بازآرایی تکاملی چندهدفه آگاه از انرژی برای تصحیح نشانه‌های کد بد در کاربردهای اندرویدی

مائده زمزمه[۱]، کارشناسی ارشد مهندسی کامپیوتر، سعید صدیقیان کاشی[۲*]، استادیار، امین نیک‌انجام[۳]، استادیار

[۱] دانشکده مهندسی کامپیوتر- دانشگاه صنعتی خواجه نصیرالدین طوسی - تهران - ایران - zamzameh@email.kntu.ac.ir

[۲] دانشکده مهندسی کامپیوتر- دانشگاه صنعتی خواجه نصیرالدین طوسی - تهران - ایران - sedighian@kntu.ac.ir

[۳] دانشکده مهندسی کامپیوتر- دانشگاه صنعتی خواجه نصیرالدین طوسی - تهران - ایران، پژوهشگر - پلی تکنیک مونترال - مونترال - کانادا - nikanjam@kntu.ac.ir

چکیده: در میان مباحث حوزه‌ی مهندسی نرم‌افزار، بهره‌وری انرژی از عوامل موثر در دو مرحله‌ی توسعه و نگهداری نرم‌افزار، مخصوصا در دستگاه‌های باتری‌دار است. انجام بازآرایی نرم‌افزار، اگرچه بهبود کیفی نرم‌افزار را به دنبال دارد، اما برخی از پژوهش‌های اخیر تصریح دارد که اعمال عملگرهای بازآرایی ممکن است به مصرف انرژی بیشتر و یا افزایش زمان اجرای اپلیکیشن‌های اندرویدی منجر شود. در این مقاله، تاثیر بازآرایی و حذف هشت نشانه کد بد و پادالگوی اندرویدی/ جاوایی را بر زمان اجرا، مصرف انرژی و معیارهای کیفی کد بررسی می‌کنیم. برای انجام بررسی‌ها و دریافت نتایج از پنج اپلیکیشن اندرویدی متن‌باز و یک اپلیکیشن اندرویدی توسعه‌داده‌شده، استفاده کردیم. در گام نخست، تغییرهای میزان مصرف انرژی، زمان اجرای اپلیکیشن و معیارهای کیفی کد را پیش و پس از انجام بازآرایی محاسبه کردیم. نتایج نشان می‌دهد اعمال بازآرایی در برخی موارد منجر به کاهش مصرف انرژی و زمان اجرا و در برخی دیگر، افزایش مصرف انرژی و زمان اجرای اپلیکیشن را رقم زده است. در گام دوم برای ارائه پیشنهاد مجموعه‌ای از عملگرهای بازآرایی از میان عملگرهای بازآرایی تشخیص داده شده و ممکن، راهکاری تازه، بااستفاده از راهکار بهینه‌سازی تکاملی چندهدفه ارائه شده است. بر همین اساس، الگوریتم ژنتیک چندهدفه با مرتب‌سازی غیرمغلوب (NSGA-II) را با در نظرگرفتن سه هدف بهبود زمان اجرا، مصرف انرژی و میزان تلاش انجام شده برای بازآرایی، به کار بردیم. خروجی این رویکرد توانسته است میزان زمان اجرا و مصرف انرژی را با دقت میانه ۷۶٪ و ۶۵٪ بهبود دهد و به‌طور میانه ۴۲٪ پادالگوها و نشانه‌های کد بد تشخیص‌داده‌شده در اپلیکیشن‌های اندرویدی را برطرف سازد.

واژه‌های کلیدی: الگوریتم‌های تکاملی، الگوریتم ژنتیک چندهدفه با مرتب‌سازی غیرمغلوب، مهندسی نرم‌افزار مبتنی بر جستجو، بازآرایی مبتنی بر جستجو، پادالگو، نشانه کد بد، انرژی مصرفی، زمان اجرا

* سعید صدیقیان کاشی، **sedighian@kntu.ac.ir**

Energy-Aware Evolutionary Multi-Objective Refactoring for Bad Code Smells Correction of Android Applications

Maede Zamzame [1], M.Sc in computer engineering - software, Saeed Sedighian Kashi [2], Assistant professor, Amin Nikanjam [3], Assistant professor

[1] Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran, Iran, zamzameh@email.kntu.ac.ir

[2*] Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran, Iran, sedighian@kntu.ac.ir

[3] Faculty of Computer Engineering, K. N. Toosi University of Technology, Tehran, Iran,

Research associate, Polytechnique Montréal, Canada, amin.nikanjam@polymtl.ca

Abstract: In software engineering, energy efficiency is an influential factor in the software development and maintenance especially for battery-limited devices. While refactoring can improve the quality of software, recent studies suggest that some refactoring operations may lead to conflicts with energy consumption and execution time of Android applications. In this paper, we analyze the impact of code refactoring for eight Android/Java bad code smells and anti-patterns on a testbed of five real and one synthetic Android applications. We measure energy consumption, execution time and quality design of application before and after refactoring. We then propose a novel refactoring recommendation approach based on evolutionary multi-objective optimization that accounts for energy consumption, execution time and refactoring effort for Android/Java anti-patterns. For this purpose, we use Nondominated Sorting Genetic Algorithm-II (NSGA-II) with three objectives: 1) energy consumption, 2) execution time, and 3) refactoring effort. The obtained results show that this approach can generate refactoring recommendations with a median precision of 65% and 76% for improving energy and execution time respectively while the median of removed antipatterns in testbed applications is 42%.

Keywords: *Software maintenance; Refactoring; Android applications; Anti-patterns; Energy consumption; Execution time; Search-based Software Engineering; Design quality*

*** Saeed Sedighian Kashi, sedighian@kntu.ac.ir**

۱. مقدمه

امروزه به کاربردن اپلیکیشن‌های اندرویدی در دستگاه‌های باتری‌دار که منبع انرژی آن‌ها محدودیت نیز دارد، گسترش پیدا کرده است. در دستگاه‌هایی با این مشخصه، بهینه‌سازی مصرف انرژی اپلیکیشن‌های اندوریدی اهمیت بسزایی پیدا می‌کند [۱]. در مرحله توسعه و نگهداری یک سامانه نرم‌افزاری یا اپلیکیشن اندرویدی، بازآرایی[1] نرم‌افزار می‌تواند به بهبود ویژگی‌هایی مانند قابلیت استفاده مجدد[2]، توسعه‌پذیری[3]، بهبود طراحی و کاهش زمان اجرا از طریق حذف پادالگوها[4] و نشانه‌های کد بد[5] منجر گردد [۲]. در پژوهش‌های پیشین از الگوریتم‌های جستجو و یا پردازش تکاملی برای انجام بازآرایی با اهداف بهبود انرژی، زمان اجرا و امنیت استفاده شده‌است. در پژوهشی نشان داده شده است که بازآرایی می‌تواند تغییر مصرف انرژی در سامانه نرم‌افزاری را رقم زند و در برخی موارد نیز، انجام بازآرایی باعث افزایش میزان مصرف انرژی در آن سامانه شده‌است [۳]. در پژوهشی دیگر، به بررسی تاثیر انجام بازآرایی بر امنیت سامانه‌های نرم‌افزاری پرداخته شد که در برخی موارد انجام بازآرایی، امنیت این سامانه‌ها را کاهش داده‌است [۴]. از آنجایی که هر نوع پادالگو و نشانه‌ی کد بد می‌تواند بر میزان مصرف انرژی و زمان اجرای سامانه‌ی نرم‌افزاری تاثیر متفاوتی نسبت به دیگری بگذارد، نیاز است که این تاثیر برحسب نوع نشانه کد بد و پادالگو بررسی گردد و باتوجه به بهبود یا بدترشدن مصرف انرژی و زمان اجرا، سنجه‌های[6] مرتبط با الگوریتم تکاملی[7] به کار گرفته شده، تنظیم گردد.

در این مقاله ابتدا تاثیر بازآرایی تعدادی نشانه کد بد بر مصرف انرژی، زمان اجرای اپلیکیشن‌ها و معیارهای کیفی کد اندازه‌گیری شد و سپس الگوریتم تکاملی با درنظرگرفتن چندهدف اجرا گشت. ارزیابی‌ها روی پنج اپلیکیشن متن‌باز[8] اندرویدی و یک اپلیکیشن توسعه‌داده‌شده[9]، مطالعه می‌شود. اقدامات انجام‌شده در این روش پیشنهادی را می‌توان به‌صورت زیر خلاصه کرد:

۱. تاثیر بازآرایی روی مصرف انرژی برای هشت نشانه کد بد بررسی شده است. چهار نشانه کد بد و یا توصیه بازآرایی (LIC، arrayLoops ، slowerToArrayCall و listCreationByLoop) تاکنون در کارهای مشابه بررسی نشده‌بودند.

۲. تغییرهای زمان اجرا را به‌ازای بازآرایی هشت نشانه کد بد اندازه‌گیری کردیم.

۳. تغییرهای معیارهای کیفی کد نیز به‌ازای بازآرایی پادالگوها، سنجیده شده است.

۴. با بازآرایی هشت نشانه کد بد، در قدم اول، اهداف مسئله بهینه‌سازی[10] را بهبود مصرف انرژی و زمان اجرای اپلیکیشن‌ها در نظر گرفتیم، سپس بهبود میزان تلاش برای بازآرایی نیز به همراه دو هدف اشاره‌شده در الگوریتم تکاملی تعریف شد. با به کاربردن الگوریتم ژنتیک چندهدفه با مرتب‌سازی غیرمغلوب[11] نسخه‌ی ۲ به تولید زیرمجموعه‌هایی از عملگرهای بازآرایی مورد بررسی، پرداخته‌شد.

[1] Refactoring
[2] Reusability
[3] Extendibility
[4] Anti-pattern
[5] Bad code smells
[6] Metric
[7] Evolutionary algorithm

[8] Open Source
[9] Synthetic
[10] Optimization problem
[11] Non-dominated Sorting Genetic Algorithm (NSGA-II)

۵. در ادامه به مقایسه میزان تغییرات انرژی و زمان اجرای به‌دست‌آمده به‌ازای حذف نشانه‌های کد بد با کارهای مشابه پرداختیم و دقت تخمین تغییرات انرژی و زمان اجرا برای زیرمجموعه‌ای از عملگرهای بازآرایی که خروجی الگوریتم است، به‌ترتیب با میانه دقت ۶۵٪ و ۷۶٪ به‌دست آمد. در این راهکار، بطور میانه ۴۲٪ پادالگوهای موجود در اپلیکیشن‌ها حذف شد. هم‌چنین با تغییر پارامترهای مربوط به الگوریتم تکاملی مانند اندازه جمعیت، نرخ انتخاب والدین و حداکثر تعداد نسل به مقایسه نتایج روی معیار HV[12] پرداختیم. HV یک شاخص عملکرد[13] است که در الگوریتم‌های چندهدفه استفاده می‌شود و با استفاده از آن، فضای میان مجموعه راه‌حل‌های تولیدشده توسط الگوریتم نسبت به یک نقطه مرجع اندازه‌گیری می‌گردد. بیشتربودن عدد HV به معنای افزایش پراکندگی و همین‌طور تنوع در مقادیر برازش راه‌حل‌های خروجی است.

