی که هر بازبین در ۷ روز اخیر بررسی کرده را نشان می‌دهد.

همچنین ماتریس FD را محاسبه می‌کنیم که کل توسعه‌دهندگان پروژه در سطرها و فایل‌های درخواست در ستون‌هایش قرار می‌گیرند. هر درایه بیانگر این است که هر توسعه‌دهنده پیش از این چند مرتبه روی هر فایل تغییری ایجاد کرده است.

۳-۲-۲-۳. الگوریتم ژنتیک

همان‌طور که پیش از این اشاره کردیم، الگوریتم مورد استفاده در این سیستم توصیه‌گر الگوریتم تکاملی است. در این‌جا به تشریح اجزای الگوریتم، تعریف بازنمایی مورد استفاده و عملگرهای تغییر خواهیم پرداخت.

- بازنمایی[۲۸]: هر کروموزوم در این مساله به شکل یک ماتریس است. این کروموزوم‌ها که افراد جمعیت را تشکیل داده و بهترین آن‌ها جواب نهایی را مشخص می‌کند، متشکل است از رتبه‌ی هر بازبین برای بازبینی تک تک فایل‌های موجود در درخواست. به طور دقیق‌تر، سطرهای ماتریس کروموزوم، فایل‌های درخواست و ستون‌هایش، همه‌ی بازبین‌های کل پروژه از ابتدای ایجاد پروژه تا زمان ارسال این درخواست می‌باشد. درایه‌ها نیز رتبه‌ی هر بازبین را برای بازبینی آن فایل مشخص می‌کنند.
  اگر درایه‌ای صفر باشد به این معناست که آن بازبین به هیچ عنوان برای بازبینی آن فایل مناسب نیست. به جز آن، هر چه درایه عدد کوچک‌تری باشد، یعنی آن بازبین مناسب‌تر است. مثلاً بازبین با رتبه‌ی ۳ برای بازبینی یک فایل از بازبین با رتبه‌ی ۵ برای بازبینی همان فایل، مناسب‌تر است و باید در رده‌های بالاتری پیشنهاد شود.

- بازترکیبی[۲۹]: بازترکیبی مورد استفاده در این مقاله به صورت *تک نقطه‌ای* است. به این صورت که یک عدد تصادفی[۳۰] برای شماره سطر و عدد تصادفی دیگری برای شماره ستون تولید و ماتریس دو کروموزوم والد، از آن سطر و ستون شکسته می‌شود. برای فرزند اول، از ابتدای والد اول تا درایه‌ی موجود در این سطر و ستون و از این درایه تا انتهای ماتریس از والد دوم انتخاب می‌شود. برای فرزند دوم نیز برعکس قبل عمل می‌شود.

توجه: پس از انجام عمل بازترکیبی، ممکن است کروموزوم‌های فرزند در سطری که شکست در آن اتفاق افتاده، نیاز به اصلاح داشته‌باشند. چون رتبه‌ها در هر سطر شامل هیچ یا تعدادی صفر بوده و بقیه رتبه‌ها اعداد یک و بزرگ‌تر از آن و به صورت یکتا هستند. با انجام بازترکیبی این یکتا بودن ممکن است برهم خورده و ژن‌ها نیاز به اصلاح

---

[۲۸] Representation
[۲۹] Crossover
[۳۰] Random

داشته‌باشند. برای این‌کار تنها کافی است رتبه‌های غیر صفر را به ترتیب مرتب کنیم و شماره‌های جدیدی به آن‌ها نسبت داده و شماره‌های جدید را به عنوان رتبه‌های اصلاح شده جایگزین نماییم.

- جهش[۳۱]: عملیات جهش مورد استفاده، جانشینی[۳۲] است و با یک احتمال پایین می‌تواند در ژن‌های یک کروموزوم رخ دهد. جهش می‌تواند در هر سطر ماتریس با یک احتمال رخ دهد. دو سطری که بالاترین احتمال رخداد جهش را داشته باشند و این احتمال از یک عدد مشخص[۳۳] بزرگ‌تر باشد، با یک‌دیگر جابه‌جا می‌شوند.

