مجله محاسبات نرم
SOFT COMPUTING JOURNAL
تارنمای مجله: scj.kashanu.ac.ir
دانشگاه کاشان
University of Kashan

روشی برای تشخیص مؤلفه‌های نرم‌افزاری مبتنی بر الگوریتم ژنتیک مرتب‌سازی نامغلوب✧

شبنم غلامشاهی[۱]، کارشناسی ارشد، سید محمدحسین هاشمی‌نژاد[۲*]، استادیار
[۱ و ۲] دانشکده فنی و مهندسی، دانشگاه الزهرا(س)، تهران، ایران.

اطلاعات مقاله

تاریخچه مقاله:
دریافت ۱۷ خرداد ماه ۱۳۹۸
پذیرش ۰۳ خرداد ماه ۱۳۹۹

کلمات کلیدی:
تشخیص مؤلفه
طراحی نرم‌افزار
الگوریتم تکاملی چندهدفه
الگوریتم مرتب‌سازی نامغلوب

چکیده

شناسایی مؤلفه‌های نرم‌افزاری مناسب در مرحلهٔ طراحی نرم‌افزار یک کار حیاتی در حوزهٔ مهندسی نرم‌افزار به حساب می‌آید و به‌عنوان یک راه مهم برای افزایش قابلیت نگهداری نرم‌افزار محسوب می‌شود. امروزه روش‌های بسیاری برای شناسایی مؤلفه‌ها مانند تقسیم‌بندی گراف و خوشه‌بندی ارائه شده است، اما اکثر این روش‌ها متکی بر نظر کارشناس و دارای ضعف دقت تشخیص هستند. یکی از دلایل ضعف دقت روش‌های شناسایی مؤلفه، عدم توجه بدین نکته است که معیارهای تشخیص مؤلفه دارای تناقض می‌باشند که لازم است درطی عملیات شناسایی مؤلفه بین آن‌ها مصالحه انجام داد. لذا در این مقاله روشی مبتنی بر الگوریتم ژنتیک مرتب‌سازی نامغلوب– نسخهٔ دو ارائه شده است که هدف آن نگاشت مسئله تشخیص مؤلفه‌های نرم‌افزاری به مسئلهٔ بهینه‌سازی چندهدفه است. در روش پیشنهادی این مقاله از معیارهای انسجام، اتصال و پیچیدگی استفاده شده و بین این معیارها به‌منظور تشخیص مؤلفه‌های مناسب مصالحه انجام شده است. در این مقاله از یک سیستم مورد مطالعه واقعی برای ارزیابی روش پیشنهادی استفاده شده است که نتایج ارزیابی نشان می‌دهد که استفاده از الگوریتم چندهدفهٔ پیشنهادی توانسته بهتر از روش‌های تک‌هدفهٔ گذشته عمل کند.

© ۱۳۹۹ ـ مجلهٔ محاسبات نرم، کلیه حقوق محفوظ است.

۱. مقدمه

بی‌شک طراحی یکی از مراحل سخت و مهم در فرایند توسعهٔ نرم‌افزار است؛ به‌خصوص وقتی سیستم نرم‌افزاری بزرگ و پیچیده باشد رسیدن به یک طراحی خوب، به مشکل بزرگی تبدیل می‌شود. یک طراحی خوب طراحی‌ای است که نیازمندی‌های وظیفه‌مندی و غیر وظیفه‌مندی کارفرما را برآورده کند. طراحی هر سیستم نرم‌افزاری به دو فاز تقسیم می‌شود: طراحی معماری نرم‌افزار و طراحی تفصیلی مؤلفه‌های معماری؛ طراحی معماری نرم‌افزار شامل طراحی مؤلفه‌ها و ارتباطات میان آن‌هاست. در فرایند توسعهٔ مبتنی بر مؤلفه[۱]، تشخیص مؤلفهٔ کاری بسیار حیاتی و ضروری به حساب می‌آید. به‌طور کلی فرایند تشخیص مؤلفه در مرحلهٔ طراحی نرم‌افزار شامل تقسیم‌بندی قابلیت‌های یک سیستم داده شده به واحدهای منطقی به نام مؤلفه است که خود نقطهٔ شروعی برای طراحی معماری نرم‌افزار خواهد بود [۱ و ۲]. این واحدهای منطقی دو ویژگی اصلی دارند [۱]: ۱. همگن بودن در یک مؤلفه به این معنی که ویژگی‌های متعلق به یک مؤلفه باید به‌طور منطقی

