مجله محاسبات نرم، جلد ۱۱، شماره ۲ (پاییز و زمستان ۱۴۰۱)، صفحه ۷۲ـ۸۹
https://doi.org/10.22052/scj.2023.246479.1074

مجله محاسبات نرم
SOFT COMPUTING JOURNAL
تارنمای مجله: scj.kashanu.ac.ir

دانشگاه کاشان
University of Kashan

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

مائده زمزمه[۱]، کارشناسی ارشد، سعید صدیقیان کاشی[۱*]، استادیار، امین نیک‌انجام[۱،۲]، استادیار
[۱] دانشکده مهندسی کامپیوتر، دانشگاه صنعتی خواجه نصیرالدین طوسی، تهران، ایران.
[۲] پلی تکنیک مونترال، مونترال، کانادا.

اطلاعات مقاله

تاریخچه مقاله:
دریافت ۲۲ خرداد ماه ۱۴۰۱
پذیرش ۲ مرداد ماه ۱۴۰۲

کلمات کلیدی:
الگوریتم ژنتیک
بازآرایی نرم‌افزار
مهندسی نرم‌افزار مبتنی بر جستجو
برنامه‌های اندرویدی
پادالگو
نشانه کد بد
انرژی مصرفی
زمان اجرا

چکیده

در میان مباحث حوزه مهندسی نرم‌افزار، بهره‌وری انرژی از عوامل موثر در دو مرحله توسعه و نگهداری نرم‌افزار، به مخصوص در دستگاه‌های با انرژی محدود 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 برنامه‌های کاربردی 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 برنامه کاربردی اندرویدی متن‌باز و یک برنامه کاربردی اندرویدی توسعه داده شده، استفاده کردیم. در گام نخست، تغییرهای میزان مصرف انرژی، زمان اجرای برنامه کاربردی 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 برنامه کاربردی 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، با استفاده 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 به طور میانه ۴۲٪ پادالگوها و نشانه‌های کد بد تشخیص‌داده شده در برنامه‌های کاربردی اندرویدی را برطرف سازد.

۱. مقدمه

امروزه استفاده از برنامه‌های کاربردی اندرویدی در دستگاه‌های دارای باتری که منبع انرژی آنها محدودیت نیز دارد، گسترش پیدا کرده است. در دستگاه‌هایی با این مشخصه، بهینه‌سازی مصرف انرژی اهمیت بسزایی دارد [۱]. در مرحله توسعه و نگهداری یک سامانه نرم‌افزاری یا برنامه اندرویدی، بازآرایی[1] نرم‌افزار می‌تواند به بهبود ویژگی‌هایی همانند قابلیت استفاده مجدد،

✧ نوع مقاله: پژوهشی
* نویسنده مسئول
پست(های) الکترونیک: zamzameh@email.kntu.ac.ir (زمزمه)
sedighian@kntu.ac.ir (صدیقیان کاشی)
amin.nikanjam@polymtl.ca (نیک‌انجام)

[1] Refactoring

نحوه ارجاع به مقاله: مائده زمزمه، سعید صدیقیان کاشی، امین نیک‌انجام، «بازآرایی تکاملی چندهدفه آگاه از انرژی برای تصحیح نشانه‌های کد بد در کاربردهای اندرویدی»، مجله محاسبات نرم، جلد ۱۱، شماره ۲، صفحات ۷۲ـ۸۹، پاییز و زمستان ۱۴۰۱.

توسعه‌پذیری[1]، بهبود طراحی و کاهش زمان اجرا از طریق حذف پادالگوها[2] و نشانه‌های کد بد[3] 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 شده است. در مرجع [۳]، 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 شده است. مرجع [۴] به بررسی تاثیر انجام بازآرایی بر امنیت سامانه‌های نرم‌افزاری پرداخته که در برخی موارد انجام بازآرایی، امنیت این سامانه‌ها را کاهش داده است. از آنجایی که هر نوع پادالگو و نشانه کد بد می‌تواند بر میزان مصرف انرژی و زمان اجرای سامانه نرم‌افزاری تاثیر متفاوتی نسبت به دیگری بگذارد، نیاز است که این تاثیر برحسب نوع نشانه کد بد و پادالگو بررسی گردد و با توجه به بهبود یا بدتر شدن مصرف انرژی و زمان اجرا، سنجه‌های[4] مرتبط با الگوریتم تکاملی[5] به کار گرفته شده، تنظیم گردد.

در این مقاله ابتدا تاثیر بازآرایی تعدادی نشانه کد بد بر مصرف انرژی، زمان اجرای برنامه‌های کاربردی و معیارهای کیفی کد اندازه‌گیری شده و سپس الگوریتم تکاملی با در نظر گرفتن چند هدف اجرا شده است. در ادامه، ارزیابی‌ها روی پنج برنامه کاربردی متن‌باز[6] اندرویدی و یک برنامه کاربردی توسعه داده شده[7]، صورت گرفته است. اقدامات انجام شده در روش پیشنهادی را می‌توان به صورت 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 نشده بودند.

۲. تغییرهای زمان اجرا به ازای بازآرایی هشت نشانه کد بد اندازه‌گیری شده است.

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

۴. با بازآرایی هشت نشانه کد بد، در قدم اول، اهداف مساله بهینه‌سازی[8] بهبود مصرف انرژی و زمان اجرای برنامه‌های کاربردی در نظر گرفته شده و سپس بهبود میزان تلاش برای بازآرایی نیز به همراه دو هدف اشاره شده در الگوریتم تکاملی تعریف شده است. با به کاربردن الگوریتم ژنتیک چندهدفه با مرتب‌سازی غیرمغلوب[9] نسخه ۲، به تولید زیرمجموعه‌هایی از عملگرهای بازآرایی مورد بررسی، پرداخته شده است.

۵. میزان تغییرات انرژی و زمان اجرای به دست آمده به ازای حذف نشانه‌های کد بد با کارهای مشابه مقایسه شده و دقت تخمین تغییرات انرژی و زمان اجرا برای زیرمجموعه‌ای از عملگرهای بازآرایی که خروجی الگوریتم است، به ترتیب با میانگین دقت ۶۵٪ و ۷۶٪ به دست آمد. در این راه‌کار، به طور میانگین ۴۲٪ پادالگوهای موجود در برنامه‌های کاربردی حذف شد. همچنین با تغییر متغیرهای مربوط به الگوریتم تکاملی مانند اندازه جمعیت، نرخ انتخاب والدین و حداکثر تعداد نسل به مقایسه نتایج روی معیار HV[10] پرداخته شده است. معیار HV یک شاخص عملکرد[11] است که در الگوریتم‌های چندهدفه استفاده می‌شود و با استفاده از آن، فضای میان مجموعه راه‌حل‌های تولید شده توسط الگوریتم نسبت به یک نقطه مرجع اندازه‌گیری می‌گردد. بیشتر بودن مقدار 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 شده است و در بخش سوم به مرور کارهای مرتبط با

[1] Extendibility
[2] Anti-pattern
[3] Bad code smells
[4] Metric
[5] Evolutionary algorithm
[6] Open Source
[7] Synthetic

[8] Optimization problem
[9] Non-dominated Sorting Genetic Algorithm (NSGA-II)
[10] Hyper Volume (HV)
[11] Performance indicator

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 ذکر شده است.

۲. پیش‌زمینه

۱.۲. مهندسی نرم‌افزار

مهندسی نرم‌افزار، به دنبال حل مساله‌های حیطه نرم‌افزار است، مساله‌هایی که اهداف آنها رقابت‌کننده و احتمالا متناقض نیز هستند. از آنجایی که این مسائل، مجموعه وسیع و گوناگونی از راه‌حل‌ها را شامل می‌شود، می‌بایست پاسخ‌هایی ارائه شود که مصالحه‌ای میان این اهداف برقرار کنند. در ادامه و به عنوان نمونه به پنج سوال قابل بررسی در حوزه مهندسی نرم‌افزار اشاره شده است [۵]:

- بهترین راه‌حل که قابلیت نگهداری[1] را در ساختار معماری یک سامانه افزایش دهد، چیست؟
- مجموعه کمینه‌ای از موارد آزمون[2] 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 باید برای یک سامانه 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 مشخص است، مباحث مورد بررسی در مهندسی نرم‌افزار، حوزه وسیعی از بازآرایی، طراحی، آزمون و مهندسی نیازمندی‌ها را در بر می‌گیرد که به دلیل رقابت‌پذیری و تناقض، از دسته مسائل بهینه‌سازی به شمار می‌آیند.