در بخش دوم پیش‌زمینه و مفاهیم مرتبط با روش پیشنهادی تعریف شده‌است و در بخش سوم به مرور کارهای مرتبط با این حوزه پرداخته شد. بخش چهارم به توضیح جزئیات روش پیشنهادی، اختصاص یافته و مقایسه نتایج در بخش پنجم آمده است. جمع‌بندی و کارهای آتی در بخش ششم ذکرشده‌است.

۲. پیش‌زمینه

۱،۲. مهندسی نرم‌افزار

مهندسی نرم‌افزار، به دنبال حل مسئله‌های حیطه‌ی نرم‌افزار است، مسائلی که اهدافشان رقابت‌کننده و احتمالا متناقض نیز هست. از آن‌جایی که این مسائل، مجموعه وسیع و گوناگونی از راه‌حل‌ها را دارند، می‌بایست پاسخ‌هایی ارائه شود که مصالحه میان این اهداف برقرارشود. در ادامه و به‌عنوان نمونه به پنج سوال قابل بررسی در حوزه مهندسی نرم‌افزار اشاره‌کرده‌ایم [۵]:

- بهترین راه‌حل که قابلیت نگهداری[14] را در ساختار معماری یک سامانه افزایش دهد، چیست؟
- مجموعه کمینه‌ای از موارد آزمون[15] که تمام بخش‌های کارکردی سامانه را پوشش‌دهد، کدام است؟
- در استقرار و توسعه سامانه، بهترین تخصیص منابع به چه نحوی می‌تواند باشد؟
- بهترین دنباله بازآرایی که می‌بایست برای یک سامانه‌ی نرم‌افزاری اعمال شود، چیست؟
- مجموعه نیازمندی‌هایی که می‌تواند میان هزینه توسعه سامانه و همین‌طور رضایتمندی مشتری تعادل ایجاد کند، کدام است؟

همان‌طور که دامنه سوال‌ها هم نشان می‌دهد، مباحث مورد بررسی در مهندسی نرم‌افزار، حوزه وسیعی از بازآرایی، طراحی، آزمون و مهندسی نیازمندی‌ها و... را در بر می‌گیرد که به‌دلیل امکان رقابت‌پذیری و تناقض، از دسته مسائل بهینه‌سازی به شمار می‌آید.

۲،۲. مهندسی نرم‌افزار مبتنی بر جستجو[16]

با توجه به ماهیت بهینه‌سازی مسائل حوزه مهندسی نرم‌افزار، استفاده از الگوریتم‌های مبتنی‌بر جستجو کاربرد گسترده‌ای دارد؛

[12] Hyper Volume (HV)
[13] Performance indicator

[14] Maintainability
[15] Test case
[16] Search-Based Software Engineering (SBSE)

چراکه استفاده از این الگوریتم‌ها منجر به کاهش زمان ارائه پاسخ و هم‌چنین بهبود نتایج می‌شود.

۳،۲. بازآرایی

بازآرایی به‌معنای انجام تغییرهایی است که باعث بهبود کیفیت نرم‌افزار می‌گردد که می‌تواند در هر دو مرحله توسعه و نگهداری نرم‌افزار انجام پذیرد. بازآرایی، ویژگی‌های کیفی[۱۷] مانند قابلیت توسعه‌پذیری[۱۸]، قابلیت نگهداری و کارایی[۱۹] را ارتقا دهد. بازآرایی در سطوح مختلفی مانند کد و معماری می‌تواند انجام گردد. منظور از بازآرایی نرم‌افزار، تغییر ساختار سامانه‌ی نرم‌افزاری با حفظ رفتار قابل مشاهده و منطق آن سامانه است [۱].

۴،۲. بازآرایی مبتنی‌بر جستجو[۲۰]

با درنظرگرفتن مسائلی که در حوزه‌ی بازآرایی نرم‌افزار، ماهیت بهینه‌سازی دارند و همچنین، ضرورت خودکارسازی یافتن این راه‌حل‌ها، برای کاهش میزان هزینه‌ها، مفهوم بازآرایی مبتنی بر جستجو مطرح می‌گردد. پیداکردن بهترین دنباله از میان همه عملگرهای بازآرایی ممکن، کاری سخت است. از یک طرف یافتن چنین دنباله‌ای می‌تواند با ویژگی‌های کیفی متنوعی هم‌بستگی داشته باشد که ممکن است انجام بازآرایی، برروی آن ویژگی‌ها تاثیر منفی داشته و منجر به کاهش کیفیت سامانه گردد و از طرفی دیگر، بهترین دنباله بازآرایی معمولا باید از میان طیف گسترده‌ای از عملگرهای بازآرایی انتخاب شود. از این رو در بازآرایی نرم‌افزار از الگوریتم‌های جستجو استفاده می‌شود [۱].

۵،۲. نشانه‌های کد بد و پادالگوها

نشانه های کد بد به مفهوم وجود ساختارهایی خاص در کد است که می‌تواند هشدار نقض اصول اساسی در طراحی آن سامانه نرم افزاری باشد و ممکن است اثری نامطلوب بر کیفیت طراحی سامانه بگذارد. وجود نشانه کد بد، بیانگر این است که ممکن است در آن قسمت از کد، بازآرایی ضروری باشد. برطرف کردن هر نشانه بد در کد، روش‌های و عملگرهای بازآرایی مربوط به خود را نیاز دارد و وجود نشانه‌های بد در کد، می‌تواند امکان استفاده مجدد از کد و یا توسعه سامانه را دشوار سازد [۶]. پادالگو، به مفهوم استفاده متداول الگویی در شرایط نادرست است؛ به عبارتی، الگو به مفهوم ارائه راه حل مناسب برای یک مسئله و به منظور طراحی بهتر آن است؛ حال آنکه پادالگو دقیقا مفهوم متضاد الگو را دارد به عبارتی، پادالگو به معنای ارائه راه حلی برای یک مسئله است که منجر به طراحی ضعیف سامانه می‌گردد [۷].

۶،۲. الگوریتم ژنتیک چند هدفه با مرتب‌سازی غیر مغلوب نسخه ۲ (NSGA-II)

یکی از الگوریتم‌های تکاملی، الگوریتم NSGA-II است. الگوریتم‌های تکاملی زیرمجموعه‌ای از الگوریتم‌های جستجو هستند که با الهام از طبیعت و همین‌طور با استفاده از تکنیک‌های جستجو، ایجاد جمعیت اولیه، انتخاب والد[۲۱] و انجام عملگرهای هم‌برش[۲۲]، جهش[۲۳] و... با تعیین مقدار تابع برازش[۲۴] را منجر می‌شود و برای حل مسائل بهینه‌سازی استفاده می‌گردد [۱].

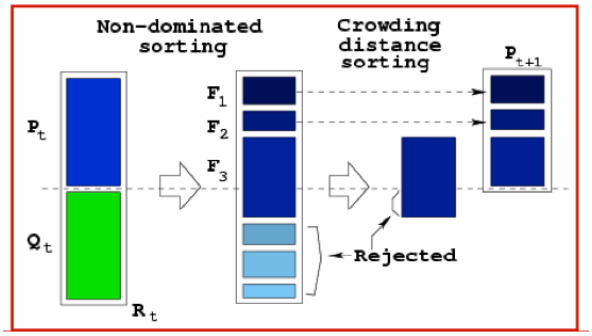
تمایز الگوریتم ژنتیک با این الگوریتم در سازوکارهایی مانند مرتب‌سازی غیرمغلوب است. الگوریتم‌های چندهدفه با مرتب‌سازی غیرمغلوب که پیش از ارائه الگوریتم فعلی، استفاده می‌شدند، چالش‌هایی مثل پیچیدگی محاسباتی بالا، حفظ‌نکردن

[17] Quality attribute
[18] Extensibility
[19] Performance
[20] Search-based refactoring (SBR)

[21] Parent selection
[22] Cross over
[23] Mutation
[24] Fitness function

اعضای نخبه و نیاز به مشخص‌کردن پارامترهای اشتراک داشتند. الگوریتم فعلی راهکارهایی برای حفظ اعضای نخبه، پیشرفت به سمت جبهه بهینه[۲۵] و حفظ تنوع در انتخاب راه‌حل‌ها در طول اجرای الگوریتم را در نظر گرفته است.

شکل ۱ روند کلی اجرای الگوریتم NSGA-II را نشان‌می‌دهد، در هر بار اجرای این الگوریتم، ابتدا از جمعیت کنونی با اجرای الگوریتم‌هایی برای انتخاب والدین، مجموعه‌ای از شایسته‌ترین والدین انتخاب می‌گردد، سپس با اجرای عملگرهایی مانند جهش و هم‌برش، فرزندان تولید می‌شوند، جمعیت حاصل بعد از انجام مرتب‌سازی غیرمغلوب[۲۶]، براساس رتبه[۲۷] به چند جبهه[۲۸] تقسیم می‌شود. سپس بهترین راه‌حل‌ها از جبهه‌های پیشین انتخاب شده و فاصله ازدحام[۲۹] و اجرای سازوکارهای باقی‌ماندن اعضا[۳۰] براساس فاصله ازدحام و مرتب‌سازی غیرمغلوب محاسبه می‌گردد و الگوریتم با همین روال ادامه می‌یابد [۸].



شکل (۱): روال اجرای الگوریتم NSGA-II [۸]

مرتب‌سازی غیرمغلوب به این مفهوم است که هریک از اعضای جمعیت بر حسب مقدار تابع برازشی که به‌ازای هرهدف دارند، با مقدار تابع برازش اعضای دیگر برای همان هدف مقایسه می‌شود؛ چنانچه هیچ کدام از مقادیر توابع برازش یک عضو از عضو دیگر بدتر نباشد (یعنی با آن مساوی یا مقدارش بهتر از آن باشد) عضو یادشده بر عضو مورد مقایسه غلبه پیدا می‌کند. برای هرعضو، مجموعه‌ای از اعضا که آن عضو بر آن‌ها غلبه کرده، محاسبه می‌گردد. پس از آن، براساس تعداد اعضای مجموعه‌ی مذکور، رتبه‌ی هرعضو محاسبه می‌گردد و با استفاده از این راهکار، جبهه‌بندی همه اعضا انجام می‌گیرد.