✧ نوع مقاله: پژوهشی
* نویسنده مسئول
پست‌های الکترونیک: Sh.Golamshahi@student.alzahra.ac.ir (غلامشاهی)
smh.hasheminejad@alzahra.ac.ir (هاشمی‌نژاد)

1. Omponent Based Development (CBD)

نحوه ارجاع به مقاله: غلامشاهی، شبنم، هاشمی‌نژاد، سید محمدحسین، «روشی برای تشخیص مؤلفه‌های نرم‌افزاری مبتنی بر الگوریتم ژنتیک مرتب‌سازی نامغلوب»، مجله محاسبات نرم، جلد ۷، شماره ۲، ص۴۷ـ۶۴، پاییز و زمستان ۱۳۹۷.

منسجم و مرتبط باشند؛ ۲. ناهمگونی بین مؤلفه‌ها به این معنی که ویژگی‌های یک مؤلفه باید متمایز از سایر مؤلفه‌ها باشد. تاکنون روش‌های خوشه‌بندی، روش مبتنی بر ماتریس CRUD[1]، روش تجزیه و تحلیل مفهوم رسمی (FCA)[2] و روش‌های جست‌وجوبنیان[3] در حوزهٔ تشخیص مؤلفه به کار گرفته شده که در مقالهٔ مروری [۱] فهرست و مزایا و معایب آن‌ها توصیف شده است.

تمامی روش‌های مبتنی بر خوشه‌بندی شامل سه مرحله اصلی هستند [۳]: ۱. تعیین ویژگی‌های داده و جمع‌آوری داده‌ها؛ ۲. محاسبهٔ میزان شباهت مجموعه داده‌ها؛ ۳. اجرای الگوریتم خوشه‌بندی. اکثر روش‌های مبتنی بر خوشه‌بندی از روش‌های کلاسیک خوشه‌بندی استفاده می‌کنند، با وجود این به‌دلیل ساده بودن ساختار این‌گونه الگوریتم‌ها ممکن است مؤلفه‌های ضعیفی را به دست آورند. همچنین یکی دیگر از مشکلات این روش‌ها عدم تشخیص بهترین تعداد مؤلفه است، با این حال تاکنون در خصوص تشخیص مؤلفه، بیشترین روش استفاده‌شده خوشه‌بندی بوده است. در خصوص روش تقسیم‌بندی گراف می‌توان به کار آلبانی[4] و همکاران [۴] اشاره داشت که در آن مدل‌های دامنه شامل اشیا داده، مراحل پردازش و کنشگرها به‌صورت رئوس و لبه‌های یک گراف به نمایش در خواهند آمد و سپس بر اساس نوع رابطه میان عناصر مدل دامنه و تنظیمات طراح، وزنی به لبه‌ها اختصاص داده می‌شود. در نهایت گراف با استفاده از تئوری‌های گراف به مؤلفه‌های منطقی تقسیم خواهد شد. اصلی‌ترین محدودیت این روش تعیین وزن لبه‌های گراف به‌صورت دستی و بر اساس تجربهٔ خبره است.