- توابع شایستگی[۳۴]: سه تابع شایستگی در این مساله تعریف می‌شود:

  - تابع دسترسی[۳۵]: این تابع معکوس زمان انتظار تقریبی است. هرچه مخرج کسر کوچک‌تر باشد، یعنی مقدار زمان انتظار برای بازبینی توسط یک بازبین کم‌تر بوده و دسترسی بیش‌تر است که مطلوب می‌باشد.
  این مجموع زمان انتظار –که در رابطه (۳–۱) هم آورده شده– برای هر بازبین به صورت حاصلضرب مقدار ماتریس دسترسی فرد (A) در رتبه‌ی فرد است. هر چه عدد ماتریس دسترسی کوچک‌تر باشد، یعنی فرد کارهای صف‌شده‌ی کم‌تری برای انجام دارد. حال اگر رتبه‌ی فرد هم عدد کوچک‌تری باشد (فرد مناسب‌تری است)، این حاصلضرب کوچک‌تر و تابع شایستگی دسترسی بزرگ‌تر می‌شود که مناسب‌تر بودن این فرد را تایید می‌کند.

$$Availability = \frac{1}{\sum_{k=1}^{P}\sum_{i=1}^{M} A_i * S[k,i]}, s_{k,i} > 0 \ . \qquad (۱–۳)$$

  توجه شود که در محاسبه‌ی این فرمول تنها دسترس‌پذیری فرد در یک هفته گذشته ارزیابی می‌شود و این قید در محاسبه‌ی ماتریس A در نظر گرفته شده.

  - تابع تخصص[۳۶]: این تابع به صورت مجموع تخصص کل بازبین‌ها بر تک تک فایل‌های درخواست محاسبه می‌شود (رابطه‌ی ۳–۲). برای یک بازبین، هرچه تخصص آن بیش‌تر باشد (یعنی آن فایل را پیش از این چند مرتبه بازبینی کرده باشد) و رتبه‌ی فرد عدد کوچک‌تری باشد، مقدار این تابع بزرگ‌تر می‌شود که به درستی شایسته‌تر بودن فرد را مشخص می‌کند.

---

[۳۱] Mutation
[۳۲] Swap
[۳۳] Gene Modification Probability(Modif_prob)
[۳۴] Fitness Functions
[۳۵] Availability
[۳۶] Expertise

$$Expertise = \sum_{k=1}^{P} \sum_{i=1}^{M} \frac{FR[k,i]}{S[k,i]}, s_{k,i} > 0 \quad . \qquad (۲-۳)$$

- تابع همکاری[۳۷]: این تابع برای بازبین‌هایی که رتبه‌ی غیرصفر دارند محاسبه می‌شود و مقدار آن به این صورت است که برای هر توسعه‌دهنده‌ی پروژه، مقدار مشارکت وی با هر بازبین در تعداد دفعاتی که توسعه‌دهنده روی هر فایل کار کرده ضرب می‌شود (رابطه‌ی ۳-۳).

$$Collaboration = \sum_{k=1}^{N} \sum_{j=1}^{P} \sum_{i=1}^{M} DR[k,i] * FD[k,j] * (S[j,i] > 0) , \qquad (۳-۳)$$

که در آن $(S[j,i] > 0)$ یک پوشش دودویی[۳۸] از ماتریس S است و به جای مقادیر غیرصفر S، مقدار یک را جایگزین می‌کند.

در محاسبه‌ی توابع شایستگی برای افزایش سرعت اجرای سیستم، در پیاده‌سازی به زبان پایتون، از توابع کتابخانه‌ی numpy مانند sum، dot و divide بهره برده‌ایم که عملیات‌ها را با سرعت بسیار بهتری انجام می‌دهند.