۳. کارهای مرتبط

در این بخش به مرور کارهای انجام‌شده در زمینه‌های استفاده از راهکارهای هوش مصنوعی در فرآیند توسعه یا خطایابی نرم‌افزار، بازآرایی مبتنی‌بر جستجو و بررسی تاثیر بازآرایی نشانه‌های کد بد و پادالگوها بر مصرف انرژی و زمان اجرای سامانه‌ها می‌پردازیم.

در سال ۲۰۱۹ از الگوریتم NSGA-II برای شناسایی مولفه‌های نرم‌افزاری[۳۱] استفاده شده است که هدف مد نظر در این پژوهش، نگاشت مسئله تشخیص مولفه های نرم افزاری به مسئله بهینه سازی چندهدفه است. روش پیشنهادی در این مقاله برحسب معیارهای انسجام، اتصال و پیچیدگی است و بین این معیارها به منظور تشخیص مولفه های مناسب مصالحه انجام می‌شود. نتایج ارزیابی در این پژوهش نشان‌دهنده‌ی این است که استفاده از الگوریتم چندهدفه پیشنهادی توانسته بهتر از روش های تک‌هدفه عمل کند [۱۹].

در سال ۲۰۱۳ از راهکارهای یادگیری ماشین برای مکان‌یابی خطاهای پنهان نرم‌افزارها استفاده شده است. در این پژوهش، هدف، ارائه راهکاری برای تعیین خودکار محدوده خطاهای پنهان در متن برنامه‌های نرم‌افزاری می‌باشد. جهت به دست آوردن شباهت مسیرها، مدل های-N گرام اجراها محاسبه شده و سپس با استفاده از آنتروپی متقاطع، شباهت بین این مدل‌ها

[25] Pareto-optimal front
[26] Non-dominated sorting
[27] Rank
[28] Front
[29] Crowding distance
[30] Survive selection
[31] Software component

محاسبه گردید. با تحلیل هر دسته، با کمک آنتروپی متقاطع، مجموعه‌ای از مکان‌های مشکوک به خطا شناسایی می‌شوند و در نهایت با استفاده از رای اکثریت بین دسته‌ها، مکان های مشکوک به خطا به صورت بخش هایی از یک زیرمسیر معرفی خواهدشد [۲۰].

سال ۲۰۱۶ نیز در پژوهشی از رویکردهای یادگیری ماشین برای تخمین هزینه‌های نرم‌افزار در مرحله‌ی توسعه‌ی نرم‌افزار استفاده شد. از آنجایی که تخمین هزینه‌ی نرم‌افزار امری ضروری در فرآیند توسعه نرم‌افزار است، استفاده از این روش برای تخمین دقیقتر هزینه‌ها ضروری می‌نماید [۲۱].

۱،۳. کارهای مرتبط با حوزه بازآرایی مبتنی‌برجستجو

در سال ۲۰۱۶ راهکار چندهدفه EARMO[32] ارائه‌شد که مصالحه‌ای میان دو هدف بهبود کیفیت طراحی از طریق حذف تعداد بیشینه پادالگوهای شی‌گرایی و اندرویدی و بهبود میزان مصرف انرژی را فراهم می‌کرد. در EARMO، سه الگوریتم چندهدفه MOCell[33]، SPEA-II[34] و NSGA-II به‌صورت مستقل استفاده‌شد و کارایی هر یک با دیگری مقایسه‌شد. در EARMO، تاثیر بازآرایی هشت پادالگوی شیءگرای جاوایی و اندرویدی موثر بر مصرف منابع دستگاه‌های اندرویدی، بررسی شده‌است. برای بررسی میزان تاثیر پیشنهادهای بازآرایی تولیدشده به‌واسطه الگوریتم‌ها، نظر تعدادی از توسعه‌دهندگان اپلیکیشن‌های اندرویدی درخواست شده‌بود که براساس آن ۶۸٪ پیشنهادات بازآرایی ارائه‌شده، مرتبط بوده و ۸۴٪ پادالگوها نیز با پیشنهادهای بازآرایی ارائه‌شده تصحیح شده‌بود [۳].

در سال ۲۰۱۷ بخاری و همکاران به بررسی استفاده از الگوریتم‌های تکاملی برای تعیین مقادیر برخی پارامترها با هدف بهبود مصرف انرژی پرداختند. در این روش بر روی بهینه‌سازی مقدارdeep parameter با کمینه‌سازی مصرف انرژی پژوهش شده بود. برای نمونه در این پژوهش بررسی شد که از پکیج جاوایی java.util.collection کدام ساختارداده hashset یا arrayList انتخاب گردد تا همان کاربرد را در پیاده‌سازی کد داشته‌باشد اما میزان مصرف انرژی آن، بهینه‌تر باشد. برای بازآرایی، تغییرات روی فایل پیکربندی اعمال می‌شد و الگوریتم NSGA-II با درنظرگرفتن دو هدف کمینه‌سازی مصرف انرژی و همین‌طور کمینه‌سازی اختلاف از مقادیر محاسبه‌شده توسط کتابخانه یادشده به ازای نمونه‌های آزمون از معین‌شده‌ی موجود در مخزن oracle اجرا شد. خروجی الگوریتم مورد استفاده، مجموعه‌ای از تنظیمات و تغییرات آن با رعایت مصالحه بین دو هدف مذکور بود. مصرف انرژی با راهکارهای سخت‌افزاری و استفاده از برخی مکانیزم‌های داخلی گوشی موبایل و همین‌طور تراشه‌ای به منظور محاسبه مصرف انرژی و دمای دستگاه اندازه‌گیری شد. نتایج نشان می‌داد که استفاده از الگوریتم‌های جستجوی به‌کارگرفته‌شده، منجر به بهبود مصرف انرژی شده‌است [۹].

در سال ۲۰۱۸ مانوتاس و دیگران طی پژوهشی، بررسی کردند که استراتژی‌های جستجو همچون الگوریتم ژنتیک چگونه می‌تواند به بهبود مصرف انرژی در اپلیکیشن‌های اندرویدی کمک کند. در این کار به منظور پیداکردن مجموعه‌ای از APIها با انرژی مصرفی بهینه‌تر و کمتر از چارچوبی به اسم SEEDS استفاده‌شد و دو الگوریتم NSGA-II و gGA[35] را برای یافتن این مجموعه استفاده کردند. اهداف تعریف‌شده در این مسئله، بهینه‌سازی زمان ارائه راه حل و پیشنهاد مجموعه APIها با مصرف انرژی کمتر بود. برای اندازه‌گیری انرژی از راهکارهای نرم‌افزاری و یک ابزار تخمین مصرف انرژی با نام RAPL که با اجرای کد، به تخمین انرژی می‌پرداخت، استفاده شده‌بود. نتایج

[32] Energy-Aware Refactoring approach for Mobile apps
[33] Multi Objective Cellular genetic algorithm
[34] Strength Pareto Evolutionary Algorithm
[35] generational Genetic Algorithm (gGa)

نشان می‌داد که در بیشتر موارد، خروجی الگوریتم جستجو منجر به کاهش مصرف انرژی شده‌بود [۱].

۲,۳. کارهای مرتبط با بررسی تاثیر بازآرایی پادالگوها بر مصرف انرژی، زمان اجرا

در سال ۲۰۱۶ هکت و همکاران سه پادالگوی اندوریدی MIM، IG و IDS و همین‌طور تاثیری که رفع آن‌ها از طریق بازآرایی بر روی کارایی سامانه‌های اندرویدی می‌گذارد را بر روی دو اپلیکیشن اندرویدی متن‌باز بررسی کردند. به منظور بررسی کارایی رویکرد ارائه شده، چهار سنجه به کار گرفته شد که Delayed Frame و Frame Time دوسنجه‌ی تعریف‌شده برای میزان تاخیر نمایش یک فریم در رابط کاربری بوده، از GC calls به معنای تعداد فراخوانی‌های Collection Garbage و Memory Usage به معنای مقدار استفاده از حافظه بودند. اندازه‌گیری چهار سنجه مذکور بااستفاده از خروجی برخی آرگومان‌های ADB یعنی Gfxinfo، Logcat و Meminfo محاسبه می‌شود. نتایج نشان می‌داد که در همه موارد، حذف پادالگوها از طریق بازآرایی باعث کاهش Frame Time شده اما تغییرات مربوط به مقادیر سه سنجه دیگر، در برخی موارد، افزایشی و در برخی دیگر کاهشی بوده‌است [۱۰].

در سال ۲۰۱۸ باراک و دیگران بررسی کردند که در زمان بازآرایی یک سامانه، به‌کاربردن هریک از عملگرهای بازآرایی و در نتیجه حذف پادالگوها و نشانه‌های کد بد شی‌گرایی جاوایی، چه تاثیری بر سرعت اجرا، میزان انرژی مصرفی و توان مصرفی سامانه‌ی نرم‌افزاری دارد. برای بررسی دقیق‌تر این تاثیر از مجموعه معیارهایی با عنوان "GPS-UP" به تفکیک هرعملگر بازآرایی مورد مطالعه در این رویکرد، استفاده کردند. معیارهای به کار گرفته‌شده برای ارزیابی و بررسی عملگرهای بازآرایی متناظر با پادالگوها و نشانه‌های کد بد، به صورت نسبت مقدارهای قبل و بعد از انجام بازآرایی آن، برای معیار مورد بررسی، تعریف شده‌بود:

- Green-up: نسبت میزان تغییر مصرف انرژی پس از بازآرایی
- Speed-up: نسبت میزان تغییر زمان اجرای اپلیکیشن پس از بازآرایی
- Power-up: نسبت میزان تغییر توان پس از بازآرایی

بعد از اعمال هر عملگر بازآرایی و سنجش معیارهای GPS-UP، هریک از روش‌های بازآرایی براساس مقدار سه معیار محاسبه‌شده، در فضایی سه بعدی نگاشت شدند. به‌منظور اندازه‌گیری انرژی، زمان و توان مصرفی برای هرعملگر بازآرایی، نمونه پیاده‌سازی‌شده‌ی هر کدام از عملگرهای بازآرایی را در قالب فراخوانی قسمتی از یک اپلیکیشن اندرویدی سنتزشده، اجرا کردند و محاسبات مربوط بر اساس آن، انجام شد. طبق نتایج به‌دست آمده برای این عملگرهای بازآرایی، آن‌ها در فضایی سه بعدی نگاشت شدند و براساس نتایج، برخی عملگرهای بازآرایی در یک یا دومعیار از میان معیارهای بهبود مصرف انرژی، توان مصرفی و یا زمان اجرا، تعارض داشتند به‌عنوان مثال مقدار یک معیار بهبود یافته ولی معیار دیگر بدتر شده بود [۱۱].