روش مبتنی بر ماتریس CRUD اولین بار در مقاله [۵] ارائه شده است. در این مقاله ابزاری به نام COMO معرفی شد که یک ماتریس «موردکاربردی/ کلاس» بر مبنای نمودارهای مورد کاربردی و کلاس ساخته شده سپس به مؤلفه‌های تجاری با انسجام[5] بالا تقسیم می‌شده‌اند. همچنین در [۶]، نویسندگان ابزاری مشابه COMO با نام O2BC ارائه کرده‌اند، اما به‌طور کلی این روش با روش‌های خوشه‌بندی تفاوت بسیاری دارد و از رویدادهای تجاری و اشیاء دامنه[6] به‌عنوان ورودی استفاده می‌کند اما محدودیت‌هایی همچون روش‌های خوشه‌بندی دارد.

در روش‌های تجزیه و تحلیل مفهوم رسمی می‌توان به کار ارائه شده در [7] اشاره داشت که در آن یک چهارچوب مبتنی بر تئوری FCA مشابه با روش‌های خوشه‌بندی برای تقسیم‌بندی نمودار کلاس به مؤلفه‌های منطقی ارائه شده است. FCA در واقع روشی برای تحلیل، تجزیه و پردازش دقیق اطلاعات به‌منظور دستیابی به تفسیر معنادار و جامع از اطلاعات است. این روش هم محدودیت‌هایی مشابه روش‌های خوشه‌بندی دارد. در نهایت روش‌های مبتنی بر الگوریتم‌های تکاملی را که به‌طور گسترده‌ای در حوزهٔ مهندسی نرم‌افزار استفاده می‌شود، بررسی می‌کنیم. در این باره می‌توان به دامنهٔ جدیدی از مهندسی نرم‌افزار به نام مهندسی نرم‌افزار مبتنی بر جست‌وجو اشاره داشت که به‌منظور اصلاح مشکلات نرم‌افزاری و نگاشت آن به مسائل بهینه‌سازی ارائه شده است.

در [۸] کارهای انجام‌شده در حوزهٔ طراحی مبتنی بر جست‌وجو مورد بررسی قرار گرفته‌اند. روش‌های مبتنی بر الگوریتم‌های تکاملی یا به‌طور کلی‌تر روش‌های مبتنی بر جست‌وجو، فضای مؤلفه‌ها را بهتر از روش‌های خوشه‌بندی کلاسیک جست‌وجو کرده و برخلاف مشکل معمول روش‌های خوشه‌بندی نیازی به تعیین تعداد مؤلفه‌ها نبوده و این عمل به‌صورت خودکار انجام می‌شود. علاوه بر موارد ذکرشده، این روش‌ها توانایی اعمال سیاست‌ها و محدودیت‌های مورد نظر طراحان سیستم را بیشتر از سایر روش‌های مطرح‌شده دارا هستند. در [۹]، هاشمی و شاه‌محمدی از الگوریتم بهینه‌سازی ذرات تجمعی (PSO)[7] به‌منظور تعیین تعداد مؤلفه‌های بهینه استفاده کرده و سپس از روش خوشه‌بندی فازی برای تشخیص

1. Create Read Update Delete (CRUD)
2. Formal Concept Analysis
3. Search Based
4. Albani
5. Cohesion
6. Domain Objects
7. Particle Swarm Optimization

مؤلفه بهره برده‌اند. در [۱۰]، احمدزاده و همکاران نیز از شبکه‌های عصبی برای تشخیص مؤلفه استفاده کرده است.

این مقاله شامل چهار بخش اصلی است. در بخش دو به پایه و اساس کار و تعاریف اولیه می‌پردازد، بخش سه شامل تعیین متغیرها، ساختار الگوریتم، و نتایج و ارزیابی آن آورده شده و در بخش چهارم نتیجه‌گیری ارائه شده است.

۲. پایه و اساس کار

در این بخش به مفاهیم پایه‌ای و داده‌های اولیهٔ مورد استفاده پرداخته شده است.