۲.۲. مهندسی نرم‌افزار مبتنی بر جستجو

با توجه به ماهیت بهینه‌سازی مساله‌های حوزه مهندسی نرم‌افزار، استفاده از الگوریتم‌های مبتنی بر جستجو[3] کاربرد گسترده‌ای دارد؛ چرا که استفاده از این الگوریتم‌ها منجر به کاهش زمان ارائه پاسخ و همچنین بهبود نتایج می‌شود.

۳.۲. بازآرایی

بازآرایی به معنای انجام تغییرهایی است که باعث بهبود کیفیت نرم‌افزار می‌گردد که می‌تواند در هر دو مرحله توسعه و نگهداری نرم‌افزار انجام پذیرد. بازآرایی، ویژگی‌های کیفی[4] مانند قابلیت توسعه‌پذیری[5]، قابلیت نگهداری و کارایی[6] را ارتقا می‌دهد. بازآرایی در سطوح مختلفی مانند کد و معماری می‌تواند انجام شود. منظور از بازآرایی نرم‌افزار، تغییر ساختار سامانه نرم‌افزاری با حفظ رفتار قابل مشاهده و منطق آن سامانه است [۱].

۴.۲. بازآرایی مبتنی بر جستجو

با در نظر گرفتن مسائلی که در حوزه بازآرایی نرم‌افزار، ماهیت بهینه‌سازی دارند و همچنین، ضرورت خودکارسازی یافتن این راه‌حل‌ها، برای کاهش میزان هزینه‌ها، مفهوم بازآرایی مبتنی بر جستجو[7] مطرح می‌گردد. پیدا کردن 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، بر روی آن ویژگی‌ها تاثیر منفی داشته و منجر به کاهش کیفیت سامانه گردد و از طرفی دیگر، بهترین دنباله بازآرایی به طور معمول 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

نشانه‌های کد بد به مفهوم وجود ساختارهایی خاص در کد

[1] Maintainability
[2] Test case

[3] Search-Based Software Engineering (SBSE)
[4] Quality attribute
[5] Extensibility
[6] Performance
[7] Search-based refactoring (SBR)

است که می‌تواند هشدار نقض اصول اساسی در طراحی آن سامانه نرم‌افزاری باشد و ممکن است اثری نامطلوب بر کیفیت طراحی سامانه بگذارد. وجود نشانه کد بد، بیانگر این است که در آن قسمت از کد، بازآرایی ممکن است 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].

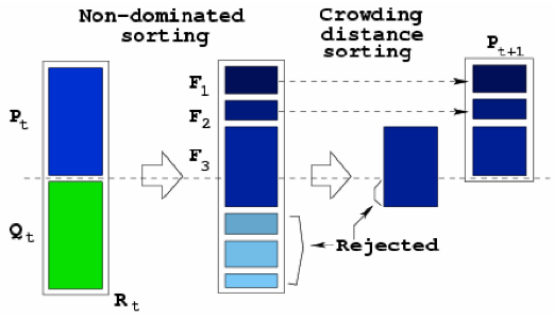
پادالگو، به مفهوم استفاده متداول الگویی در شرایط نادرست است؛ به عبارتی، الگو به مفهوم ارائه راه‌حل مناسب برای یک مساله و به منظور طراحی بهتر آن است؛ حال آنکه پادالگو دقیقا مفهوم متضاد الگو را دارد. به عبارت دیگر، پادالگو به معنای ارائه راه‌حلی برای یک مساله است که منجر به طراحی ضعیف سامانه می‌گردد [۷].

۶.۲. الگوریتم ژنتیک چندهدفه با مرتب‌سازی نامغلوب 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 روش‌های جستجو، ایجاد جمعیت اولیه، انتخاب والد[1] و انجام عملگرهای ادغام[2] و جهش[3] با تعیین مقدار تابع برازش[4] برای حل مساله‌های بهینه‌سازی استفاده می‌شوند [۱].

تمایز الگوریتم ژنتیک با این الگوریتم در ساز و کارهایی مانند مرتب‌سازی غیرمغلوب است. الگوریتم‌های چندهدفه با مرتب‌سازی غیرمغلوب که پیش از ارائه الگوریتم فعلی، استفاده می‌شدند، چالش‌هایی مثل پیچیدگی محاسباتی بالا، حفظ نکردن اعضای نخبه و نیاز به مشخص کردن متغیرهای اشتراک داشتند. الگوریتم فعلی راهکارهایی برای حفظ اعضای نخبه، پیشرفت به سمت جبهه بهینه[5] و حفظ تنوع در انتخاب راه‌حل‌ها در طول اجرای الگوریتم را در نظر گرفته است.

شکل(۱) روند کلی اجرای الگوریتم NSGA-II را نشان می‌دهد، 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 ادغام، فرزندان تولید می‌شوند. جمعیت حاصل بعد از انجام مرتب‌سازی غیرمغلوب[6]، بر اساس رتبه[7] به چند جبهه[8] تقسیم می‌شود. سپس بهترین راه‌حل‌ها از جبهه‌های پیشین انتخاب شده و فاصله ازدحام[9] و اجرای ساز و کارهای باقی ماندن اعضا[10] بر اساس 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 هر یک از اعضای جمعیت بر حسب مقدار تابع برازشی که به ازای هر هدف 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 یاد شده بر عضو مورد مقایسه غلبه پیدا می‌کند. برای هر عضو، مجموعه‌ای از اعضا که آن عضو بر آنها غلبه کرده، محاسبه می‌گردد. پس از آن، بر اساس تعداد اعضای مجموعه مذکور، رتبه هر عضو محاسبه شده و با استفاده از این راه‌کار،

[1] Parent selection
[2] Cross over
[3] Mutation
[4] Fitness function

[5] Pareto-optimal front
[6] Non-dominated sorting
[7] Rank
[8] Front
[9] Crowding distance
[10] Survive selection

جبهه‌بندی همه اعضا انجام می‌گیرد.

۳. کارهای مرتبط

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

در سال ۲۰۱۹ از الگوریتم NSGA-II برای شناسایی مولفه‌های نرم‌افزاری[1] استفاده شده است که هدف مد نظر در آن پژوهش، نگاشت مساله تشخیص مولفه‌های نرم‌افزاری به مساله بهینه‌سازی چندهدفه است [۹]. روش پیشنهادی در این مقاله بر حسب معیارهای انسجام، اتصال و پیچیدگی است و بین این معیارها به منظور تشخیص مولفه‌های مناسب مصالحه انجام می‌شود. نتایج ارزیابی در این مرجع نشان‌دهنده آن است که استفاده از الگوریتم چندهدفه پیشنهادی توانسته بهتر از روش‌های تک‌هدفه عمل کند.

در سال ۲۰۱۳ از راهکارهای یادگیری ماشین برای مکان‌یابی خطاهای پنهان نرم‌افزارها استفاده شده است [۱۰]. در این پژوهش، هدف، ارائه راهکاری برای تعیین خودکار محدوده خطاهای پنهان در متن برنامه‌های نرم‌افزاری می‌باشد. جهت به دست آوردن شباهت مسیرها، مدل‌های N-گرام اجراها محاسبه شده و سپس با استفاده از آنتروپی متقاطع، شباهت بین این مدل‌ها محاسبه گردیده است. 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، مکان‌های مشکوک به خطا به صورت بخش‌هایی از یک زیرمسیر معرفی خواهند شد.

سال ۲۰۱۶ نیز در پژوهشی از رویکردهای یادگیری ماشین برای تخمین هزینه‌های نرم‌افزار در مرحله توسعه نرم‌افزار استفاده شده است [۱۱]. از آنجایی که تخمین هزینه نرم‌افزار امری ضروری در فرآیند توسعه نرم‌افزار است، استفاده از این روش برای تخمین دقیق‌تر هزینه‌ها ضروری می‌نماید.