حال که اجزای الگوریتم تکاملی را شرح دادیم، به توضیح اصل الگوریتم ژنتیک و چگونگی استفاده از موارد قبل در کنار یک دیگر خواهیم پرداخت.

برای شروع الگوریتم تکاملی ابتدا باید جمعیت اولیه را ایجاد کنیم. برای این امر، به تعداد اعضای جمعیت (که می‌تواند ۱۰، ۲۰، ۳۰، ۴۰ یا ۵۰ باشد) کروموزوم‌هایی می‌سازیم که دارای ژن‌ها با مقادیر تصادفی است. به بیان دقیق‌تر، برای هر سطر از ماتریس کروموزوم، به تعداد رندومی عدد صفر داریم (این فرض را در نظر گرفته‌ایم که همه‌ی درایه‌های یک سطر نمی‌تواند صفر باشد و حداقل یک بازبین باید برای هر فایل پیشنهاد شود) و بقیه درایه‌ها مقادیر یک و بزرگ‌تر از یک را خواهند داشت.

در هر نسل از الگوریتم تکاملی، پس از ایجاد فرزندان از والدین با کمک عملگرهای بازترکیبی و جهش به شکلی که گفته شد، جمعیت را اجتماع والدین و فرزندان ایجاد شده در نظر می‌گیریم. توجه شود که انتخاب دو والد برای ایجاد فرزندان به صورت تصادفی می‌باشد و هر والد می‌تواند با خود نیز ترکیب شود. حال برای همه‌ی اعضای جمعیت، سه مقدار توابع شایستگی را محاسبه می‌کنیم.

الگوریتم ژنتیک چند منظوره‌ای که در این پژوهش استفاده می‌شود، الگوریتم NSGA-II [۷] می‌باشد. پس از محاسبه‌ی توابع شایستگی برای هر عضو جمعیت، باید اعضا را به ترتیب شایستگی مرتب کنیم و بهترین‌ها را به نسل بعد منتقل کنیم

---

[۳۷] Collaboration

[۳۸] Binary Mask

(Selection Mechanism). این مرتب‌سازی در NSGA-II با الگوریتم Fast Non-Dominated Sort و Crowding Distance انجام می‌گیرد. به این صورت که ابتدا کل جمعیت را با Fast Non-Dominated Sort مرتب می‌کنیم. با این مرتب‌سازی، اعضا در بسته‌هایی به نام Front قرار می‌گیرند. هر چند عضو در یک Front خواهند بود و Front اول شایسته‌ترین اعضا را در خود دارد. سپس از اولین Front شروع می‌کنیم و اعضا را انتخاب کرده و به نسل بعد می‌بریم. ممکن است نیاز باشد که یک فرانت شکسته شده و تنها بعضی اعضایش برای رفتن به نسل بعد انتخاب شوند. از آن‌جا که اعضای داخل یک فرانت همگی از نظر شایستگی یکسان تلقی می‌شوند، نیاز به پارامتر دیگری برای تعیین شایستگی میان اعضای یک فرانت داریم. این پارامتر همان Crowding Distance می‌باشد. با محاسبه‌ی این فاصله برای تک تک اعضای فرانتی که قرار است شکسته شود، این اعضا بر حسب شایستگی مرتب خواهند شد و بهترین‌ها به نسل بعد خواهند رفت. مجددا اعضایی که به نسل بعد رفته‌اند، دچار بازترکیبی و جهش شده، فرزندان ایجاد و همین روند تکرار می‌شود. شرط خاتمه برای الگوریتم، انجام ۱۰۰,۰۰۰ ارزیابی شایستگی[۳۹] بیان شده. این Crowding Distance علاوه بر محاسبه برای اعضای فرانتی که شکسته می‌شود، باید در آخرین نسل نیز برای کل جمعیت حساب شود؛ زیرا خروجی الگوریتم یک مجموعه جواب به اندازه‌ی سایز جمعیت است که ما برای گزارش یک جواب، باید بهترین آن‌ها را از نظر شایستگی انتخاب کنیم. پس جواب‌های داخل یک فرانت هم می‌بایست بر اساس شایستگی از بهترین تا بدترین مرتب شوند که این کار با محاسبه‌ی Crowding Distance امکان‌پذیر است. بنابراین در مجموعه جواب نهایی الگوریتم، اولین جواب، بهترین جواب خواهد بود.