۲.۱. داده‌های اولیه

آنچه در این مقاله به‌عنوان داده‌های اولیه استفاده می‌شود موردهای کاربردی، کنشگرها[1] و کلاس‌های موجودیت[2] است و در نهایت هدف اصلی ما تقسیم‌بندی موردهای کاربردی به واحدهای منطقی به نام مؤلفه است. در ابتدا ماتریسی از ارتباط میان موردهای کاربردی، کنشگرها و کلاس‌های موجودیت سیستم ایجاد خواهیم کرد که نمونه‌ای از این ماتریس در شکل (۱) نشان داده شده است.

کلاس موجودیت m	...	کلاس موجودیت ۱	کنشگر I	...	کنشگر ۱	
۰ یا ۱	...	۰ یا ۱	۰ یا ۱	...	۰ یا ۱	مورد کاربرد ۱
۰ یا ۱	...	۰ یا ۱	۰ یا ۱	...	۰ یا ۱	مورد کاربرد ۲
...	...	...	...	...	...	...
۰ یا ۱	...	۰ یا ۱	۰ یا ۱	...	۰ یا ۱	مورد کاربرد n

شکل (۱): ساختار ماتریس ارتباط **میان** موردهای کاربردی، کنشگرها و کلاس‌های موجودیت

در ادامه، بر اساس این ماتریس که توسط طراح سیستم ساخته می‌شود، ماتریس دیگری به نام ماتریس شباهت ساخته می‌شود که ارتباط و نزدیکی موردهای کاربردی به همدیگر را نشان می‌دهد. برای محاسبهٔ این شباهت فرمول‌های متعددی ارائه شده است که در این مقاله از ضریب همبستگی ساده (فرمول ۱) [۱۰] استفاده شده است.

$$S\left(UC_i, UC_j\right) = \frac{n_{11} + n_{00}}{n_{11} + n_{01} + n_{10} + n_{00}} \qquad (۱)$$

در این فرمول، UC نشان‌دهندهٔ مورد کاربردی است و مطابق با مسئلهٔ ما n_{11} نشان‌دهندهٔ تعداد ویژگی‌های موجود در UC_i و UC_j را نشان می‌دهد و منظور از ویژگی‌ها تعداد اشتراکات موردهای کاربردی در کنشگرها و کلاس‌های موجودیت است، n_{10} تعداد ویژگی‌ها موجود در UC_i را نشان می‌دهد که در UC_j نیست و n_{01} تعداد ویژگی‌های موجود در UC_j را نشان می‌دهد که در UC_i موجود نیست و n_{00} تعداد ویژگی‌های است که در هیچ‌کدام وجود ندارد.

۲.۲. کدگذاری

در این مقاله از یکی از کدگذاری‌های معمول در روش‌های خوشه‌بندی برای کدگذاری پاسخ‌های الگوریتم NSGA-II استفاده شده است. به‌طور کلی روش‌های کدگذاری در خوشه‌بندی به سه دستهٔ اصلی تقسیم می‌شوند: ۱. دودویی، ۲. عددی، ۳. حقیقی که در ادامه توضیح مختصری دربارهٔ هرکدام خواهیم داد.