در سال ۲۰۱۹ پالومبا و همکاران به بررسی ارتباط حذف ۹ نشانه کد بد اندرویدی و تاثیرش بر تغییر مقادیر مصرف انرژی پس از انجام بازآرایی پرداختند. این بررسی روی تعدادی از اپلیکیشن‌های اندرویدی موجود انجام شد. در گام اول، با محاسبه مصرف انرژی اپلیکیشن، پیش از بازآرایی، و سپس با تشخیص و برطرف کردن نشانه‌های کد بد شناخته‌شده، مصرف انرژی را این بار پس از انجام بازآرایی و با تکرار همان سناریوی اجرایی در اپلیکیشن، محاسبه‌کردند. برای اندازه‌گیری مصرف انرژی از رویکرد نرم‌افزاری و ابزاری به نام PETrA استفاده

شد. نتیجه این پژوهش نشان داد، از میان نه نشانه کد بد بررسی‌شده، حذف چهار نشانه کد بد یعنی LT، IS، MIM و SL به‌میزان قابل ملاحظه‌ای مصرف انرژی را بهبود داده‌است [۱۲].

۴. روش پیشنهادی

روش پیشنهادی از دو قسمت اصلی تشکیل می‌شود. در بخش نخست و طبق شکل ۲ هشت نشانه‌ی کد بد و پادالگوی اندرویدی و جاوایی (معرفی‌شده در جدول ۱) بررسی شدند و پرسش اصلی این بود که با اجرای سناریوهایی با قابلیت تکرار، بازآرایی هریک از نشانه‌های کد بد و پادالگوهای مورد مطالعه چه تاثیری بر مصرف انرژی، زمان اجرا و معیارهای کیفی کد در آن اپلیکیشن دارد؟ براین اساس، انرژی، زمان اجرا و معیارهای کیفی کد با به‌کاربردن ابزارهایی اندازه‌گیری شد. براساس مقادیر تغییرهای مرتبط با انرژی، زمان اجرا و معیارهای کیفی کد، مقدار سنجه‌های متناظر با هرکدام از این سه، در الگوریتم تکاملی تنظیم شد. در بخش دوم و با توجه به شکل ۳، ابتدا یک اپلیکیشن اندرویدی توسعه‌یافته با زبان جاوا، دریافت شده و بااستفاده از دو ابزار Paprika [۱۳] و PMD [۱۴] وجود پادالگوها و نشانه‌های کد بد احتمالی مشخص شد و براساس آن با اجرای الگوریتم NSGA-II همه پیشنهادهای بازآرایی ممکن تولیدشد.

شکل (۲): روند اجرای قسمت نخست (بخش نرم‌افزاری): بررسی تاثیر بازآرایی پادالگوها بر تغییر میزان انرژی و زمان

براساس مقادیر سنجه‌های مربوط به تغییر مقادیر مصرف انرژی و زمان اجرای هر نشانه کد بد و مجموعه پیشنهادهای بازآرایی، فضای مسئله تعریف گشت. سپس الگوریتم NSGA-II را یکبار با دو هدف و سپس با درنظرگرفتن کمینه‌سازی سه هدف برای میزان مصرف انرژی و زمان اجرا و میزان تلاش برای بازآرایی، اجرا کردیم.

شکل (۳): روند اجرای قسمت دوم (بخش مربوط به هوش مصنوعی): اجرای الگوریتم NSGA-II

۱،۴. پادالگوها و بازآرایی

پادالگوها و بازآرایی‌های متناظر بررسی‌شده در روش پیشنهادی در جدول ۱ عنوان شده‌است. در EARMO [۳] نیز مانند روش پیشنهادی، تاثیر بازآرایی بر میزان مصرف انرژی برای دو پادالگوی IDS یا HashMapUsage و IGS بررسی شد؛ اما در EARMO برای تخمین میزان انرژی از روش سخت‌افزاری استفاده شده و میزان مصرف انرژی به‌ازای اجرای یک متد تخمین‌زده نمی‌شد و از طرفی در EARMO بر تاثیر بازآرایی بر زمان اجرای اپلیکیشن تمرکز نشده؛ به‌این دلیل این دو پادالگو در روش پیشنهادی ما نیز مجددا بررسی شدند. بررسی تاثیر بازآرایی بر میزان مصرف انرژی پادالگوهای IDS، MIM، SL و IS ذکرشده در پژوهش پالومبا با روش پیشنهادی حاضر، مشترک و نحوه محاسبه انرژی نیز همانند روش پیشنهادی حاضر بوده اما در پالومبا تاثیر بازآرایی بر زمان اجرا بررسی نشد.

۲،۴. اپلیکیشن‌های مورد بررسی

در روش مورد مطالعه در این مقاله، ۱۱ اپلیکیشن اندرویدی متن‌باز که پوشه کد و سایر مستندهای مربوط به آن در مخزن github موجود بود و یا فایل apk. آن اپلیکیشن در مخزن Fdroid [۱۵] در دسترس است، بررسی گشت و از میان آن‌ها پنج اپلیکیشن، برگزیده شده و فیلتر شده و در مرحله تحلیل پادالگوها مطالعه شد. شاخص‌های مهم در فیلترکردن و انتخاب پنج اپلیکیشن شامل موارد زیر است:

- در نظر گرفتن تنوع در اندازه کد مربوط به هر اپلیکیشن (کوچک، متوسط و بزرگ)
- رعایت تنوع در کاربرد و موضوع هر اپلیکیشن: مانند بازی، مدیریت گوشی، ریاضیات و...
- توجه به معتبربودن وابستگی‌ها و کتابخانه‌های استفاده‌شده در کد اپلیکیشن
- لزوم دسترسی به نسخه‌ای از اپلیکیشن که با کد موجود در مخزن‌هایی مانند github همگام است.

علاوه‌بر پنج اپلیکیشن مذکور، اپلیکیشنی نیز شامل همه موارد وقوع نشانه‌های کد بد و بازآرایی‌هایشان توسعه یافت تا دقت تحلیل تغییرات انرژی و زمان اجرا بیشتر شود.

جدول (۱): پادالگوها و نشانه‌های کد بد بررسی‌شده و بازآرایی آن‌ها

نام پادالگو و نشانه کد بد	بازآرایی متناظر
MIM (Member Ignoring Method)	Make method static
IDS (Inefficient Data Structure)/ HMU (HashMap Usage)	Replace HashMap data structure with ArrayMap
IGS (Internal Getter / Setter)	Assign values of these variables directly
SL (Slow Loop)	Replace for-loop with for-

each	
Make inner class static	LIC (Leaking inner class)
Use Arrays.asList() in package: java.util.Arrays	List creation from array by loop
call ToArray() method in proper argument: (referred to element 0 of the array	slowerToArrayCall
Use Arrays.copyOf or System.arraycopy instead of for-loop to copy an array	Array Loops

۳،۴. محاسبه زمان اجرا، مصرف انرژی و معیارهای کیفی

در نخستین قسمت از رویکرد پیشنهادی، برای محاسبه انرژی مصرفی و زمان اجرا، از راهکار و رویکرد نرم‌افزاری و پروفایلری با نام PETrA [۱۲] برای تخمین میزان زمان اجرا و مصرف انرژی هرمتد جاوایی فراخوانی‌شده در سناریوی اجرایی، استفاده کردیم. برای اجرای این ابزار، یک فایل سناریوی اجرایی اپلیکیشن دریافت، اجرا و انرژی وزمان اجرای متدهای فراخوانی‌شده در آن سناریو تخمین زده می‌شود. برای ارزیابی و مقایسه دقت محاسبه زمان اجرا توسط ابزار PETrA، از کتابخانه Debug [۱۶] نیز استفاده کردیم. هم‌چنین از ابزار RefGen [۱۷] برای اندازه‌گیری معیارهای کیفی کد همچون قابلیت استفاده مجدد[36]، میزان تاثیر[37]، قابل فهم‌بودن کد[38]، توسعه‌پذیری[39] و انعطاف‌پذیری[40] استفاده شد. در بخش ۵ به نتایج این بررسی پرداخته شده‌است.

۴،۴. خودکارسازی سناریوهای اجرایی

در روش پیشنهادی ، ابزار monkeyrunner [۱۸] را برای تکرارپذیری سناریوهای اجرایی به‌کار گرفتیم. این ابزار به‌عنوان ورودی، یک فایل به زبان پایتون را دریافت می‌کند که می‌تواند شامل دستورهایی مانند نصب و یا لغو نصب اپلیکیشن، ، فشاردادن دکمه‌ای با مختصات معین، گرفتن عکس از صفحه نمایش دستگاه و ایجاد وقفه‌های چندثانیه‌ای و... باشد. ابزار PETrA یک فایل متنی را که شامل مختصات دکمه‌های موردنظر و وقفه‌های میان هرعمل است را دریافت می‌کند و با کمک دو اسکریپت به زبان پایتون، شامل چند تابع برای تنظیم آرگومان‌های مربوط، ابزار را فراخوانی می‌کند. جدول‌های ۲، ۳، ۴، ۵ و۶ شامل پادالگوهای تشخیص‌داده‌شده در هر اپلیکیشن و سناریوی اجرایی متناظر آن است.

جدول (۲): سناریوهای اجرایی در اپلیکیشن Hacker keyboard

نام اپلیکیشن	نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
Hacker keyboard	LIC	تایپ یک کلمه و ذخیره آن برای به کاربردن در پیشنهادهای آتی/ isValidWord
	IDS	تایپ بخشی از یک کلمه و برگزیدن یکی از پیشنهادها/ setSeggustions
	List creation from array by loop	نمایش فهرستی از زبان‌ها در قسمت تنظیمات / InputLanguageSelection

جدول (۳): سناریوهای اجرایی در اپلیکیشن Hot death

نام اپلیکیشن	نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
Hot death	SL	هرسه پادالگو، سناریوی اجرایی مشترکی دارند. بازکردن اپلیکیشن و انتخاب اولین حرکت در بازی / :ArrayLoops
	ArrayLoops	
	IDS	

[36] Reusability
[37] Effectiveness
[38] Understandability
[39] Extendibility
[40] Flexibility

		CardDeck GameTable :IDS constructor GameTable :SL constructor

جدول (۴): سناریوهای اجراشده‌ی مربوط به اپلیکیشن‌های Password maker و Anycut

نام اپلیکیشن	نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
Anycut	LIC	انتخاب میان‌بر جدید، سپس activity جدید و نمایش لیست activity / ActivityPicker
	IG	در قسمت system equation آرایه‌ای با ورودی تصادفی را وارد کنیم و محاسبات انجام می‌گیرد./ itemMatrix -> solve
Password maker	MIM	در منوی کناری، قسمت تنظیمات، رفتن به بخش import/export ImportExportRdf