- Fast Non-Dominated Sort

در این الگوریتم برای مرتب‌سازی اعضای جمعیت، آرایه‌ی S و مقدار n برای هر عضو محاسبه می‌شود. S یک عضو، لیستی است از افرادی که آن عضو بر آن‌ها برتری دارد. یک عضو از عضو دیگر برتر است اگر در حداقل یکی از ۳ تابع شایستگی از دیگری بهتر باشد و در هیچ یک از توابع شایستگی از دیگری بدتر نباشد[۴۰]. n یک عضو نیز برابر است با تعداد افرادی از جمعیت که از این عضو برتر هستند. اعضایی از جمعیت که دارای n برابر صفر باشند در اولین فرانت قرار می‌گیرند که این افراد شایسته‌ترین اعضا هستند. جواب‌هایی که بر یک دیگر برتری ندارند و هیچ جواب برتر از آن‌ها نیز در جمعیت وجود ندارد در این فرانت قرار گرفته و جواب‌های Pareto-optimal نام دارند.

برای جای دادن سایر اعضا در بقیه‌ی فرانت‌ها می‌بایست برای اعضای فرانت فعلی، عناصر S را بررسی کرده و هرکدام دارای n=1 بود، این عنصر در فرانت بعدی قرار می‌گیرد [۸].

- Crowding Distance

برای تخمین تراکم جواب‌ها در اطراف یک جواب خاص، باید میانگین فواصل دو نقطه در دو سمت این نقطه را با کمک هر تابع شایستگی محاسبه کنیم که این مقدار Crowding Distance نام دارد. برای محاسبه‌ی این مقدار برای هر عضو جمعیت، ابتدا Crowding Distance همه را صفر قرار می‌دهیم. سپس جمعیت را هر بار

[۳۹] Fitness Evaluation

[۴۰] Domination

بر اساس یکی از توابع شایستگی مرتب کرده و مقدار فاصله را برای دو جواب کرانه‌ای برابر بی‌نهایت قرار می‌دهیم. با کمک مقادیر بیشینه و کمینه، فاصله‌ی نقطه از نقاط قبلی و بعدی‌اش را نرمال می‌کنیم. در انتها باید Crowding Distanceهای محاسبه شده به ازای هر تابع شایستگی را با یک‌دیگر جمع کنیم تا Crowding Distance هر عضو جمعیت به دست آید [۹].
جوابی که دارای Crowding Distance بزرگ‌تر است، جواب شایسته‌تری می‌باشد. بنابراین جواب‌هایی که در یک فرانت هستند را بر اساس Crowding Distance به صورت نزولی مرتب می‌کنیم و به تعدادی که برای انتقال به نسل بعد لازم داریم از ابتدای لیست مرتب شده برمی‌داریم.

۳-۲-۲-۴. اجرای الگوریتم بر روی دادگان

همان‌طور که پیش از این گفتیم، دادگان مورد استفاده مربوط به پروژه‌ی OpenStack در یک بازه‌ی زمانی حدودا ۱۰ ماهه می‌باشد که اطلاعات کلی آن در جدول ۳-۱ آورده شده.