۱.۳. کارهای مرتبط با حوزه بازآرایی مبتنی بر جستجو

در سال ۲۰۱۶ راه‌کار چندهدفه EARMO[2] ارائه شد 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[3]، SPEA-II[4] و NSGA-II به صورت مستقل استفاده و کارایی هر یک با دیگری مقایسه شده است [۳]. در EARMO، تاثیر بازآرایی هشت پادالگوی شی‌گرای جاوایی و اندرویدی موثر بر مصرف منابع دستگاه‌های اندرویدی، بررسی شده است. برای بررسی میزان تاثیر پیشنهادهای بازآرایی تولید شده به واسطه الگوریتم‌ها، نظر تعدادی از توسعه‌دهندگان برنامه‌های کاربردی اندرویدی درخواست شده بود که بر اساس آنها ۶۸٪ پیشنهادات بازآرایی ارائه شده، مرتبط بوده و ۸۴٪ پادالگوها نیز با پیشنهادهای بازآرایی ارائه شده، تصحیح شده بود.

در سال ۲۰۱۷ بخاری و همکاران به بررسی استفاده از الگوریتم‌های تکاملی برای تعیین مقادیر برخی متغیرها با هدف بهبود مصرف انرژی پرداختند [۱۲]. در روش آنها بر روی بهینه‌سازی مقدار deep parameter با کمینه‌سازی مصرف انرژی پژوهش شده بود. برای نمونه در این پژوهش بررسی شد که از پکیج جاوایی java.util.collection، کدام ساختار داده hashset یا arrayList انتخاب گردد تا همان کاربرد را در پیاده‌سازی کد داشته باشد اما میزان مصرف انرژی آن، بهینه‌تر باشد. برای بازآرایی، تغییرات روی فایل پیکربندی اعمال می‌شد و الگوریتم NSGA-II با در نظر گرفتن دو هدف کمینه‌سازی مصرف انرژی و همین‌طور کمینه‌سازی اختلاف از مقادیر محاسبه شده توسط کتابخانه یاد شده به ازای نمونه‌های آزمون از پیش معین‌شده موجود در مخزن oracle، اجرا شد. خروجی الگوریتم مورد استفاده، مجموعه‌ای از تنظیمات و تغییرات آن با رعایت مصالحه

[1] Software component

[2] Energy-Aware Refactoring approach for Mobile apps
[3] Multi Objective Cellular genetic algorithm
[4] Strength Pareto Evolutionary Algorithm

بین دو هدف مذکور بود. مصرف انرژی با راهکارهای سخت‌افزاری و استفاده از برخی راهکارهای داخلی گوشی تلفن همراه و همین‌طور تراشه‌ای به منظور محاسبه مصرف انرژی و دمای دستگاه اندازه‌گیری شد. نتایج نشان می‌داد که استفاده از الگوریتم‌های جستجوی بکار گرفته شده، منجر به بهبود مصرف انرژی شده است.

در سال ۲۰۱۸ مانوتاس و دیگران طی پژوهش [۱]، بررسی کردند که استراتژی‌های جستجو مانند الگوریتم ژنتیک چگونه می‌تواند به بهبود مصرف انرژی در برنامه‌های کاربردی اندرویدی کمک کند. در این کار به منظور پیدا کردن مجموعه‌ای از APIها با انرژی مصرفی بهینه‌تر و کمتر از چارچوبی به اسم SEEDS استفاده شده و دو الگوریتم NSGA-II و gGA[1] برای یافتن این مجموعه بکار گرفته شد. اهداف تعریف شده در این مساله، بهینه‌سازی زمان ارائه راه‌حل 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 شده بود. نتایج نشان می‌داد که در بیشتر موارد، خروجی الگوریتم جستجو منجر به کاهش مصرف انرژی شده است.

۲.۳. کارهای مرتبط با بررسی تاثیر بازآرایی پادالگوها بر مصرف انرژی و 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 برنامه کاربردی اندرویدی متن‌باز بررسی کردند [۱۳]. به منظور بررسی کارایی رویکرد ارائه شده، چهار سنجه به کار گرفته شد که Delayed Frame و Frame Time دو سنجه تعریف شده برای تاخیر نمایش یک فریم در رابط کاربری بوده، از GC calls برای محاسبه تعداد فراخوانی‌های Collection Garbage و از سنجه Memory Usage برای تعیین میزان استفاده از حافظه استفاده شده است. اندازه‌گیری چهار سنجه مذکور با استفاده از خروجی برخی متغیرهای 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 بوده است.

در سال ۲۰۱۸ باراک و همکاران [۱۴]، بررسی کردند که در زمان بازآرایی یک سامانه، بکار بردن هر یک 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 سامانه نرم‌افزاری دارد. برای بررسی دقیق‌تر این تاثیر از مجموعه معیارهایی با عنوان «GPS-UP» به تفکیک هر عملگر بازآرایی مورد مطالعه در این رویکرد، استفاده کردند. معیارهای بکار گرفته شده برای ارزیابی و بررسی عملگرهای بازآرایی متناظر با پادالگوها و نشانه‌های کد بد، به صورت نسبت مقدارهای قبل و بعد از انجام بازآرایی، برای معیار مورد بررسی، تعریف شده بود.

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

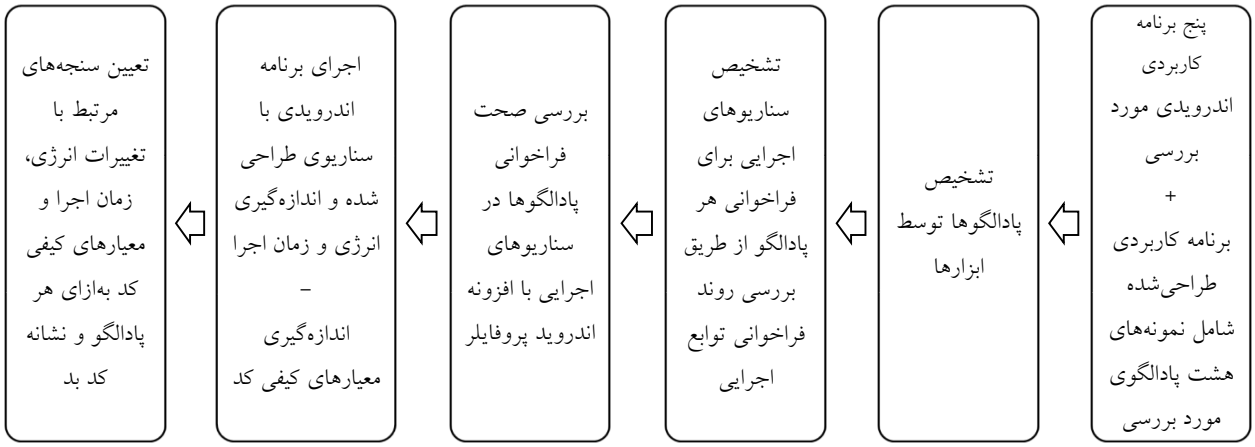
بعد از اعمال هر عملگر بازآرایی و سنجش معیارهای GPS-UP، هر یک از روش‌های بازآرایی بر اساس مقدار سه معیار محاسبه شده، در فضایی سه بعدی نگاشت شدند. به منظور اندازه‌گیری انرژی، زمان و توان مصرفی برای هر عملگر بازآرایی، نمونه پیاده‌سازی شده هر کدام از عملگرهای بازآرایی را در قالب فراخوانی قسمتی از یک برنامه کاربردی اندرویدی بررسی شده، اجرا کردند و محاسبات مربوط بر اساس آن، انجام شد. طبق نتایج به دست آمده برای این عملگرهای بازآرایی، آنها در فضایی سه بعدی نگاشت شدند و بر اساس نتایج، برخی عملگرهای بازآرایی در یک یا دو معیار از میان معیارهای بهبود مصرف انرژی، توان مصرفی و یا زمان اجرا، تعارض داشتند، به این معنا که مقدار یک معیار بهبود یافته ولی معیار دیگر بدتر شده بود.