جدول (۵): سناریوهای اجراشده در اپلیکیشن Matrix Hamilton

نام اپلیکیشن	نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
Matrix Hamilton	LIC	فشردن دکمه مرتبط با محاسبات پایه‌ای ماتریس و سپس تعیین ابعاد و نوع آرایه/ Adapter result
	MIM	نخست در بخش solve یک basic matrix ماتریس را وارد و سپس محاسبات مربوط انجام می‌گردد و در ادامه، انتخاب عملگر جمع آرایه و تعیین ابعاد آرایه جدید/ getCheckedMatrix
	SL	فشردن دکمه مرتبط با محاسبات پایه‌ای ماتریس ، تعیین ابعاد و نوع آرایه و در نهایت نمایش نتیجه محاسبات/ fragmentResult
	IS	انتخاب دکمه مربوط به محاسبات پایه‌ای ماتریس و تعیین نوع و ابعاد آرایه تصادفی، نمایش نتیجه محاسبات/ sampleMatrixDialog

جدول (۶): سناریوهای اجرایی در اپلیکیشن androSmell

نام اپلیکیشن	نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
androSmell	IDS	فشردن HMU_button / hmu_test -> IDS.hmu و بازآرایی آن
	LIC	فشردن LIC_button / LIC_test -> getEnclosedClassObject و بازآرایی آن
	SL	فشردن SL_button / SL_test -> sl.loop و بازآرایی آن
	ToArrayCall	انتخاب arrayCall_button/ ArrayRelatedSmells ->

		toArrayCall و بازآرایی آن
	MIM	فشردن MIM_button / mim_test -> mim.mimMethod و بازآرایی آن
	IS	انتخاب IGS_button / IGS -> igs.initializeIS و بازآرایی آن
	List creation from array by loop	انتخاب asList_button/ ArrayRelatedSmells -> arrayAsList و بازآرایی آن
	Array Loops	انتخاب arrayLoops_button/ ArrayRelatedSmells -> arrayLoops و بازآرایی آن

۵,۴. اجرای الگوریتم NSGA-II

در روش پیشنهادی، نسخه استاندارد الگوریتم NSGA-II را به‌کارگرفتیم. ابتدا این الگوریتم با دوهدف بهبود مصرف انرژی و زمان اجرایی اپلیکیشن، اجرا و سپس هدف بهبود میزان تلاش برای بازآرایی نیز به دو هدف مذکور، افزوده و الگوریتم مجددا اجرا شد.

۱,۵,۴. توصیف فضای مساله[41]

مجموعه داده‌های مرتبط با یک اپلیکیشن اندرویدی به عنوان ورودی دریافت گشت، با اجرای ابزارهای تشخیص پادالگو، پادالگوها تشخیص داده شد و معیارهای مورد نیاز، اندازه‌گیری شد. زیرمجموعه‌هایی از تمام پیشنهادهای بازآرایی شناسایی‌شده و معتبر، به‌ازای داده‌های اپلیکیشن ورودی می‌توانست راه‌حل و خروجی الگوریتم باشد. به‌عبارت دقیق‌تر هرپاسخ در خروجی الگوریتم اجرایی، مجموعه‌ای است از پیشنهادهای بازآرایی با اهداف:

- کمینه‌سازی زمان اجرا
- کمینه‌سازی انرژی مصرفی اپلیکیشن
- کمینه‌سازی میزان تلاش برای بازآرایی

هر ژن معادل یک عملگر بازآرایی است. در بررسی یک اپلیکیشن، احتمال آن که تنها یک وقوع بازآرایی داشته باشیم، تقریبا نزدیک به صفر است. از طرفی در روال اجرای الگوریتم، با اعمال روش‌هایی که در مراحل انتخاب والدین و نسل بعد و... وجود دارد، تنوع بازآرایی‌ها و ترکیب آن‌ها لحاظ می‌شود که مشروح آن در قسمت‌های ۴,۵,۴ و ۵,۵,۴ مقاله آمده است.

به طور کل در هربار اجرای الگوریتم:

۱- خروجی الگوریتم: مجموعه‌ای از چند کروموزوم یا چند راه‌حل بازآرایی

۲- یک کروموزم: مجموعه چند ژن یا مجموعه چند عمل بازآرایی + تخمین میزان بهبود یا بدترشدن کلی انرژی، زمان اجرا و میزان تلاش برای بازآرایی به‌ازای آن مجموعه (برحسب درصد)

۳- ژن: معادل یک پیشنهاد یا عمل بازآرایی، نمایش داده شده در شکل (۴)

درهربار اجرای الگوریتم، معمولا بیش از یک کروموزوم تولید می‌شود.

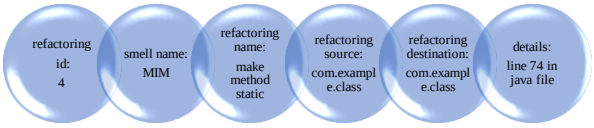
[41] Problem Space

۲،۵،۴. بازنمایی[۴۲]

در روش پیشنهادی، با هرمرتبه اجرای الگوریتم، تعدادی راه‌حل ارائه می‌گردد که هر کدام از مجموعه‌ای پیشنهاد بازآرایی تشکیل شده است. این پیشنهادها زیرمجموعه‌ای از تمام پیشنهادهای بازآرایی تشخیص‌داده‌شده توسط ابزارهای مورد استفاده است. هرراه‌حل، متناظر با یک کروموزوم است که از تعدادی ژن تشکیل شده و هرژن، معادل یک پیشنهاد بازآرایی است.

هرژن طبق شکل ۴ قسمت‌هایی دارد:

- شناسه‌ای عددی که با آن، هر ژن از سایر ژن‌ها مجزا می‌گردد. (refactoring id)
- نام نشانه کد بد یا پادالگو (smell name)
- نوع و نام عملگر متناظر با رفع نشانه کد بد و پادالگو (refactoring name)
- محل وقوع پادالگو که با نام پکیج و کلاس، تعیین می‌گردد و به عنوان مبدا بازآرایی در نظر گرفته می‌شود. (refactoring source)
- محل ایجاد تغییرات برای بازآرایی که با نام کلاس و پکیج تعیین می‌شود که مقصد بازآرایی است. (refacrtoring destination)
- جزئیات بیشتر درباره پادالگو و عملگرهای بازآرایی (درصورت وجود) (details)



شکل (۴): روند نمونه‌ای از بازنمایی ژن در یک کروموزم

۳،۵،۴. تعیین برازش[۴۳]، اهداف تعیین‌شده در مسئله و بازنمایی آن

به‌منظور اندازه‌گیری توابع برازش الگوریتم، ضریب‌های بهبود زمان اجرا، مصرف انرژی و سپس میزان تلاش برای بازآرایی را در نظر گرفتیم. فرمول‌های (۱) و (۲) محاسبه میزان بهبود زمان اجرا و انرژی را برای میانگین ده مرتبه اجرای سناریو در اپلیکیشن‌ها را قبل و بعد از بازآرایی نشان می‌دهد و سپس درصد تغییرات سنجیده شد و میزان تغییرهای هرکدام به طور میانه، گزارش شد. براساس فرمول (۱)، برای هر هدف (O) از میان اهداف بهبود مصرف انرژی و زمان اجرا، کد بازآرایی‌شده (refactored) و کد اولیه (original) با سناریوی متناظرشان اجرا شده و میانگین مصرف انرژی و زمان ((..)avg(CI برای ده مرتبه اجرا، اندازه‌گیری و تغییرات آن‌ها محاسبه شد. بر اساس فرمول (۲) میانه مقادیر محاسبه‌شده براساس فرمول (۱) بررسی گشت. برای محاسبه میزان تلاش برای بازآرایی طبق رابطه (۳) نسبت تعداد بازآرایی‌های لازم به تعداد الگوهای حذف‌شده، را درنظرگرفتیم. فرمول (۴) به تعریف تابع برازش می‌پردازد که طبق آن، میزان تغییرات به‌ازای هر پادالگوی مورد بررسی، $CI(a)$ در تعداد وقوع پادالگوها، $\#detected(a)$ ضرب شده و مجموع آن برای همه انواع پادالگوهای موجود محاسبه شد. از آن جایی که کمینه‌سازی اهداف مدنظر بود، براساس فرمول (۵) محدودیت مسئله ما در این است که مجموع هرکدام از توابع برازش مرتبط با اهداف بهبود زمان اجرا و مصرف انرژی نباید به ۱- برسد چون ۱- به معنای صفرشدن میزان انرژی مصرفی و زمان اجرا است که منطقا چنین امکانی در واقعیت وجود نخواهد داشت.

$$CI(app) = \frac{avg(CI(refactored)) - avg(CI(original))}{avg(CI(original))} :$$

$$\forall O \in Energy, Execution\ time\ Objectives \qquad (۱)$$

$$CI(a) = \text{median}(\text{CI(app)}),$$

[42] Representation

[43] Fitness

$$\forall O \in Energy, Execution\ time\ Objectives \quad (۲)$$

$$CI(a) = \frac{\#refactoring\ operations}{\#fixed\ antipatterns}:$$
$$refactoring\ effrot\ objective \quad (۳)$$

$$Fitness(O) = \sum_{a \in \text{Antippaterns}} CI(a) * \#detected(a),$$
$$\forall O \in Energy, Execution\ time\ and\ refactoring\ effort\ Objectives \quad (۴)$$

$$Fitness(O) > -1.0 :$$
$$\forall O \in Energy, Execution\ time\ Objectives \quad (۵)$$

۴،۵،۴. انتخاب والدین[۴۴]

در فرآیند انتخاب والدین، روش مسابقه[۴۵] با اندازه‌ی ۲ استفاده شد. در این روش ابتدا به تعداد اندازه‌ی مسابقه یعنی (۲) فرد از میان اعضای جمعیت انتخاب می‌شود و از میان آن ۲ فرد، بهترین عضو، انتخاب و به مجموعه والدین اضافه می‌گردد. بعد از آن، همه افرادی که در مسابقه شرکت کردند به جمعیت بازمی‌گردند و این روند برای تعیین و انتخاب هر والد، تکرار می‌گردد. علت این که از روش استفاده کردیم، آن است که در این روش، احتمال انتخاب اعضای جمعیت با برازش کم که به حفظ تنوع جمعیت کمک می‌کند، وجود دارد و احتمال افتادن در نقاط بهینه‌ی محلی کمتر است. پس از آن، عملگر هم‌برش با یک نقطه بازترکیب را استفاده کردیم و پس از آن، عملگر جهش استاندارد با احتمال ۰،۵ را اجرا کردیم.