جدول ۳-۱. خلاصه‌ی سیستم مورد مطالعه

نام پروژه	بازه زمانی مورد مطالعه	تعداد بازبینی‌ها	تعداد فایل‌ها	تعداد بازبین‌ها	تعداد مالکین
OpenStack	2011-07-18 15:43:34 تا 2012-05-30 21:39:57	۶,۵۴۵	۱۱,۴۲۱	۸۲	۳۲۴

حال باید دادگان را به دو قسمت یادگیری[۴۱] و ارزیابی[۴۲] تقسیم کنیم. برای این کار باتوجه به مرتب بودن درخواست‌ها در دادگان بر حسب زمان ایجاد، ۹۰٪ اول داده‌ها را برای اولین یادگیری استفاده می‌کنیم (درخواست ۰ تا ۵۸۹۹) و درخواست بعدی (۵۹۰۰ام) را به عنوان اولین داده‌ی تست در نظر می‌گیریم. سپس از درخواست ۰ تا ۵۹۰۰ را برای یادگیری استفاده کرده و ۵۹۰۱ را به عنوان تست انتخاب می‌کنیم و به همین ترتیب پیش می‌رویم. با این ترفند، ۱۰٪ انتهایی داده‌ها که ۶۴۵ درخواست است را ارزیابی کرده‌ایم و نتیجه را بر اساس ارزیابی این داده‌ها گزارش می‌کنیم.

به ازای هر درخواست تست، ابتدا کل فایل‌های این درخواست و کل بازبین‌ها و توسعه‌دهندگان موجود در داده‌ی یادگیری را استخراج می‌کنیم. سپس ۴ ماتریس FR، DR، A و FD را با کمک داده‌ی یادگیری محاسبه می‌کنیم. اکنون الگوریتم ژنتیک باید اجرا شود. با در نظر گرفتن محدودیت‌های زمانی و منابع در اختیار، می‌بایست تست‌ها را با یک جمعیت و تعداد نسل محدودتر اجرا کنیم به نحوی که بتوان خروجی‌های صحیح به دست آورد. باتوجه به آزمون و خطاهای انجام شده،

[۴۱] Train
[۴۲] Test

جمعیت ۳۰ و تعداد نسل ۱۰۰ – تقریبا معادل ۱۰,۰۰۰ محاسبه‌ی تابع شایستگی– را برای انجام الگوریتم انتخاب کرده‌ایم. احتمال بازترکیبی و جهش مطابق متن مقاله به ترتیب مقادیر ۰/۵ و ۰/۴ را خواهند داشت.

اشاره کردیم که پاسخ نهایی یک مجموعه جواب است که براساس شایستگی مرتب شده و جواب نهایی همان اولین عضو از این مجموعه جواب می‌باشد. جواب نهایی خود یک ماتریس است که رتبه‌بندی بازبین‌ها را برای هر *فایل* موجود در درخواست به طور *مجزا* مشخص می‌کند؛ اما ما باید به ازای هر *درخواست*، تنها ۱ تا ۱۰ بازبین را به عنوان بازبین‌های پیشنهادی، به ترتیب مناسب بودن، پیش‌بینی کنیم. چون مقاله برای این کار ترفندی ارائه نداده یا حداقل آن را صراحتاً ذکر نکرده، باید یک فرض به مساله اضافه کنیم. روش کار به این صورت است که باتوجه به این که ما حداکثر ۱۰ تا بازبین برتر را می‌خواهیم، پس top-10 هر فایل درخواست را استخراج کرده و اجتماع آن را نگه می‌داریم. حال این بازبین‌های حاصل از اجتماع را در دو مرحله مرتب می‌کنیم:

- برای هر بازبین در این مجموعه، *تعداد رتبه‌های غیرصفر* در کل فایل‌های درخواست را محاسبه می‌کنیم. هر چه این مقدار برای یک بازبین *بزرگ‌تر* باشد، به این معنا است که آن بازبین برای بازبینی فایل‌های بیش‌تری از این درخواست مناسب است و در نتیجه آن بازبین مناسب‌تر می‌باشد.
- سپس از بین افرادی که دارای مقادیر قبلی یکسانی هستند، کسی را مناسب‌تر گوییم که *مجموع* رتبه‌هایش برای فایل‌های آن درخواست عدد *کم‌تری* باشد؛ چون به این معناست که با اولویت بالاتری مناسب فایل‌های این درخواست است.