[1] generational Genetic Algorithm (gGa)

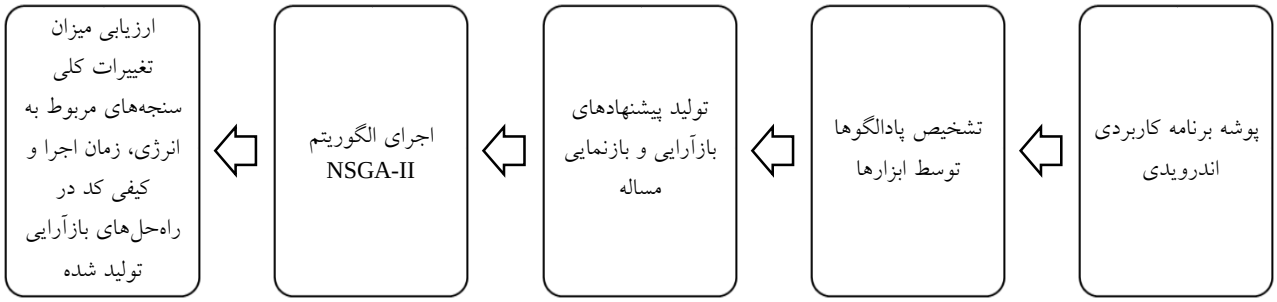
در سال ۲۰۱۹ پالومبا و همکاران [۱۵]، به بررسی ارتباط حذف ۹ نشانه کد بد اندرویدی و تاثیر آنها بر تغییر مقادیر مصرف انرژی پس از انجام بازآرایی پرداختند. این بررسی روی تعدادی از برنامه‌های کاربردی اندرویدی موجود انجام شد. در گام اول، با محاسبه مصرف انرژی برنامه کاربردی، پیش از بازآرایی و سپس با تشخیص و برطرف کردن نشانه‌های کد بد شناخته شده، مصرف انرژی را این بار پس از انجام بازآرایی و با تکرار همان سناریوی اجرایی در برنامه کاربردی، محاسبه کردند. به منظور اندازه‌گیری مصرف انرژی از رویکرد نرم‌افزاری و ابزاری به نام PETrA استفاده شد. نتیجه این پژوهش نشان داد، از میان نه نشانه کد بد بررسی شده، حذف چهار نشانه کد بد یعنی LT، IS، MIM و SL به میزان قابل ملاحظه‌ای مصرف انرژی را بهبود داده است.

۴. روش پیشنهادی

روش پیشنهادی از دو قسمت اصلی تشکیل می‌شود. در بخش نخست و مطابق با شکل (۲)، هشت نشانه کد بد و پادالگوی اندرویدی و جاوایی (معرفی شده در جدول (۱)) بررسی شدند و پرسش اصلی این بود که با اجرای سناریوهایی با قابلیت تکرار، بازآرایی هر یک از نشانه‌های کد بد و پادالگوهای مورد مطالعه چه تاثیری بر مصرف انرژی، زمان اجرا و معیارهای کیفی کد در آن برنامه کاربردی دارد. بر این اساس، انرژی، زمان اجرا و معیارهای کیفی کد با بکار بردن ابزارهایی اندازه‌گیری شد. بر اساس مقادیر تغییرهای مرتبط با انرژی، زمان اجرا و معیارهای کیفی کد، مقدار سنجه‌های متناظر با هر کدام از این سه هدف، در الگوریتم تکاملی تنظیم شد. در بخش دوم و با توجه به شکل (۳)، ابتدا یک برنامه کاربردی اندرویدی توسعه یافته با زبان جاوا، دریافت شده و با استفاده از دو ابزار Paprika [۱۶] و PMD [۱۷] وجود پادالگوها و نشانه‌های کد بد احتمالی در آنها مشخص و بر اساس آن با اجرای الگوریتم NSGA-II همه پیشنهادهای بازآرایی ممکن تولید شد.



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



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

بر اساس مقادیر سنجه‌های مربوط به تغییر مقادیر مصرف انرژی و زمان اجرای هر نشانه کد بد و مجموعه پیشنهادهای بازآرایی، فضای مساله تعریف شد. سپس الگوریتم NSGA-II یک بار با دو هدف و سپس با در نظر گرفتن کمینه‌سازی سه هدف برای میزان مصرف انرژی و زمان اجرا و میزان تلاش برای بازآرایی، اجرا گردید.

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

۱.۴. پادالگوها و بازآرایی

پادالگوها و بازآرایی‌های متناظر بررسی شده در روش پیشنهادی در جدول (۱) عنوان شده است. در EARMO [۳] نیز مانند روش پیشنهادی، تاثیر بازآرایی بر میزان مصرف انرژی برای دو پادالگوی IDS یا HashMapUsage و IGS بررسی شده است؛ اما در EARMO به منظور تخمین میزان انرژی از روش سخت‌افزاری استفاده شده و میزان مصرف انرژی به ازای اجرای یک متد تخمین زده نمی‌شود و از طرفی در EARMO بر تاثیر بازآرایی بر زمان اجرای برنامه کاربردی تمرکز نشده؛ به این دلیل این دو پادالگو در روش پیشنهادی ما نیز مجدد بررسی شدند. بررسی تاثیر بازآرایی بر میزان مصرف انرژی پادالگوهای IDS، MIM، SL و IS ذکر شده در پژوهش پالومبا با روش پیشنهادی حاضر، مشترک و نحوه محاسبه انرژی نیز همانند روش پیشنهادی حاضر بوده، با این حال در پالومبا تاثیر بازآرایی بر زمان اجرا بررسی نشده است.

۲.۴. برنامه‌های کاربردی مورد بررسی

در روش مورد مطالعه در این مقاله، ۱۱ برنامه کاربردی اندرویدی متن‌باز که پوشه کد و سایر مستندهای مربوط به آن در مخزن github موجود بود و یا فایل apk. آن برنامه در مخزن Fdroid [۱۸] در دسترس است، بررسی گردید و از میان آنها پنج برنامه کاربردی، برگزیده و فیلتر شده و در مرحله تحلیل پادالگوها مطالعه شد. شاخص‌های مهم در فیلترکردن و انتخاب این پنج برنامه شامل موارد زیر است:

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

علاوه بر این پنج برنامه کاربردی، یک برنامه کاربردی نیز شامل همه موارد وقوع نشانه‌های کد بد و بازآرایی‌های آنها توسعه داده شد تا دقت تحلیل تغییرات انرژی و زمان اجرا بیشتر شود.

۳.۴. 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 [۱۵] برای تخمین میزان زمان اجرا و مصرف انرژی هر متد جاوایی فراخوانی شده در سناریوی اجرایی، استفاده شد. برای اجرای این ابزار، یک فایل سناریوی اجرایی برنامه کاربردی دریافت، اجرا و انرژی و زمان اجرای متدهای فراخوانی شده در آن سناریو تخمین زده می‌شود. برای ارزیابی و مقایسه دقت محاسبه زمان اجرا توسط ابزار PETrA،

از کتابخانه Debug [۱۹] نیز استفاده شد. همچنین از ابزار RefGen [۲۰] برای اندازه‌گیری معیارهای کیفی کد مانند قابلیت استفاده مجدد[1]، میزان تاثیر[2]، قابل فهم‌بودن کد[3]، توسعه‌پذیری[4] انعطاف‌پذیری[5] استفاده گردید. در بخش ۵، به نتایج این بررسی پرداخته شده است.