۵،۵،۴. انتخاب نسل بعد[۴۶]

به منظور انتخاب نسل بعدی، روش برش[۴۷] به کار گرفته شده‌است. در این روش، بعد از این که جمعیت، مرتب شد مادامی که همه اعضای جمعیت مشخص نشده‌اند، اعضا به ترتیب و برحسب مقدار برازش بهتر و مناسب‌تر، انتخاب می‌شود. علت استفاده از این روش این است که در آن، شانسی برای حضور اعضای با برازش بدتر و نامناسب‌تر در نسل‌های بعدی قائل نیست. از نقاط مثبت این روش انتخاب، این است که اعضای بهینه همواره در جمعیت، باقی می‌مانند و روشی نخبه‌گرا[۴۸] است. پس این احتمال که در یک نسل میانی پاسخی بهتر پیدا شده باشد اما از دست برود، وجود نخواهدداشت.

۶،۵،۴. شرط خاتمه الگوریتم

شرط خاتمه الگوریتم بر اساس تعداد نسل، انتخاب می‌شود. در فصل ۵، تاثیر جمعیت بر میزان توابع برازش را مقایسه و بررسی کرده‌ایم.

۵. نتایج

۱،۵. تغییرات زمان اجرا، انرژی مصرفی و معیارهای کیفی کد

در این بخش، پاسخ این سوال را که بازآرایی پادالگوهای مورد بررسی در این پژوهش، چه تاثیری بر مصرف انرژی، زمان اجرای اپلیکیشن و معیارهای کیفی کد دارد، در قالب جداولی بررسی می‌کنیم. نتایج آزمایش‌ها برای ۱۰ بار تکرار هرسناریوی مربوط به هر یک از نشانه‌های بد کد و پادالگوها مربوط محاسبه شده است.

آنچه در جدول‌های ۷ و۸ محل توجه است، نوع تغییرات مقادیر انرژی مصرفی و زمان اجرا پس از اعمال بازآرایی است. برای نمونه بعد از بازآرایی و حذف پادالگوی LIC در یک اپلیکیشن، کاهش مصرف انرژی را شاهد بودیم و در دو اپلیکیشن دیگر، افزایش مصرف انرژی دیده شد. همچنین بازآرایی و حذف این پادالگو در بررسی زمان ا جرا نیز باعث افزایش زمان اجرا شد و در نتیجه رفتاری مشابه را نشان داد. چنین عدم ثباتی به هنگام بازآرایی و حذف پادالگوی MIM نیز مشاهده شده است. در مابقی پادالگوهای مورد بررسی در این

[44] Parent Selection
[45] Tournament
[46] Survive Selection
[47] Truncation
[48] Elitist

پژوهش، در حیطه اپلیکیشن‌های واقعی چنین عدم ثباتی وجود نداشت. به دلیل وجود نمونه‌هایی از عدم ثبات تغییرهای مصرف انرژی و زمان اجرا برای بازآرایی دو پادالگوی یادشده و مشخص‌نبودن علت این عدم ثبات، که البته امکان دارد مربوط به چالش عدم انجام trace سناریوی اجرایی به صورت کامل، توسط ابزار PETrA باشد، از دیگر علت‌هایی بود که باعث شد تا یک اپلیکیشن اندرویدی (androSmell) ا توسعه دهیم تا چالش trace نکردن کامل با ابزار PETrA را نداشته‌باشد و مقادیر قابل اعتمادتر و باثبات‌تری را در اختیار بگذارد. برای محاسبه دقیق‌تر مقادیر زمان اجراشده در اجرای سناریوها، علاوه‌بر ابزار PETrA، از راهکاری دیگر یعنی فراخوانی و استفاده از کتابخانه Debug در میانه کد اپلیکیشن توسعه داده‌شده را هم به کار بردیم.

جدول (۷): میانگین درصد تغییر مقادیر انرژی و زمان اجرا در اپلیکیشن androSmell

نام نشانه بد کد یا نام پادالگو	(%) تغییر انرژی مصرفی	(%) تغییر زمان اجرا بر اساس ابزار PETrA	(%) تغییر زمان اجرا بر اساس Debug
MIM	-23.992	-19.4348	-66.5364

جدول (۸): میانگین درصد تغییر انرژی مصرفی بعد از نجام بازآرایی به تفکیک هراپلیکیشن اندرویدی مورد بررسی

نام اپلیکیشن	LIC	MIM	SL	IG	IS	IDS(HMU)	ArrayAsList	TO ARRAY	Array Loops
hacker_keyboard	-5.43	X	X	X	X	9.4319	0.0088	X	X
anycut	9.35	X	X	X	X	X	X	X	X
hot_death	X	X	31.3817	X	X	2.8588	X	X	-10.5578
password_maker	X	-3.9599	X	X	X	X	X	X	X

نام نشانه بد کد یا نام پادالگو	(%) تغییر انرژی مصرفی	(%) تغییر زمان اجرا بر اساس ابزار PETrA	(%) تغییر زمان اجرا بر اساس Debug
IS	13.7754	15.8264	-0.3564
SL	-14.5184	-15.3391	-2.6695
IDS	-9.4113	-13.3417	8.8597
LIC	-7.7174	-6.7646	1.1986
Array Loops	2.7968	2.4007	173.0838
ArrayAsList	-18.0254	-16.1633	-85.5198
To Array	13.1008	13.4876	32.3153

X	X	X	X	4.2748	-4.2894	10.1678	0.8864	3.7682	matrix_hamilton

در جدول‌های ۷ و ۸ به‌تفکیک هراپلیکشن واقعی مورد بررسی، میزان تغییر انرژی مصرفی و زمان اجرا پس از انجام بازآرایی، ارزیابی شده و به نمایش درآمده است. برای نمونه طبق جدول ۸ در اپلیکیشن matrix_hamilton بعد از این که بازآرایی برای حذف پادالگوی LIC انجام شد، به طور میانگین و بعد از ۱۰ بار تکرار سناریوی اجرایی متناظر با آن، مصرف انرژی حدود ۱۳,۳ درصد بهبود یافت.

در جدول ۷ که مرتبط با تغییرهای زمان اجرا و مصرف انرژی اپلیکیشن توسعه‌یافته androSmell است، با حذف نشانه کد بد MIM انرژی و زمان اجرا کاهش یافت، با حذف نشانه کد بد IS و بازآرایی آن، مشابه نتایج مربوط به سایر اپلیکیشن‌های واقعی مورد مطالعه، افزایش انرژی و برعکس نتایج مرتبط با اپلیکیشن‌های واقعی دیگر، کاهش زمان اجرا را شاهد بودیم. طبق جدول‌های ۷ و ۸ حذف نشانه کد بد SL و بازآرایی آن در اپلیکیشن‌های واقعی، منجر به افزایش زمان اجرا و مصرف انرژی اما طبق جدول ۷، بازآرایی SL در اپلیکشن androSmell منجر به کاهش زمان اجرا و مصرف انرژی است.

بازآرایی IDS طبق جدول‌های ۷ و۸ افزایش مصرف انرژی را نشان می‌دهد اما براساس جدول‌های ۷ و ۹ در اندازه‌گیری زمان اجرا، تفاوت‌ها و ناهماهنگی‌هایی در مقادیر محاسبه‌شده با ابزار PETrA و کتابخانه Debug دیده می‌شد. با بررسی و مقایسه جدول‌های ۷، ۸ و ۹ بازآرایی LIC تفاوت‌هایی در تغییر میزان انرژی مصرفی و زمان اجرا را داشت و حذف ArrayLoops از طریق بازآرایی، رفتاری هماهنگ را در همه اپلیکشن‌ها نشان داد و باعث افزایش انرژی و زمان اجرا شد. بازآرایی ArrayAsList به منظور حذف آن، در اپلیکیشن‌های واقعی مورد بررسی منجر به افزایش انرژی و زمان اجرا و در اپلیکیشن androSmell به کاهش انرژی و زمان اجرا منجر گشت. از آنجایی که نمونه وقوع و تشخیص نشانه کد بد ToArray در میان اپلیکیشن‌های واقعی بررسی‌شده در این رویکرد، مشاهده نشد، در اپلیکیشن توسعه‌یافته‌ی androSmell نمونه آن را پیاده‌سازی کردیم. محاسبات نشان می‌داد که حذف این پادالگو و بازآرایی آن منجر به افزایش زمان اجرا و انرژی شده‌است.

جدول (۹): میانگین درصد تغییر زمان اجرا پس از انجام بازآرایی براساس هراپلیکشن مورد مطالعه

Array Loops	ArrayAsList	TO ARRAY	IS	IG	LIC	SL	IDS(HMU)	MIM	نام اپلیکیشن
X	X	X	X	X	X	X	X	-6.7770	password_maker
X	0.8233	X	X	X	-5.1745	X	8.3082	X	hacker_keyboard
-9.7456	X	X	X	X	X	37.0753	3.5376	X	hot_death
X	X	X	0.8099	-3.7433	-0.6333	9.1747	X	1.1876	matrix_hamilton
X	X	X	X	X	8.4145	X	X	X	anycut

جدول (۱۰): درصد تغییرهای مربوط به معیارهای کیفی کد پس از بازآرایی

Flexibility	Reusability	Extendibility	Understandability	Effectiveness	نشانه بد	نام اپلیکیشن

	کد	(%delta)	(%delta)	(%delta)	(%delta)	(%delta)
passwordmaker	MIM	0	0	0	0	0
hacker_keyboard	asList	0	0.33	0.5	0.25	0.25
	IDS	-	-	-	-	-
	LIC	-	-	-	-	-
	ArrayCopy (ArrayLoops)	0	0	0	0	0
hotdeath	SL	0	0	0	0	0
	ArrayCopy (ArrayLoops)	0	0	0	0	0
	IDS	0	0	0	0	0
matrix	IS_IGS	0	0	0	0	0
	SL	0	0	0	0	0
	MIM	0	0	0	0	0
anycut	LIC	0	0	0	0	0
androSmell	MIM	0	0.33	0	-0.5	0
	SL	0	0	0	0	0
	IDS	0	0.66	0.5	-0.25	0.25
	LIC	0	0	0	0	0
	IS_IGS	0	0	0	0	0
	ArrayCopy (ArrayLoops)	0	0	0	0	0
	asList	0	0	0	0	0
	toArrayCall	0	0.33	0.5	0.25	0.25

۲،۵. مقایسه تغییر مقادیر انرژی در پژوهش حاضر با پالومبا، ۲۰۱۸ [۱۲]

ارجاع [۱۲] که در آن تغییر انرژی در اثر بازآرایی مطالعه شده، برای اندازه‌گیری انرژی، مشابه راهکار پژوهش حاضر از ابزار PETrA استفاده کرده‌است. سه نشانه کد بد IS، MIM و SL که در پژوهش حاضر نیز مطالعه شده، بازآراییشان به عنوان تاثیرگذارترین نشانه‌های کد بد روی مصرف انرژی شناخته شده‌بود. براساس پالومبا، بازآرایی هر سه نشانه کد بد، به طور میانگین، مصرف انرژی را در میان ۶۰ اپلیکیشن کاهش داده است.