ـ کدگذاری دودویی

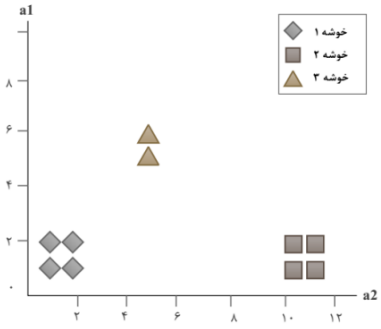
برای استفاده از کدگذاری دودویی، دو شیوهٔ روش مبتنی بر Medoid و روش مبتنی بر ماتریس استفاده شده است. در روش مبتنی بر Medoid، یک رشته N بیتی وجود دارد که هر بیت آن به یک نمونه اشاره می‌کند. در این روش در صورتی که یک بیت مقدار «۱» داشته باشد، بدین معناست که آن نمونه مرکز یک خوشه است و معمولاً برای تعیین اینکه هر نمونهٔ داده به کدام خوشه متعلق است، ابتدا فاصلهٔ آن با تمام مراکز خوشه محاسبه می‌شود، سپس خوشه‌ای که مرکز آن با داده کمترین فاصله را داشته است، انتخاب می‌شود. در روش مبتنی بر ماتریس، به‌ازای هر خوشه یک رشتهٔ N بیتی (به‌اندازه تعداد نمونه‌ها) در نظر گرفته می‌شود و در هر رشته مقدار بیت «۱» نشان‌دهندهٔ این است که نمونهٔ متناظر به این خوشه متعلق است، بنابراین به ماتریسی با اندازهٔ K×N بیت نیاز است. برای روشن‌تر شدن توضیحات داده‌شده مثالی آورده‌ایم؛ شکل (۲) ساختار خوشه‌بندی مثال جدول (۱) را به‌صورت نموداری نشان می‌دهد. همچنین در شکل (۳)، ساختار کدگذاری انجام‌شده

1. Actor
2. Entity Class

برای مثال خوشه‌بندی نمونه‌های جدول (۱) مشاهده می‌شود. در این مثال N (تعداد نمونه‌ها) برابر با ۱۰ و K (تعداد خوشه‌ها) برابر با ۳ است.

جدول (۱): اطلاعات خوشه‌بندی نمونه‌ها (مثال)

خوشه	a_2	a_1	داده (x_i)
خوشهٔ ۱	۱	۱	x_1
خوشهٔ ۱	۲	۱	x_2
خوشهٔ ۱	۱	۲	x_3
خوشهٔ ۱	۲	۲	x_4
خوشهٔ ۲	۱	۱۰	x_5
خوشهٔ ۲	۲	۱۰	x_6
خوشهٔ ۲	۱	۱۱	x_7
خوشهٔ ۲	۲	۱۱	x_8
خوشهٔ ۳	۵	۵	x_9
خوشهٔ ۳	۶	۵	x_{10}



شکل (۲): ساختار خوشه‌بندی جدول ۱

۱	۱	۱	۱	۰	۰	۰	۰	۰	۰
۰	۰	۰	۰	۱	۱	۱	۱	۰	۰
۰	۰	۰	۰	۰	۰	۰	۰	۱	۱

شکل (۳): ساختار کدگذاری شکل ۲

ـ کدگذاری صحیح

برای استفاده از کدگذاری صحیح نیز از دو شیوهٔ مبتنی بر برچسب و مبتنی بر Medoid استفاده شده است. در روش مبتنی بر برچسب، یک آرایهٔ N عنصری وجود دارد که هر خانهٔ آن نشان‌دهندهٔ شمارهٔ خوشه‌ای است که نمونه به آن تعلق دارد. برای مثال، بردار [۱۱۱۱۲۲۲۲۳۳] نمایش کروموزوم خوشه‌بندی انجام‌شده در شکل (۴) هست. روش‌های کدگذاری صحیح به‌صورت ذاتی از مشکل تکراری بودن کروموزوم‌ها رنج می‌برد.

در روش مبتنی بر Medoid ، از آرایه‌ای با K عنصر استفاده می‌شود که در آن هر عنصر مرکز یک خوشه را نشان می‌دهد. برای مثال خوشه‌بندی شکل (۲) به‌صورت [۹ ۵ ۱] نمایش داده می‌شود. شمارهٔ خوشهٔ سایر نمونه‌ها هم با استفاده از کوتاه‌ترین فاصله با مراکز خوشه تعیین می‌شود. گفتنی است که کدگذاری صحیح مبتنی بر Medoid از نظر محاسباتی کاراتر از کدگذاری دودویی مبتنی بر Medoid است. اما در این روش نیز وجود کروموزوم‌های تکراری امکان‌پذیر است.