۴.۴. خودکارسازی سناریوهای اجرایی

در روش پیشنهادی، از ابزار monkeyrunner [۲۱] به منظور تکرارپذیری سناریوهای اجرایی استفاده شده است. این ابزار به عنوان ورودی، یک فایل به زبان پایتون را دریافت می‌کند که می‌تواند شامل دستورهایی مانند نصب و یا لغو نصب برنامه کاربردی، فشار دادن دکمه‌ای با مختصات معین، گرفتن عکس از صفحه نمایش دستگاه و ایجاد وقفه‌های چندثانیه‌ای و از این دست دستورات باشد. ابزار PETrA یک فایل متنی را که شامل مختصات دکمه‌های مورد نظر و وقفه‌های میان هر عمل است را دریافت می‌کند و با کمک دو اسکریپت به زبان پایتون، شامل چند تابع برای تنظیم متغیرهای مربوط، ابزار را فراخوانی می‌کند. جدول‌های (۲) تا (۶) شامل پادالگوهای تشخیص داده شده در هر برنامه کاربردی و سناریوی اجرایی متناظر آن است.

جدول (۲): سناریوهای اجرایی در برنامه HACKER KEYBOARD

نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
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 برنامه HOT DEATH

نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
SL ArrayLoops IDS	هرسه پادالگو، سناریوی اجرایی مشترکی دارند. بازکردن برنامه و انتخاب اولین حرکت در بازی / CardDeck :ArrayLoops GameTable constructor :IDS GameTable constructor :SL

جدول (۴): سناریوهای اجرا شده مربوط به برنامه‌های PASSWORD MAKER و ANYCUT

نام برنامه	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 اجرا شده در برنامه MATRIX HAMILTON

نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
LIC	فشردن دکمه مرتبط با محاسبات پایه‌ای ماتریس و سپس تعیین ابعاد و نوع آرایه/ Adapter result
MIM	نخست در بخش solve basic matrix یک 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

[1] Reusability
[2] Effectiveness
[3] Understandability
[4] Extendibility
[5] Flexibility

جدول (۶): سناریوهای اجرایی در برنامه ANDROSMELL

نشانه کد بد و پادالگو	سناریوی اجرایی مرتبط / متد / کلاس شامل نشانه کد بد و پادالگو
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 بکار گرفته شده است. ابتدا این الگوریتم با دو هدف بهبود مصرف انرژی و زمان اجرایی برنامه کاربردی، اجرا و سپس هدف بهبود میزان تلاش برای بازآرایی نیز به دو هدف مذکور، افزوده و الگوریتم مجدد اجرا شده است.

۱.۵.۴. توصیف فضای مساله[1]

مجموعه داده‌های مرتبط با یک برنامه کاربردی اندرویدی به عنوان ورودی دریافت و با اجرای ابزارهای تشخیص پادالگو، پادالگوها تشخیص داده شده و معیارهای مورد نیاز، اندازه‌گیری می‌شوند. در ادامه زیرمجموعه‌هایی از تمام پیشنهادهای بازآرایی شناسایی شده و معتبر، به ازای داده‌های برنامه کاربردی ورودی می‌توانست راه‌حل و خروجی الگوریتم باشد. به عبارت دقیق‌تر، هر پاسخ در خروجی الگوریتم اجرایی، مجموعه‌ای از پیشنهادهای بازآرایی با اهداف زیر است.

- کمینه‌سازی زمان اجرا
- کمینه‌سازی انرژی مصرفی برنامه کاربردی
- کمینه‌سازی میزان تلاش برای بازآرایی

هر ژن معادل یک عملگر بازآرایی است. در بررسی یک برنامه کاربردی، 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 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 هر بار اجرای الگوریتم داریم:

۱. خروجی الگوریتم: مجموعه‌ای از چند کروموزوم یا چند راه‌حل بازآرایی
۲. یک کروموزم: مجموعه چند ژن یا مجموعه چند عمل بازآرایی به علاوه تخمین میزان بهبود یا بدتر شدن کلی انرژی، زمان اجرا و میزان تلاش برای بازآرایی به ازای آن مجموعه (برحسب درصد)
۳. ژن: معادل یک پیشنهاد یا عمل بازآرایی، نمایش داده شده در شکل (۴)

دقت داشته باشید که در هر بار اجرای الگوریتم، به طور معمول بیش از یک کروموزوم تولید می‌شود.

۲.۵.۴. بازنمایی[2]

در روش پیشنهادی، با هر مرتبه اجرای الگوریتم، تعدادی راه‌حل ارائه می‌گردد که هر کدام از مجموعه‌ای پیشنهاد بازآرایی تشکیل شده است. این پیشنهادها زیرمجموعه‌ای از تمام پیشنهادهای بازآرایی تشخیص داده شده توسط ابزارهای مورد استفاده است. هر راه‌حل، متناظر با یک کروموزوم است که از تعدادی ژن

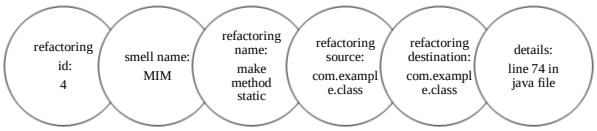
[1] Problem Space

[2] Representation

تشکیل شده و هر ژن، معادل یک پیشنهاد بازآرایی است.

هر ژن طبق شکل (۴)، دارای بخش‌های زیر است.

- 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 (در صورت وجود) (details).



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

۳.۵.۴. برازش، اهداف تعیین شده در مساله و بازنمایی آن

به منظور اندازه‌گیری توابع برازش الگوریتم، ضریب‌های بهبود زمان اجرا، مصرف انرژی و سپس میزان تلاش برای بازآرایی در نظر گرفته شده است. رابطه‌های (۱) و (۲) محاسبه میزان بهبود زمان اجرا و انرژی را برای میانگین ده مرتبه اجرای سناریو در برنامه‌های کاربردی را قبل و بعد از بازآرایی نشان می‌دهد و سپس درصد تغییرات سنجیده شد و میزان تغییرهای هر کدام به طور میانه، گزارش شد.

$$CI(app) = \frac{avg(CI(ref)) - avg(CI(org))}{avg(CI(org))} \quad (۱)$$

$$CI(a) = median(CI(app)) \quad (۲)$$

بر اساس رابطه (۱)، برای هر هدف از میان اهداف بهبود مصرف انرژی و زمان اجرا، کد بازآرایی شده (ref) و کد اولیه (org) با سناریوی متناظر آنها اجرا شده و میانگین مصرف انرژی و زمان ($avg(CI(.))$) برای ده مرتبه اجرا، اندازه‌گیری و تغییرات آنها محاسبه می‌شود. بر اساس رابطه (۲)، میانه مقادیر محاسبه شده بر اساس رابطه (۱) بررسی می‌گردد. برای محاسبه میزان تلاش برای بازآرایی طبق رابطه (۳)، نسبت تعداد بازآرایی‌های لازم به تعداد الگوهای حذف شده، در نظر گرفته شده است.

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

رابطه (۴) به تعریف تابع برازش می‌پردازد که طبق آن، میزان تغییرات به ازای هر پادالگوی مورد بررسی، $CI(a)$، در تعداد وقوع پادالگوها، $\#detected(a)$، ضرب شده و مجموع آن برای همه انواع پادالگوهای موجود محاسبه می‌شود.

$$Fitness = \sum_{a \in \text{Antippaterns}} CI(a) \times \#detected(a) \quad (۴)$$

از آن جایی که کمینه‌سازی اهداف مد نظر است، بر اساس رابطه (۵) محدودیت مساله در این است که مجموع هر کدام از توابع برازش مرتبط با اهداف بهبود زمان اجرا و مصرف انرژی نباید به ۱- برسد چون ۱- به معنای صفر شدن میزان انرژی مصرفی و زمان اجرا است که منطقا چنین امکانی در واقعیت وجود نخواهد داشت.

$$Fitness > -1.0 \quad (۵)$$

۴.۵.۴. انتخاب والدین

در فرآیند انتخاب والدین، روش مسابقه[1] با اندازه ۲ استفاده شده است. در این روش ابتدا به تعداد اندازه مسابقه، 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 استفاده از این روش، آن است که در این روش، احتمال انتخاب اعضای جمعیت با برازش کم که به حفظ تنوع جمعیت کمک

[1] Tournament

می‌کند، وجود دارد و احتمال افتادن در نقاط بهینه محلی کمتر است. پس از آن، عملگر ادغام با یک نقطه بازترکیب استفاده شده است و سپس عملگر جهش استاندارد با احتمال ۰/۵ اجرا شده است.