در پژوهش حاضر، بازآرایی MIM به صورت میانگین کاهش زمان اجرا و مصرف انرژی را رقم زد اما بازآرایی IS و SL به طور میانگین، مصرف انرژی را افزایش داد. در اپلیکیشن androSmell نیز اگرچه با بازآرایی SL کاهش مصرف انرژی را شاهد هستیم اما میانگین تغییرهای مصرف انرژی، مربوط به حذف و بازآرایی SL در همه شش اپلیکیشن بررسی‌شده،

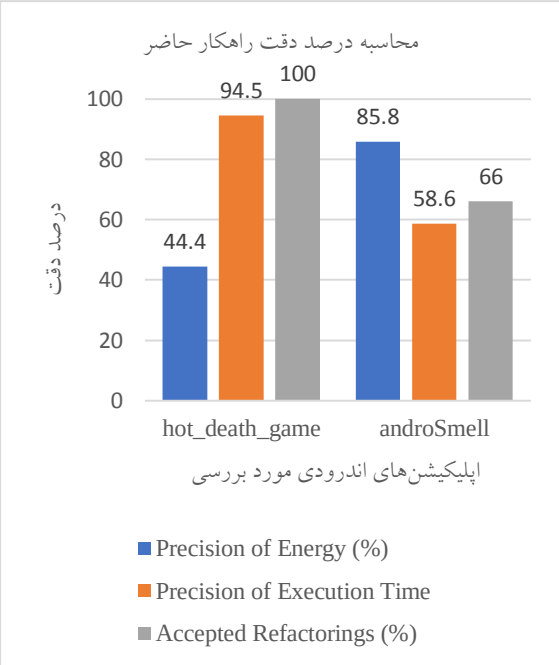
افزایشی است. همچنین حذف IS در اپلیکیشن androSmell افزایش مصرف انرژی همراه را رقم زد و هماهنگ و همسو با نتایج گزارش‌شده در جدول ۸ برای IS می‌باشد. ناهماهنگی در نتایج به‌دست‌آمده در این پژوهش و پالومبا، می‌تواند به علت تفاوت در ساختار اپلیکیشن‌های مورد بررسی در دو پژوهش و یا نحوه فراخوانی پادالگوها و نشانه‌های کد بد در آن اپلیکیشن‌ها باشد. از طرف دیگر در پژوهش حاضر، تفاوتی چشمگیر در میزان تغییر مصرف انرژی بعد از انجام بازآرایی و یا زمان اجرای اپلیکیشن‌های مختلف احتمال دارد به محل فراخوانی آن متد در کد اپلیکیشن‌ها مربوط شود.

۳,۵. محاسبه و بررسی میزان دقت راه‌حل پیشنهادی

برای محاسبه میزان دقت روش حاضر، مشابه مقاله EARMO اقدام شد. در ابتدا با اجرای الگوریتم برای دو اپلیکیشن، خروجی الگوریتم را سنجیدیم. این خروجی شامل دنباله‌هایی از پیشنهادهای بازآرایی بودند که در هر پیشنهاد، دو مقدار متناظر با تغییرهای زمان اجرا و مصرف انرژی تخمین‌زده‌شده، وجود دارد. این پیشنهادهای بازآرایی را ر روی کد هر دو اپلیکیشن، برای برخی از پادالگوها به صورت دستی و برای بعضی دیگر، با ابزار، اعمال کردیم و سپس زمان اجرا و انرژی مصرفی را برای اپلیکیشن‌های بازآرایی‌شده با تکرار اجرای همان سناریوی مربوط، با ابزار PETrA اندازه‌گیری شد و با میزان تخمین‌زده‌شده در الگوریتم، مقایسه گشت. نمودار شکل ۵ و جدول ۱۱ نتایج مربوط را نشان می‌دهد. میانه میزان دقت راه‌حل‌های پیشنهادی از طریق الگوریتم حاضر، برای بهبود زمان اجرا و انرژی به‌ترتیب ۷۶,۵ و ۶۵,۱ درصد است. از سویی دیگر، نسبت تعداد نشانه‌های کد بد و پادالگوهای پیشنهادشده برای حذف از طریق الگوریتم حاضر، به تعداد کل نشانه‌های کد بد و پادالگوهای تشخیص‌داده‌شده با ابزارها، در آن دو اپلیکیشن محاسبه شد. پیشنهادهای بازآرایی در روش پیشنهادی حاضر می‌تواند ۴۲ درصد پادالگوها و نشانه‌های کد بد موجود در اپلیکیشن‌ها را به طور میانه برطرف نماید.

۴,۵. بررسی و مقایسه راه‌حل پیشنهادی در پژوهش حاضر با پژوهش پایه EARMO

اپلیکیشن‌های بررسی‌شده، اهداف الگوریتم‌ها، پادالگوهای مطالعه‌شده و روش محاسبه انرژی در روش پیشنهادی و EARMO متفاوت است. در این بخش به مقایسه قسمتی از خروجی دو پژوهش روی مقادیر مربوط به دقت تغییرات انرژی و زمان اجرا و میزان پادالگوهای حذف‌شده می‌پردازیم. میانه دقت در تخمین تغییرهای انرژی مصرفی برای خروجی الگوریتم‌ها در EARMO، ۶۸٪ اعلام شده، در حالی که در روش پیشنهادی حاضر میزان دقت تخمین تغییر انرژی توسط الگوریتم‌ها، ۶۵,۱٪ و برای زمان اجرا ۷۶,۵٪ به دست آمد. در EARMO خروجی الگوریتم‌های به‌کارگرفته‌شده به‌طور میانه می‌تواند ۸۴٪ پادالگوهای موجود را برطرف کند، اما روش پیشنهادی حاضر به طور میانه ۴۲٪ پادالگوهای موجود را برطرف می‌کند.



شکل(۵): نمودار میزان دقت در تخمین‌ها در روش پیشنهادی برای دو اپلیکیشن

جدول (۱۱): بررسی دقت روش پیشنهادی در مقادیر محاسبه‌شده برای خروجی الگوریتم و در حالت انجام بازآرایی

نام اپلیکیشن	تعداد پیشنهادهای بازآرایی ارائه‌شده در خروجی الگوریتم	تعداد پیشنهادهای بازآرایی قابل انجام در اپلیکیشن	تغییر (انرژی – زمان اجرا) اندازه گرفته‌شده (٪)	تغییر (انرژی – زمان اجرا) تخمین‌زده‌شده در خروجی الگوریتم	دقت خروجی الگوریتم (انرژی مصرفی و – زمان اجرا) (%)
Hot_death_game	2	2	32.53	14.459	44.4
			11.115	11.757	94.5
androSmell	6	4	-30.67	-26.33	85.8
			-29.842	-17.49	58.6

۵.۵. بررسی استفاده از الگوریتم NSGA-II دوهدفه

در این بخش، بررسی می‌کنیم که تغییر فاکتورهایی از الگوریتم مانند نرخ انتخاب والدین، اندازه جمعیت، و حداکثر تعداد نسل چه تاثیری بر عملکرد الگوریتم NSGA-II، می‌تواند داشته‌باشد. خروجی الگوریتم به‌ازای اپلیکیشن‌ها با تعداد پادالگوی متوسط و کم بازآرایی، با تغییر پارامترهای مربوط به الگوریتم که در ادامه به آن اشاره می‌کنیم، تغییر چندانی نکرد، بنابراین این تغییرها را روی یک اپلیکیشن اندرویدی که ۸۴ نشانه کد بد داشت، ارزیابی کردیم و با ۱۰ مرتبه اجرای الگوریتم، میانگین مقادیر توابع برازش در جبهه‌های pareto را در هر اجرا محاسبه

نمودیم. سپس نتایج با آزمون Mann–Whitney–Wilcoxon با ضریب اطمینان ۰,۹۵ ارزیابی شدند.

۱,۵,۵. بررسی تغییر اندازه جمعیت بر عملکرد الگوریتم

به‌منظور بررسی تغییرهای اندازه جمعیت بر الگوریتم، مقادیر جمعیت ۵۰۰ و ۷۰۰ و ۱۰۰۰ را در الگوریتم، تنظیم کردیم. نتایج در جدول ۱۲ آمده‌است که براساس آن، افزایش میزان جمعیت تاثیری در بهبود نتایج نداشته، در نتیجه اندازه جمعیت برای اجرای الگوریتم، ۵۰۰ تنظیم شد.

۲,۵,۵. بررسی تغییر مقدار نرخ انتخاب والدین بر خروجی الگوریتم

به‌منظور بررسی تغییر مقدار نرخ انتخاب والدین بر عملکرد الگوریتم مقادیر ۰,۵ و ۰,۶ و ۰,۷ را تنظیم کردیم. طبق نتایج به‌دست‌آمده براساس جدول ۱۳ افزایش نرخ انتخاب والدین روی بهبود نتایج اثر گذاشته، بنابراین مقدار آن ۰,۷ درنظرگرفته شد.

۳,۵,۵. بررسی تغییر مقدار حداکثر تعداد نسل در الگوریتم

برای بررسی تاثیر حداکثر نسل، مقادیر ۵۰، ۱۰۰ و ۱۵۰ برای اجرای الگوریتم درنظرگرفتیم. طبق جدول ۱۴ و نتایج این آزمایش، افزایش نسل از ۵۰ به ۱۰۰ باعث بهبود قابل توجه مقادیر برازش اما از ۱۰۰ به ۱۵۰ با تاثیر کمی روی نتایج، همراه بود. در نتیجه مقدار مناسب برای حداکثر تعداد نسل در اجرای الگوریتم، ۱۰۰ می‌باشد.