شکل (۴): نمونهٔ کدگذاری مورد استفادهٔ پاسخ‌ها

ـ کدگذاری حقیقی

در این ساختار آرایه‌ای وجود دارد که هر عنصر آن مراکز ثقل خوشه‌ها (Centroid) را نگهداری می‌کند. به‌عنوان مثال کروموزومی که خوشه‌بندی شکل ۲ را نمایش دهد عبارت است از [(۱,۵ ۱,۵) (۱,۵ ۱۰,۵) (۵,۵ ۵,۰)] ، در این کروموزوم مرکز ثقل هر خوشه توسط یک زوج مرتب حقیقی نمایش داده‌شده است. سپس برای تعیین خوشه‌ی هر نمونه، از روش کمترین فاصله به مراکز ثقل خوشه استفاده می‌شود. روش دیگر ساختار کدگذاری حقیقی مبتنی بر Medoid است. در این ساختار همانند ساختار صحیح مبتنی بر Medoid مراکز خوشه‌های به‌صورت یک زوج مرتب نمایش داده می‌شوند.

در این مقاله از میان ساختارهای اشاره‌شده ساختار کدگذاری دودویی مبتنی بر Medoid استفاده شده است. شکل (۴) نمونه‌ای از ساختار مورد استفاده را نمایش می‌دهد، با توجه به مسئلهٔ مورد بررسی ما در این ساختار، هر کروموزوم از یک آرایه با N درایه تشکیل شده است که N برابر با تعداد موردهای کاربردی است و صفر و یک بودن هر خانه به این معنی است که آن خانه مرکز خوشه یا به‌عبارتی مؤلفه است یا خیر. سایر موردهای کاربردی با توجه به میزان شباهتشان به مراکز خوشه بر مبنای ماتریس شباهت هرکدام در یک خوشه قرار خواهند گرفت.

۳.۲. الگوریتم NSGA-II

در ابتدا باید اشاره‌ای به بهینه‌سازی چندهدفه داشته باشیم؛ به‌عبارتی بهینه‌سازی چندهدفه پیدا کردن بردار $\vec{x} = (x_1, x_2, \dots, x_n)^T$ متشکل از N متغیر تصمیم‌گیری است که بردار تابع هدف $\vec{f}(\vec{x}) = \{f_1(\vec{x}), \dots, f_m(\vec{x})\}$ را بهینه می‌کند. زمانی که بحث از یک الگوریتم تک‌هدفه مطرح است، معیار برتری جواب‌ها نسبت به یکدیگر ساده است؛ زیرا تنها یک تابع هدف مدنظر است. برای مثال در صورتی که مسئلهٔ مورد بحث یک مسئله کمینه‌سازی باشد، جوابی که کمترین مقدار تابع هدف را دارا باشد، بر سایر جواب‌ها برتری دارد؛ اما زمانی برای حل مسئله‌ای از یک الگوریتم چندهدفه استفاده می‌شود که حداقل دو تابع هدف مدنظر است و دیگر به‌آسانی نمی‌توان دربارهٔ برتری بعضی از جواب‌ها نظر قطعی داد؛ زیرا در اکثر موارد، نقاطی یافت می‌شود که هیچ‌کدام بر دیگری برتری کامل ندارد و نمی‌توان با مفهوم غلبه، دوبه‌دو بین آن‌ها مقایسه‌ای انجام داد. یک بردار تصمیم‌گیری $\overrightarrow{x^*}$ بهینه پارتو[1] نامیده می‌شود اگر و تنها اگر هیچ بردار $\overrightarrow{x^*}$ دیگری پیدا نشود که $\overrightarrow{x^*}$ را مغلوب کند؛ به‌عبارت‌دیگر هیچ بردار $\vec{x}$ وجود نداشته باشد که به‌ازای $\forall i \in \{1, 2, \dots, m\}$ مقدار $f_i(\vec{x}) \leq f_i(\overrightarrow{x^*})$ و $\exists i \in \{1, 2, \dots, m\}$ مقدار توابع هدف $f_i(\vec{x}) < f_i(\overrightarrow{x^*})$ باشد. بهینه‌سازی پارتو معمولاً مجموعه‌ای از راه‌حل‌هایی را به نام راه‌حل‌های غیرمغلوب پیشنهاد می‌دهد.