۵.۵.۴. انتخاب نسل بعد

به منظور انتخاب نسل بعدی، روش برش[1] بکار گرفته شده 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[2] است. پس این احتمال که در یک نسل میانی پاسخی بهتر پیدا شده باشد اما از دست برود، وجود نخواهد داشت.

۶.۵.۴. شرط خاتمه الگوریتم

شرط خاتمه الگوریتم بر اساس تعداد نسل، انتخاب می‌شود. در بخش ۵، تاثیر جمعیت بر میزان توابع برازش مقایسه و بررسی شده است.

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 برنامه‌های کاربردی و معیارهای کیفی کد دارد، در قالب جداولی بررسی شده است. نتایج آزمایش‌ها برای ۱۰ بار تکرار هر سناریوی مربوط به هر یک از نشانه‌های بد کد و پادالگوها مربوط محاسبه شده است.

آنچه در جدول‌های (۷) و (۸) 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 برنامه کاربردی، کاهش مصرف انرژی را شاهد بودیم، در حالی که در دو برنامه کاربردی دیگر، افزایش مصرف انرژی دیده می‌شود. همچنین بازآرایی و حذف این پادالگو در بررسی زمان اجرا نیز باعث افزایش زمان اجرا شده است و در نتیجه رفتاری مشابه را نشان می‌دهد. چنین عدم ثباتی به هنگام بازآرایی و حذف پادالگوی MIM نیز مشاهده شده است. در مابقی پادالگوهای مورد بررسی در این پژوهش، در حیطه برنامه‌های کاربردی واقعی چنین عدم ثباتی وجود ندارد. به دلیل وجود نمونه‌هایی از عدم ثبات تغییرهای مصرف انرژی و زمان اجرا برای بازآرایی دو پادالگوی یاد شده و مشخص نبودن 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 برنامه کاربردی اندرویدی (androSmell) توسعه داده شود تا چالش trace نکردن کامل با ابزار PETrA را نداشته باشد و مقادیر قابل اعتمادتر و باثبات‌تری را در اختیار بگذارد. برای محاسبه دقیق‌تر مقادیر زمان اجرا در سناریوها، علاوه بر ابزار PETrA، از راه‌کاری دیگر یعنی فراخوانی و استفاده از کتابخانه Debug در میانه کد برنامه کاربردی توسعه داده شده هم بهره برده شد.

جدول (۷): میانگین درصد تغییر مقادیر انرژی و زمان اجرا در برنامه کاربردی ANDROSMELL

نام نشانه بد کد یا نام پادالگو	تغییر زمان اجرا (٪)		تغییر انرژی مصرفی (٪)
	Debug	PETrA	
MIM	-66.5364	-19.4348	-23.992
IS	-0.3564	15.8264	13.7754
SL	-2.6695	-15.3391	-14.5184
IDS	8.8597	-13.3417	-9.4113
LIC	1.1986	-6.7646	-7.7174
Array Loops	173.0838	2.4007	2.7968
ArrayAsList	-85.5198	-16.1633	-18.0254
To Array	32.3153	13.4876	13.1008

[1] Truncation
[2] Elitist

جدول (۸): میانگین درصد تغییر انرژی مصرفی بعد از انجام بازآرایی به تفکیک هر برنامه کاربردی اندرویدی مورد بررسی

برنامه کاربردی	LIC	MIM	SL	IG	IS	IDS (HMU)	ArrayAsList	ARRAY TO	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
matrix_hamilton	3.7682	0.8864	10.1678	-4.2894	4.2748	X	X	X	X

در جدول‌های (۷) و (۸) به تفکیک هر برنامه کاربردی 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 برنامه 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 برنامه کاربردی توسعه یافته androSmell است، با حذف نشانه کد بد MIM انرژی و زمان اجرا کاهش یافته است. با حذف نشانه کد بد IS و بازآرایی آن، مشابه نتایج مربوط به سایر برنامه‌های کاربردی واقعی مورد مطالعه، افزایش انرژی و برعکس نتایج مرتبط با برنامه‌های کاربردی واقعی دیگر، کاهش زمان اجرا را شاهد بودیم. طبق جدول‌های (۷) و (۸) حذف نشانه کد بد SL و بازآرایی آن در برنامه‌های کاربردی واقعی، منجر به افزایش زمان اجرا و مصرف انرژی شده، اما طبق جدول (۷)، بازآرایی SL در برنامه androSmell منجر به کاهش زمان اجرا و مصرف انرژی شده است. بازآرایی IDS طبق جداول (۷) و (۸) افزایش مصرف انرژی را نشان می‌دهد اما بر اساس جدول‌های (۷) و (۹) در اندازه‌گیری زمان اجرا، تفاوت‌ها و ناهماهنگی‌هایی در مقادیر محاسبه شده با ابزار PETrA و کتابخانه Debug دیده می‌شد. با بررسی و مقایسه جداول (۷) تا (۱۰)، بازآرایی LIC تفاوت‌هایی در تغییر میزان انرژی مصرفی و زمان اجرا را دارد و حذف ArrayLoops از طریق بازآرایی، رفتاری هماهنگ را در همه برنامه‌های کاربردی نشان می‌دهد و باعث افزایش انرژی و زمان اجرا می‌شود. بازآرایی ArrayAsList به منظور حذف آن، در برنامه‌های کاربردی واقعی مورد بررسی منجر به افزایش انرژی و زمان اجرا و در برنامه androSmell به کاهش انرژی و زمان اجرا منجر شده است. از آنجایی که نمونه وقوع و تشخیص نشانه کد بد ToArray در میان برنامه‌های کاربردی واقعی بررسی شده در این رویکرد، مشاهده نشد، در برنامه‌های کاربردی توسعه یافته androSmell نمونه آن را پیاده‌سازی کردیم. محاسبات نشان می‌دهد که حذف این پادالگو و بازآرایی آن منجر به افزایش زمان اجرا و انرژی شده است.

۲.۵. مقایسه تغییر مقادیر انرژی با مرجع [۱۵]

مرجع [۱۵] که در آن تغییر انرژی در اثر بازآرایی مطالعه شده، برای اندازه‌گیری انرژی، مشابه راهکار پژوهش حاضر از ابزار PETrA استفاده کرده است. بازآرای سه نشانه کد بد IS، MIM و SL که در این پژوهش نیز مطالعه شده، به عنوان تاثیرگذارترین نشانه‌های کد بد روی مصرف انرژی شناخته شده است. بر اساس این مرجع، بازآرایی هر سه نشانه کد بد، به طور میانگین، مصرف انرژی را در میان ۶۰ برنامه کاربردی کاهش داده است.

جدول (۹): میانگین درصد تغییر زمان اجرا پس از انجام بازآرایی بر اساس هر برنامه کاربردی مورد مطالعه

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

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

نام برنامه کاربردی	نشانه بد کد	Effectiveness (%delta)	Understandability (%delta)	Extendibility (%delta)	Reusability (%delta)	Flexibility (%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

در پژوهش حاضر، بازآرایی MIM به صورت میانگین کاهش زمان اجرا و مصرف انرژی را رقم زده است اما بازآرایی IS و SL به طور میانگین، مصرف انرژی را افزایش داده است. در برنامه کاربردی 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 برنامه کاربردی بررسی شده، افزایشی است. همچنین حذف IS در برنامه androSmell افزایش مصرف انرژی را رقم زده است و هماهنگ و همسو با نتایج گزارش شده در جدول (۸) برای IS می‌باشد. ناهماهنگی در نتایج به دست آمده در این پژوهش و مرجع [۱۵]، می‌تواند به علت تفاوت در ساختار برنامه‌های کاربردی مورد بررسی در دو پژوهش و یا نحوه فراخوانی پادالگوها و نشانه‌های کد بد در آن برنامه‌ها باشد. از طرف دیگر در پژوهش حاضر، تفاوتی چشمگیری در میزان تغییر مصرف انرژی بعد از انجام بازآرایی و یا زمان اجرای برنامه‌های مختلف احتمال دارد به محل فراخوانی آن متد در کد برنامه‌های کاربردی مربوط باشد.