در نهایت از میان این بازبین‌های مرتب شده، ۱۰ مورد اول را به عنوان پیش‌بینی‌های سیستم خروجی می‌دهیم. اگر تعداد بازبین‌های پیشنهادی مرتب شده از ۱۰ کم‌تر بود، کل موارد را اعلام می‌کنیم.

خروجی نهایی مساله یک فایل json. است که لیستی از اشیاء[۴۳] را در بر دارد. هر شیء دارای سه خصیصه است: شناسه‌ی تغییر، لیستی از بازبین‌های اصلی درخواست و لیستی از بازبین‌های پیشنهادی پیش‌بینی شده توسط سیستم توصیه‌گر.

۳-۲-۳. ارائه نتایج و تحلیل آن‌ها

در روند اجرای الگوریتم برای تست‌های در نظر گرفته شده، با مشکل سرعت پایین اجرای الگوریتم مواجه بودیم. برای حل آن راه‌های متنوعی را امتحان و بررسی کردیم:

- الگوریتم NSGA-II در پکیج DEAP [۱۰] از پایتون پیاده‌سازی شده است. ساختار این الگوریتم را در پکیج DEAP بررسی کردیم تا اگر به صورت بهینه‌تری الگوریتم را پیاده کرده، از پکیج آماده به جای کدهای این بخش استفاده کنیم. اما روند الگوریتم در پشت پرده کاملاً با الگوریتم ما مطابقت داشت.

[۴۳] Object

- به عنوان راه‌حل دیگر، به بررسی زمان اجرای تک تک قسمت‌های کد پرداختیم. این بررسی بسیاری از قسمت‌های زمان‌بر را نمایان کرد و توانستیم با تکنیک‌هایی زمان اجرای هر بخش را تا جای ممکن کاهش دهیم.

تکنیک‌هایی که بیان شد شامل:

– استفاده از کتابخانه‌ی numpy در سایر بخش‌های کد
– استفاده از مقداردهی تک خطی لیست‌ها به جای حلقه‌های for و append های پی در پی به لیست در داخل حلقه
– استفاده از تعریف مستقیم توابع موجود در کتاب‌خانه‌ها به جای این که ابتدا پکیج فراخوانی شود و سپس تابع خاصی از آن (مثلا استفاده از dot به جای np.dot)
– در تابع شایستگی همکاری، به جای ضرب کردن عبارت $(S[j,i] > 0)$ که بولین است، با یک شرط این کار را انجام دادیم و ۴ بار محاسبه‌ی تابع را سریع‌تر کرد

با این تغییرات، زمان اجرای الگوریتم برای هر تست به طور چشم‌گیری کاهش یافت.

برای ارزیابی سیستم از چند معیار ارزیابی بهره می‌بریم که هر یک به همراه تعریف و رابطه‌ی مربوطه در ادامه آورده شده. K در هر مورد بیانگر تعداد بازبین‌های پیشنهادی است که می‌تواند از ۱ تا ۱۰ باشد[۱۱].