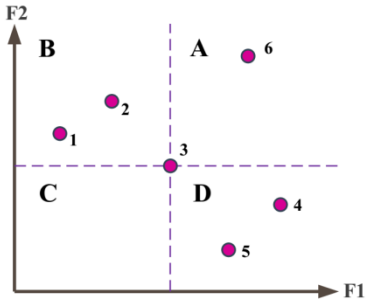
الگوریتم‌های تکاملی چندهدفه که از مرتب‌سازی و به‌اشتراک‌گذاری نامغلوب استفاده می‌کنند، به‌طور کلی از این جهت مورد انتقاد قرار می‌گیرند: ۱. پیچیدگی محاسباتی آن‌ها $O(MN^3)$ است، M تعداد اهداف و N اندازهٔ جمعیت است؛ ۲. از رویکردهای غیر نخبه‌گرایی استفاده می‌کنند؛ ۳. نیازمند تعیین پارامترهای به اشتراک‌گذاری هستند [۱۱]. دب[2] و همکارانش [۱۱] الگوریتم تکاملی چندهدفه مبتنی بر مرتب‌سازی نامغلوب به نام مرتب‌سازی نامغلوب الگوریتم ژنتیک نسخهٔ ۲ (NSGA-II) را ارائه کردند که تمام سه مشکل اشاره‌شده در بالا را حل می‌کرد. در واقع این الگوریتم با اضافه شدن دو عملگر ضروری به الگوریتم ژنتیک تک‌هدفهٔ معمولی به یک الگوریتم چندهدفه تبدیل‌شده است که به‌جای یافتن بهترین جواب تک‌بعدی، دسته‌ای از بهترین جواب‌ها رار می‌دهد که با نام جبههٔ پارتو شناخته می‌شود. ویژگی‌های اصلی این الگوریتم عبارت است از: ۱. فرایند مرتب‌سازی نامغلوب و جبهه‌بندی اعضای جمعیت بر اساس سطحی از عدم غلبهٔ سایر جواب‌ها؛ ۲. استفاده از نخبه‌گرایی و ذخیره‌سازی تمامی راه‌حل‌های نامغلوب که میزان همگرایی را افزایش می‌دهد؛ ۳. استفاده از عملگری که تنوع جواب و پراکندگی آن‌ها را در میان جواب‌های با رتبه برابر حفظ می‌کند؛ ۴. محدودیت‌ها با استفاده از تعریف اصلاح‌شده‌ای از غلبه بدون استفاده از توابع جریمه پیاده‌سازی می‌شود. فلوچارت این الگوریتم در شکل (۵) نمایش داده شده است.

شکل (۵): فلوچارت الگوریتم NSGA-II

در این الگوریتم به هر جواب یک رتبه اختصاص داده می‌شود که بر اساس تعداد مغلوب شدن آن‌ها نسبت به سایر نقاط محاسبه می‌گردد (بدین صورت که نقاطی که در جبههٔ اول قرار دارند و توسط هیچ‌کدام از جواب‌ها مغلوب نشده‌اند،