۳.۵. محاسبه و بررسی میزان دقت راه‌حل پیشنهادی

برای محاسبه میزان دقت روش حاضر، مشابه مقاله EARMO اقدام شده است. در ابتدا با اجرای الگوریتم برای دو برنامه کاربردی، خروجی الگوریتم سنجیده می‌شود. این خروجی شامل دنباله‌هایی از پیشنهادهای بازآرایی هستند که در هر پیشنهاد، دو مقدار متناظر با تغییرهای زمان اجرا و مصرف انرژی تخمین زده شده، وجود دارد. این پیشنهادهای بازآرایی بر روی کد هر دو برنامه کاربردی، برای برخی از پادالگوها به صورت دستی و برای بعضی دیگر، با ابزار، اعمال شده و سپس زمان اجرا و انرژی مصرفی برای برنامه‌های بازآرایی شده با تکرار اجرای همان سناریوی مربوط، با ابزار PETrA اندازه‌گیری شده است و با میزان تخمین زده شده در الگوریتم، مقایسه گردیده است. 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 به ترتیب ۷۶/۵ و ۶۵/۱ درصد است. از سویی دیگر، نسبت تعداد نشانه‌های کد بد و پادالگوهای پیشنهاد شده

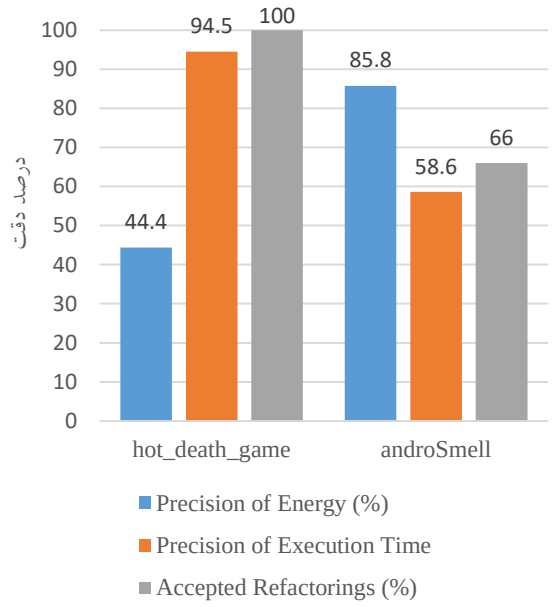
برای حذف از طریق الگوریتم حاضر، به تعداد کل نشانه‌های کد بد و پادالگوهای تشخیص داده شده با ابزارها، در آن دو برنامه کاربردی محاسبه شده است. پیشنهادهای بازآرایی در روش پیشنهادی حاضر می‌تواند ۴۲ درصد پادالگوها و نشانه‌های کد بد موجود در برنامه‌ها را به طور میانه برطرف نماید.

۴.۵. مقایسه راه‌حل پیشنهادی با پژوهش EARMO

برنامه‌های کاربردی بررسی شده، اهداف الگوریتم‌ها، پادالگوهای مطالعه شده و روش محاسبه انرژی در روش پیشنهادی و پژوهش 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 حذف شده پرداخته می‌شود. میانه دقت در تخمین تغییرهای انرژی مصرفی برای خروجی الگوریتم‌ها در EARMO، ۶۸٪ اعلام شده، در حالی که در روش پیشنهادی میزان دقت تخمین تغییر انرژی توسط الگوریتم‌ها، ۶۵/۱٪ و برای زمان اجرا ۷۶/۵٪ به دست آمده است. در پژوهش EARMO خروجی الگوریتم‌های بکار گرفته شده به طور میانه می‌تواند ۸۴٪ پادالگوهای موجود را برطرف کند، اما روش پیشنهادی به طور میانه ۴۲٪ پادالگوهای موجود را برطرف می‌کند.

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 داشته باشد. خروجی الگوریتم به ازای برنامه‌های کاربردی با تعداد پادالگوی متوسط و کم بازآرایی، با تغییر متغیرهای مربوط به الگوریتم که در ادامه به آنها اشاره می‌کنیم، تغییر چندانی نکرده است، بنابراین این تغییرها را روی یک برنامه اندرویدی که ۸۴ نشانه کد بد دارد، ارزیابی می‌کنیم و با ۱۰ مرتبه اجرای الگوریتم، میانگین مقادیر توابع برازش در جبهه‌های Pareto را در هر اجرا محاسبه خواهیم کرد. سپس نتایج با آزمون Mann–Whitney–Wilcoxon با ضریب اطمینان ۰/۹۵ ارزیابی خواهند شد.



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

۱.۵.۵. بررسی تغییر اندازه جمعیت بر عملکرد الگوریتم

به منظور بررسی تغییرهای اندازه جمعیت بر الگوریتم، مقادیر جمعیت ۵۰۰ و ۷۰۰ و ۱۰۰۰ در الگوریتم، تنظیم شده است و نتایج در جدول (۱۲) آمده است که بر اساس آن، افزایش میزان جمعیت تاثیری در بهبود نتایج نداشته، در نتیجه اندازه جمعیت برای اجرای الگوریتم، ۵۰۰ در نظر گرفته شده است.

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

نام برنامه کاربردی	پیشنهادهای بازآرایی ارائه شده توسط الگوریتم	پیشنهادهای بازآرایی قابل انجام در برنامه	تغییر (انرژی - زمان اجرا) اندازه گرفته شده (٪)	تغییر (انرژی - زمان اجرا) تخمینی توسط الگوریتم	دقت خروجی الگوریتم (انرژی - 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

۲.۵.۵. بررسی تغییر مقدار نرخ انتخاب والدین بر خروجی الگوریتم

به منظور بررسی تغییر مقدار نرخ انتخاب والدین بر عملکرد الگوریتم مقادیر ۰/۵، ۰/۶ و ۰/۷ تنظیم شده است. طبق نتایج به دست آمده بر اساس جدول (۱۳)، افزایش نرخ انتخاب والدین روی بهبود نتایج اثر گذاشته، بنابراین مقدار آن در الگوریتم پیشنهادی برابر با ۰/۷ در نظر گرفته شده است.

۳.۵.۵. بررسی تغییر مقدار حداکثر تعداد نسل در الگوریتم

برای بررسی تاثیر حداکثر نسل، مقادیر ۵۰، ۱۰۰ و ۱۵۰ برای اجرای الگوریتم در نظر گرفته شده است. طبق جدول (۱۴) و نتایج این آزمایش، افزایش نسل از ۵۰ به ۱۰۰ باعث بهبود قابل توجه مقادیر برازش شده اما افزایش از ۱۰۰ به ۱۵۰ با تاثیر کمی روی نتایج همراه است. 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	p-value		U		تاثیر
	انرژی	زمان	انرژی	زمان	
[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

نرخ انتخاب والدین	p-value		U		تاثیر
	انرژی	زمان	انرژی	زمان	
[0.5 , 0.6]	0.03	0.014	10	83	قابل توجه
[0.6 , 0.7]	0.007	0.038	86	22	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	p-value		U		تاثیر
	انرژی	زمان	انرژی	زمان	
[50 , 100]	0.089	0.038	25	78	قابل توجه
[100 , 150]	0.162	0.307	69	36	تاثیر کم