- Precision@k

این مقدار برای هر درخواست برابر است با تعداد جواب‌های صحیح پیش‌بینی شده، تقسیم بر تعداد کل جواب‌های پیش‌بینی شده. درواقع مخرج کسر همواره برابر k می‌باشد. درواقع در این رابطه با افزایش مقدار k دقت کاهش می‌یابد چون مخرج کسر بزرگ‌تر می‌شود. فرمول دقیق آن در رابطه‌ی ۳–۴ آورده شده.

$$Precision@k(r) = \frac{|AR_r \cap RR_r|}{|RR_r|}, \qquad (۴–۳)$$

که در آن AR لیست بازبین‌های اصلی و RR لیست بازبین‌های پیشنهاد شده است.
رابطه‌ی فوق برای محاسبه‌ی Precision هر درخواست است. برای محاسبه‌ی Precision کل باید **میانگین** این مقدار را برای کل درخواست‌های داده‌ی ارزیابی به دست آوریم (رابطه‌ی ۳–۵).

$$Mean\ Precision@k = \frac{1}{|R|}\sum_{r\in R} Precision@k(r)\ . \qquad (۵–۳)$$

- Recall@k

مقدار این معیار برای هر درخواست از رابطه‌ی ۳–۶ به دست می‌آید. مشابه قبل، Recall کل برابر با میانگین Recall هر درخواست تست خواهد بود.

$$Recall@k(r) = \frac{|AR_r \cap RR_r|}{|AR_r|} \ . \qquad (۳–۶)$$

- Top-k Accuracy

این معیار از رابطه‌ی ۳–۷ محاسبه می‌شود که در آن اگر حداقل یکی از k بازبین پیشنهاد شده جزو بازبین‌های اصلی باشد، مقدار isCorrect(r) یک و در غیر این صورت صفر خواهد بود.

$$Accuracy = \frac{1}{|R|}\sum_{r\in R} isCorrect(r) \ . \qquad (۳–۷)$$

- MRR [۴۴]

رابطه‌ی مورد استفاده برای محاسبه‌ی این معیار در رابطه‌ی ۳–۸ آمده که در آن، rank(r) *رتبه‌ی اولین* بازبینی که به درستی پیش‌بینی شده است را بیان می‌کند. اگر هیچ یک از پیش‌بینی‌ها در میان این k بازبین صحیح نباشد، این مقدار *بی‌نهایت* خواهد بود تا مقدار کسر را صفر کند و تاثیری در مقدار نهایی MRR نگذارد. با این توصیف‌ها، هرچه مقدار این معیار بزرگ‌تر باشد، بهتر است.

از آن‌جا که در نام این معیار هم مشخص است، میانگین مقادیر مد نظر است و فرم رابطه به شکلی که در رابطه‌ی ۳–۸ آمده، می‌شود.

$$MRR = \frac{1}{|R|}\sum_{r\in R} \frac{1}{rank(r)} \ . \qquad (۳–۸)$$

معیارهای دیگری نیز مانندMAP و F-Score@k نیز در این مسائل تعریف می‌شوند که ما از ذکر آن‌ها صرف نظرکردیم زیرا مقاله‌ی منتخب، تنها دو مورد Precision و Recall را محاسبه و گزارش کرده است و ما نیز به همین چند مورد بالا اکتفا می‌کنیم. هرچند توجه داشته باشیم که در این مسائل معمولاً Recall، Accuracy و MRR معیارهای بهتری نسبت به Precision هستند. در ادامه دلیل این قضیه را توضیح می‌دهیم.

حال دقت اجرای الگوریتم را برای ۱۰٪ داده‌ها با کمک معیارهای بالا ارزیابی کرده و نتایج را در جدول ۳–۲ آورده‌ایم.

[۴۴] Mean Reciprocal Rank

جدول ۳-۲. نتایج اجرای الگوریتم برای پروژه OpenStack

Top-k	Precision@k	Recall@k	Top-k Accuracy	MRR
1				
3				
5				
10				

مقایسه نتایج با اصل مقاله ....

دترمینیستیک نبودن و به جای ۳۰ بار اجرا ۱ بار ....

نوآوری‌ها ....