جدول (۱۲): بررسی تاثیر اندازه جمعیت بر مقادیر برازش در خروجی الگوریتم با آزمون Mann–Whitney–Wilcoxon با ضریب اطمینان ۰,۹۵

اندازه جمعیت در آزمون	p-value (energy , execution time)	U (energy , execution time)	میزان تاثیر
[500 , 700]	0.791 , 0.775	49 , 54	بدون تاثیر
[700 , 1000]	0.089 , 0.385	27 , 62	بدون تاثیر
[500 , 1000]	0.212 , 0.236	33 , 60	بدون تاثیر

جدول (۱۳): بررسی تاثیر نرخ انتخاب والدین بر مقادیر برازش در خروجی الگوریتم روش پیشنهادی با آزمون Mann–Whitney–Wilcoxon با ضریب اطمینان ۰,۹۵

مقادیر نرخ انتخاب والدین در آزمون	U (energy , execution time)	p-value (energy , execution time)	میزان تاثیر
[0.5 , 0.6]	10 , 83	0.03 , 0.014	تاثیر قابل توجه
[0.6 , 0.7]	86 , 22	0.007 , 0.038	تاثیر قابل توجه

جدول (۱۴): بررسی تاثیر حداکثر نسل در الگوریتم بر مقادیر برازش روش پیشنهادی با آزمون Mann–Whitney–Wilcoxon با ضریب اطمینان ۰,۹۵

تعداد نسل در آزمون	U (energy , execution time)	p-value (energy , execution time)	تاثیر
[50 , 100]	25 , 78	0.089 , 0.038	تاثیر قابل توجه
[100 , 150]	69 , 36	0.162 , 0.307	تاثیر کم
[50 , 150]	51 , 70	0.97 , 0.14	تاثیر کم

۶,۵. مقایسه و بررسی عملکرد الگوریتم دوهدفه و سه‌هدفه

نخست، الگوریتم با درنظرگرفتن دو هدف بهبود مصرفی انرژی و زمان اجرای الگوریتم، اجرا شد و سپس در نسخه‌ی سه‌هدفه الگوریتم، هدف بهبود میزان تلاش برای بازآرایی را نیز افزودیم و نتایج برحسب معیار (HV)، برای سه اپلیکیشن بررسی شد. در جدول ۱۵ نتایج این مقایسه ذکر شده است.

برای این که بتوانیم مقادیر توابع برازش را برحسب HV، مقایسه کنیم، در یک حالت، مقادیر توابع برازش را برای هرعضو موجود در میان راه‌حل‌های ارائه‌شده در خروجی الگوریتم دوهدفه، به صورت سه‌تایی‌هایی با میزان تلاش برای بازآرایی مساوی صفر تنظیم کردیم و مقدار HV را برای خروجی‌های هرحالت نسبت به یک نقطه‌ی مرجع که ۱۰ درصد زمان اجرا و انرژی مصرفی را بهبود داده و مقدار تلاش برای بازآرایی آن ۱۰۰ است اندازه گرفتیم. در جدول ۱۵ زیر ستون راست به این مقادیر اختصاص دارد، براساس این نتایج، اندازه HV به‌ازای الگوریتم دوهدفه بیشتر شده و معنای آن این است که پاسخ‌های الگوریتم دوهدفه مطلوب‌تر است. در یک حالت دیگر، بدون درنظرگرفتن تلاش برای بازآرایی، صرفا مقادیر توابع برازش مربوط به انرژی مصرفی و زمان اجرا را برای راه‌حل‌های خروجی نسخه‌های دوهدفه و سه‌هدفه الگوریتم ارزیابی کردیم و مقدار HV برایشان محاسبه شد. در این حالت، نقطه مرجع مورد بررسی در HV به صورت دوتایی انرژی و زمان اجرا بود. جدول ۱۵ زیر ستون چپ، به این مقادیر اختصاص دارد. براساس این نتایج، برای اپلیکیشن نخست، HV نسخه سه‌هدفه، مقدار بیشتری از HV نسخه دوهدفه الگوریتم داشته و در اپلیکیشن دوم، هردو مقدار، برابر بوده اما HV اپلیکیشن سوم برای الگوریتم دوهدفه بیش از نسخه سه‌هدفه است.

جدول (۱۵): مقایسه نسخه دوهدفه و سه‌هدفه الگوریتم در روش پیشنهادی برحسب معیار HV

نام اپلیکیشن مورد بررسی	HV به ازای الگوریتم دوهدفه		HV به ازای الگوریتم سه‌هدفه	
	با درنظرگرفتن تلاش برای بازآرایی = ۰	بدون درنظرگرفتن تلاش برای بازآرایی	با درنظرگرفتن مقادیر تلاش برای بازآرایی	بدون درنظرگرفتن مقادیر تلاش برای بازآرایی
androSmell	4.6	0.0460	4.52	0.0460

Hacker_keyboard	36.62	0.366277	36.20	0.366285
Hot_death	40.68	0.406892	39.86	0.406397

۶. جمع‌بندی و کارهای آتی

به‌کارگیری الگوریتم‌های هوش مصنوعی در حوزه بازآرایی مبتنی‌بر انرژی و همین‌طور، بازآرایی مبتنی‌بر زمان اجرای اپلیکیشن‌های اندرویدی به استفاده از روش‌های نرم‌افزاری و یا استفاده از سخت‌افزارهای دقیق و یا انجام راهکارهای آماری نیاز دارد تا میزان تخمین تغییرهای انرژی مصرفی و زمان اجرای اپلیکیشن بر اثر انجام بازآرایی، دقیق‌تر گردد. از سویی دیگر میزان تغییر مقادیر انرژی مصرفی و زمان اجرا، پس از انجام بازآرایی در اپلیکیشن‌های متفاوت، نتایج متفاوتی را ممکن است به همراه داشته‌باشد و از الگوی شناخته‌شده‌ای پیروی نکند.

اقدام‌هایی که می‌توان در ادامه‌ی کار این مقاله و پژوهش حاضر به آن‌ها پرداخت، در دو شاخه‌ی کلی نرم‌افزاری و سخت‌افزاری قابل دسته‌بندی است.

از نگاه حوزه نرم‌افزاری:

- بررسی انواع دیگری از بازآرایی برای حذف پادالگوهای شی‌گرا و بررسی تاثیرش بر مصرف انرژی و زمان اجرا
- بررسی بازآرایی برای حذف سایر پادالگوهای جاوایی و اندرویدی و مطالعه تاثیر آن بر انرژی و زمان اجرا
- بررسی اپلیکیشن‌های اندرویدی با تنوع کاربرد و ساختار
- بررسی میزان کارآیی و دقت سایر ابزارهای تشخیص پادالگو و ارائه یا انجام بازآرایی
- بررسی دقت محاسباتی سایر راهکارهای سخت‌افزاری و یا نرم‌افزاری برای محاسبه مصرف انرژی و زمان اجرا و امکان مقایسه مقادیر با مقادیر حاصل از به‌کارگیری راهکارهایی مثل کتابخانه Debug
- امکان‌سنجی خودکارسازی اعمال پیشنهادهای بازآرایی‌ها

از نگاه حوزه هوش مصنوعی:

- استفاده از دیگر الگوریتم‌های تکاملی و بررسی آن
- درنظرگرفتن اهدافی دیگر برای نمونه در حوزه امنیت با محاسبه و بررسی معیارهای کیفی مرتبط آن

مراجع

[1] I. Manotas, J. Clause, and L. Pollock, “Exploring Evolutionary Search Strategies to Improve Applications’ Energy Efficiency,” Lecture Notes in Computer Science, Jan. 2018, doi: https://doi.org/10.1007/978-3-319-99241-9_15.

[2] T. Mariani and S. R. Vergilio, “A systematic review on search-based refactoring,” Information and Software Technology, vol. 83, pp. 14–34, Mar. 2017, doi: https://doi.org/10.1016/j.infsof.2016.11.009.

[3] R. Morales, R. Saborido, F. Khomh, F. Chicano, and G. Antoniol, “EARMO: An Energy-Aware Refactoring Approach for Mobile Apps,” IEEE Transactions on Software Engineering, vol. 44, no. 12, pp. 1176–1206, Dec. 2018, doi: https://doi.org/10.1109/tse.2017.2757486.

[4] H. Mumtaz, M. Alshayeb, S. Mahmood, and M. Niazi, “An empirical study to improve software security through the application of code refactoring,” Information and Software Technology, vol. 96, pp. 112–125, Apr. 2018, doi: https://doi.org/10.1016/j.infsof.2017.11.010.

[5] M. Harman, S. A. Mansouri, and Y. Zhang, “Search-based software engineering,” ACM Computing Surveys, vol. 45, no. 1, pp. 1–61, Nov. 2012, doi: https://doi.org/10.1145/2379776.2379787.

[6] Fields., Harvie., Fowler., Beck., *Refactoring*. Pearson, India, 2009.

[7] Brown W., *AntiPatterns*. Wiley, NewYork, 1998.

[8] K. Deb, “Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction,” Multi-objective Evolutionary Optimisation for Product Design and Manufacturing, pp. 3–34, 2011, doi: https://doi.org/10.1007/978-0-85729-652-8_1.

[9] M. A. Bokhari, B. R. Bruce, B. Alexander, and M. Wagner, “Deep parameter optimisation on Android smartphones for energy minimisation,” Proceedings of the Genetic and Evolutionary Computation Conference Companion, Jul. 2017, doi: https://doi.org/10.1145/3067695.3082519.

[10] G. Hecht, Naouel Moha, and Romain Rouvoy, “An empirical study of the performance impacts of Android code smells,” HAL (Le Centre pour la Communication Scientifique Directe), May 2016, doi: https://doi.org/10.1145/2897073.2897100.

[11] O. Barack and L. Huang, “Effectiveness of Code Refactoring Techniques for Energy Consumption in a Mobile Environment,” Proceedings of the International Conference on Software Engineering Research and Practice (SERP) 165-171, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2018.

[12] F. Palomba, D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, “On the impact of code smells on the energy consumption of mobile applications,” Information and Software Technology, vol. 105, pp. 43–55, Jan. 2019, doi: https://doi.org/10.1016/j.infsof.2018.08.004.

[13] G. Hecht, Romain Rouvoy, Naouel Moha, and L. Duchien, “Detecting Antipatterns in Android Apps,” HAL (Le Centre pour la Communication Scientifique Directe), May 2015, doi: https://doi.org/10.1109/mobilesoft.2015.38.

[14] https://pmd.github.io/

[15] "F-Droid - Free and Open Source Android App Repository", *F-droid.org*, 2021. [Online]. Available: https://www.f-droid.org/. [Accessed: 07- Aug- 2021].

[16] "Debug | Android Developers", *Android Developers*, 2021. [Online]. Available: https://developer.android.com/reference/android/os/Debug?hl=en. [Accessed: 07- Aug- 2021].

[17] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," in IEEE Transactions on Software Engineering, vol. 28, no. 1, pp. 4-17, Jan. 2002, doi: 10.1109/32.979986.

[18] "monkeyrunner | Android Developers", *Android Developers*, 2021. [Online]. Available: https://developer.android.com/studio/test/monkeyrunner. [Accessed: 07- Aug- 2021].

[19] S. Gholamshahi and SMH. Hasheminejad, “A method for identifying software components based on Non-dominated Sorting Genetic

Algorithm," in Soft Computing Journal, vol.7, no. 2, pp. 47-64, 2019.

[20] M. Hajibaba and S. Parsa, "Software Fault Localization using Cross Entropy and N-gram Models," in Soft Computing Journal, vol. 2, no. 1, pp. 44-59, 2013.

[21] S. Beiranvand and MA. Chahooki, "A Review on Software Cost Estimation Based on Machine Learning," in Soft Computing Journal, vol. 5, no. 1, pp. 36-65, 2016.