۶.۵. مقایسه عملکرد الگوریتم دوهدفه و سه‌هدفه

نخست، الگوریتم با در نظر گرفتن دو هدف بهبود انرژی مصرفی و زمان اجرای الگوریتم، اجرا شده و سپس در نسخه سه‌هدفه الگوریتم، هدف بهبود میزان تلاش برای بازآرایی به آن اضافه شده است. نتایج بر حسب معیار (HV)، برای سه برنامه کاربردی نیز بررسی شده است. در جدول (۱۵) نتایج این مقایسه ذکر شده است. برای این که بتوان مقادیر توابع برازش را برحسب HV، مقایسه کرد، در یک حالت، مقادیر توابع برازش برای هر عضو موجود در میان راه‌حل‌های ارائه شده توسط الگوریتم دوهدفه، به صورت سه‌تایی‌هایی با میزان تلاش برای بازآرایی مساوی صفر تنظیم شده است و مقدار HV برای خروجی‌های هر حالت نسبت به یک نقطه مرجع که ۱۰ درصد زمان اجرا و انرژی مصرفی را بهبود داده و مقدار تلاش برای بازآرایی آن ۱۰۰ است، اندازه گرفته شده است. در جدول (۱۵) ستون راست به این مقادیر اختصاص داده شده است. بر اساس این نتایج، اندازه HV به ازای الگوریتم دوهدفه بیشتر شده و معنای آن این است که پاسخ‌های الگوریتم دوهدفه مطلوب‌تر هستند. در یک حالت دیگر، بدون در نظر گرفتن تلاش برای بازآرایی، صرفا مقادیر توابع برازش مربوط به انرژی مصرفی و زمان اجرا برای راه‌حل‌های خروجی نسخه‌های دوهدفه و سه‌هدفه الگوریتم ارزیابی شده‌اند و مقدار HV برای آنها محاسبه شده است. در این حالت، نقطه مرجع مورد بررسی در HV به صورت دوتایی انرژی و زمان اجرا است. جدول (۱۵) ستون چپ، به این مقادیر اختصاص

دارد. بر اساس این نتایج، برای برنامه کاربردی نخست، HV نسخه سه‌هدفه، مقدار بیشتری از HV نسخه دوهدفه الگوریتم دارد و در برنامه دوم، هر دو مقدار برابر بوده اما HV برنامه سوم برای الگوریتم دوهدفه بیش از نسخه سه‌هدفه است.

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

نام برنامه کاربردی	الگوریتم دوهدفه		الگوریتم سه‌هدفه	
	تلاش بازآرایی برابر صفر	بدون تلاش بازآرایی	با مقادیر تلاش بازآرایی	بدون 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. نتیجه‌گیری و کارهای آینده

بکارگیری الگوریتم‌های هوش مصنوعی در حوزه بازآرایی مبتنی بر انرژی و همین‌طور، بازآرایی مبتنی بر زمان اجرای برنامه‌های کاربردی 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 برنامه بر اثر انجام بازآرایی، دقیق‌تر گردد. از سویی دیگر میزان تغییر مقادیر انرژی مصرفی و زمان اجرا، پس از انجام بازآرایی در برنامه‌های کاربردی متفاوت، نتایج متفاوتی را ممکن است به همراه داشته باشد و از الگوی شناخته شده‌ای پیروی نکند. اقدام‌هایی که می‌توان در ادامه کار این مقاله و پژوهش حاضر به آنها پرداخت، در دو شاخه کلی نرم‌افزاری و هوش مصنوعی قابل دسته‌بندی است. از نگاه نرم‌افزاری می توان به (۱) بررسی انواع دیگری از بازآرایی برای حذف پادالگوهای شی‌گرا و بررسی تاثیر آنها بر مصرف انرژی و زمان اجرا، (۲) بررسی بازآرایی برای حذف سایر پادالگوهای جاوایی و اندرویدی و مطالعه تاثیر آن بر انرژی و زمان اجرا، (۳) بررسی برنامه‌های اندرویدی با تنوع کاربرد و ساختار، (۴) بررسی میزان کارآیی و دقت سایر ابزارهای تشخیص پادالگو و ارائه یا انجام بازآرایی، (۵) بررسی دقت محاسباتی سایر راه‌کارهای سخت‌افزاری و یا نرم‌افزاری برای محاسبه مصرف انرژی و زمان اجرا و (۶) امکان‌سنجی خودکارسازی اعمال پیشنهادهای بازآرایی‌ها، اشاره کرد. از نگاه هوش مصنوعی هم استفاده از دیگر الگوریتم‌های تکاملی و در نظر گرفتن اهدافی دیگر برای نمونه در حوزه امنیت با محاسبه و بررسی معیارهای کیفی مرتبط آن می‌تواند به عنوان کارهای آینده مطرح شود.

تعارض منافع: نویسندگان اعلام می‌کنند که هیچ تعارض منافعی ندارند.

مراجع

[1] I. Manotas, J. Clause, and L.L. Pollock, "Exploring Evolutionary Search Strategies to Improve Applications' Energy Efficiency," in Search-Based Softw. Eng. - 10th Int. Symp. (SSBSE), Montpellier, France, 2018, Proc., pp. 278-292, doi: 10.1007/978-3-319-99241-9_15.

[2] T. Mariani and S.R. Vergilio, "A systematic review on search-based refactoring," Inf. Softw. Technol., vol. 83, pp. 14-34, 2017, doi: 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 Trans. Software Eng., vol. 44, no. 12, pp. 1176-1206, 2018, doi: 10.1109/tse.2017.2757486.

[4] H. Mumtaz, M.R. Alshayeb, S. Mahmood, and M. Niazi, "An empirical study to improve software security through the application of code refactoring," Inf. Softw. Technol., vol. 96, pp. 112-125, 2018, doi: 10.1016/j.infsof.2017.11.010.

[5] M. Harman, S.A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," ACM Comput. Surv., vol. 45, no. 1, pp. 1-61, 2012, doi: 10.1145/2379776.2379787.

[6] J. Fields., S. Harvie., M. Fowler., and K. Beck., Refactoring, Pearson Education, 2009, isbn: 0321604172.

[7] W.J. Brown, R.C. Malveau, H.W.S. McCormick, and T.J. Mowbray, AntiPatterns, Wiley, 1998, isbn: 0471197130.

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

[9] S. Gholamshahi and S.M.H. Hasheminejad, "A method for identifying software components based on Non-dominated Sorting Genetic Algorithm," Soft Comput. J., vol.7, no. 2, pp. 47-64, 2019, dor: 20.1001.1.23223707.1397.7.2.4.5 [In Persian].

[10] M. Hajibaba and S. Parsa, "Software Fault Localization using Cross Entropy and N-gram Models," Soft Comput. J., vol. 2, no. 1, pp. 44-59, 2013, dor: 20.1001.1.23223707.1392.2.1.59.3 [In Persian].

[11] S. Beiranvand and M.A. Zare Chahooki, "A Review on Software Cost Estimation Based on Machine Learning," Soft Comput. J., vol. 5, no. 1, pp. 36-65, 2017 [In Persian].

[12] M.A. Bokhari, B.R. Bruce, B. Alexander, and M. Wagner, "Deep parameter optimisation on Android smartphones for energy minimisation," in Proc. Genet. Evol. Comput. Conf. Companion, 2017, pp. 1501-1508, doi: 10.1145/3067695.3082519.

[13] G. Hecht, N. Moha, and R. Rouvoy, "An empirical study of the performance impacts of Android code smells," in Proc. Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft), Austin, Texas, USA, 2016, pp. 59-69, doi: 10.1145/2897073.2897100.

[14] O. Barack and L. Huang, "Effectiveness of Code Refactoring Techniques for Energy Consumption in a Mobile Environment," in Proc. Int. Conf. Softw. Eng. Res. Practice (SERP), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), Las Vegas, NV, USA, 2018, pp. 165-171.

[15] 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," Inf. Softw. Technol., vol. 105, pp. 43-55, 2019, doi: 10.1016/j.infsof.2018.08.004.

[16] G. Hecht, R. Rouvoy, N. Moha, and L. Duchien, "Detecting Antipatterns in Android Apps," in 2nd ACM Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft), Florence, Italy, 2015, pp. 148-149, doi: 10.1109/mobilesoft.2015.38.

[17] PMD Source Code Analyser (07 Aug. 2021), [Online]. Available: https://pmd.github.io/

[18] F-Droid - Free and Open Source Android App Repository (07 Aug. 2021), [Online]. Available: https://www.f-droid.org/

[19] Debug - Android Developers (07 Aug. 2021), [Online]. Available: https://developer.android.com/reference/android/os/Debug?hl=en

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

[21] Monkeyrunner - Android Developers, Android Developers (07 Aug. 2021), [Online]. Available: https://developer.android.com/studio/test/monkeyrunner