۳-۲-۴. جمع‌بندی و نتیجه‌گیری

در پیاده‌سازی مقاله‌ی انتخابی، درمورد جزئیات کار و چالش‌هایی که به آن‌ها مواجه شدیم، توضیح دادیم. گفتیم که در برخی موارد تنها کلیت کار در متن مقاله آمده بود و ما می‌بایست فرضیاتی به آن اضافه می‌کردیم. گاه این فرضیات دقیق و مناسب نبود و باید فرضیات دیگری را جایگزین می‌کردیم. در انتها مشاهده کردیم که سیستم طراحی شده در مقایسه با سیستم‌های توصیه‌گر موجود نتایج خوبی را ارائه می‌کند.

در کنار موارد فوق، ایراداتی نیز بر سیستم توصیف شده در مقاله وارد است؛ از جمله آن که، برای هر درخواست جدید ارسالی، یافتن بازبین‌های مناسب تا حدی زمان‌بر است؛ زیرا برای هر *درخواست*، باید الگوریتم ژنتیک برای *کل درخواست‌های* پروژه تا آن زمان اجرا شود. ایراد دیگر این است که اگر درخواست جدید ارسال شده شامل فایل یا فایل‌هایی باشد که پیش از این مورد بازبینی قرار نگرفته‌اند، سیستم راهکاری برای آن‌ها ندارد چون در تاریخچه وجود ندارند. همچنین مساله‌ی غیرقطعی بودن الگوریتم‌های ژنتیک موجب می‌شود که سیستم نتواند همواره جواب صحیح را کشف کند.

۳-۳. روش ...

۳-۳-۱. تعریف دقیق مساله

۳-۳-۲. بیان حل مساله و روش پیاده‌سازی

۳-۳-۳. ارائه نتایج و تحلیل آن‌ها

۳-۳-۴. جمع‌بندی و نتیجه‌گیری

فصل چهارم – نتیجه‌گیری و پیشنهاد کارهای آتی

۴-۱. مقدمه

۴-۲. خلاصه تحقیق

۴-۳. نتایج تحقیق

۴-۴. پیشنهادهایی برای پژوهش‌های آتی

مراجع

[1] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, K.-i. Matsumoto, Who should review my code? a file location based code-reviewer recommendation approach for modern code review, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, 2015, pp. 141–150.

[2] C. Yang, X.-h. Zhang, L.-b. Zeng, Q. Fan, T. Wang, Y. Yu, G. Yin, H.-m. Wang, Revrec: A two-layer reviewer recommendation algorithm in pull-based development model, Journal of Central South University 25 (5) (2018) 1129–1143.

[3] M. B. Zanjani, H. Kagdi, C. Bird, Automatically recommending peer reviewers in modern code review, IEEE Transactions on Software Engineering 42 (6) (2015) 530–543.

[4] V. Balachandran, Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 931–940.

[5] X. Xia, D. Lo, X. Wang, X. Yang, Who should review this change?: Putting text and file location analyses together for more accurate recommendations, in: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2015, pp. 261–270.

[6] Rebai S, Amich A, Molaei S, Kessentini M, Kazman R. Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations. Automated Software Engineering. 2020 Dec;27(3):301-28.

[7] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2):182 – 197, April 2002.

[8] https://github.com/haris989/NSGA-II/blob/master/NSGA%20II.py

[9] https://medium.com/@rossleecooloh/optimization-algorithm-nsga-ii-and-python-package-deap-fca0be6b2ffc

[10] https://github.com/DEAP/deap/blob/master/examples/ga/nsga2.py

[11] Çetin HA, Doğan E, Tüzün E. A review of code reviewer recommendation studies: Challenges and future directions. Science of Computer Programming. 2021 Apr 14:102652.

فرهنگ واژگان

پیوست‌ها

ABSTRACT

Keywords: Developer, Reviewer, Code Reviewer Recommendation, Recommender System, Machine Learning, Artificial Intelligence

Shahid Beheshti University

Faculty of Computer Science and Engineering

Implementation and Evaluation of Code Reviewer Recommendation Methods

By

Shiva Zeymaran

Supervisor

Dr. Sadegh Aliakbary

January 2022