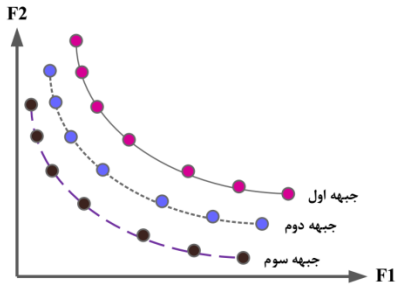
1. Pareto Optimal
2. Deb

دارای رتبهٔ یک و جواب‌هایی که فقط توسط (حداقل یکی از) جواب‌های جبههٔ اول مغلوب می‌شوند، در جبههٔ دوم و رتبهٔ دوم قرار می‌گیرند و...). در پایان الگوریتم، نقاطی که بهترین رتبه یعنی رتبهٔ یک را دارند، به‌عنوان مجموعه جواب یا نقاط جبههٔ پارتو انتخاب می‌شوند. این موضوع با توجه به شکل (۶) که مثالی برای یک مسئلهٔ کمینه‌سازی با دو تابع هدف است شرح داده می‌شود.



شکل (۶): مجموعه جواب برای یک مسئلهٔ کمینه‌سازی دوهدفه

با توجه به شکل (۷)، ابتدا دسته‌ای از اعضای جمعیت که هرگز مغلوب نشده‌اند مشخص‌شده و به آن‌ها رتبهٔ یک اختصاص داده می‌شود. سپس برای بقیهٔ اعضا با نادیده گرفتن اثر اعضای با رتبهٔ یک بر جمعیت، مجدداً مرتب‌سازی نامغلوب انجام می‌شود و اعضایی که در این مرحله هرگز مغلوب نشده‌اند، با رتبهٔ دو مشخص می‌شوند.



شکل (۷): جبهه‌بندی جواب‌ها

برای بقیهٔ اعضا، با نادیده گرفتن اثر اعضای با رتبهٔ یک و دو بر جمعیت، بار دیگر مرتب‌سازی نامغلوب را انجام داده و اعضایی که در این مرحله هرگز مغلوب نشده‌اند، با رتبهٔ سه مشخص می‌شوند و این روند تا جایی ادامه می‌یابد که رتبهٔ همهٔ اعضای جمعیت مشخص شود و در نهایت اعضای جمعیت رتبه‌بندی می‌شوند و بر مبنای روش انتخاب و سیاست در نظر گرفته‌شدهٔ یک مجموعه جواب انتخاب می‌شوند. مراحل اصلی الگوریتم پیشنهادی به‌همراه جزئیات در شکل (۸) نمایش داده شده است.

۴.۲. اهداف الگوریتم

برخی از اهدافی که باید در مسئلهٔ تشخیص مؤلفهٔ نرم‌افزاری مورد بررسی قرار داد، مواردی همچون انسجام، اتصال[1]، پیچیدگی[2] و بسیاری دیگر است که ما در این مقاله به این سه مورد مطرح‌شده خواهیم پرداخت و یکی از نکات مهم تناقض دو هدف انسجام و اتصال است و باید در نظر گرفت که جنبه‌های اتصال و انسجام و به‌طبع آن پیچیدگی مؤلفه‌های نرم‌افزاری از جمله ویژگی‌های کیفی است که به‌شدت بر معماری نرم‌افزار، تعمیر و نگهداری، تکامل و استفادهٔ مجدد از سیستم اثر می‌گذارد [۱۲] و سیستم نرم‌افزاری مطلوب است که انسجام بالا و اتصال پایین داشته باشد.شایان ذکر است که منظور از مؤلفه، در روش پیشنهادی، مجموعه‌ای از موردهای کاربری است.

در این مقاله برای به‌دست‌آوردن میزان انسجام، اتصال و پیچیدگی مؤلفه‌های به‌دست‌آمده، از فرمول‌های ارائه‌شده در مقالهٔ [۱۳] استفاده شده است. قبل از آنکه به فرمول‌ها اشاره شود، لازم است تعاریفی از انسجام، اتصال و پیچیدگی را ارائه گردد. بر این مبنا، تعریفی از چیدامبر[3] و کمرر[4] [۱۴] انتخاب شده است: «اتصال به درجهٔ استقلال بین قسمت‌هایی از یک طراحی اشاره می‌کند، درحالی‌که انسجام به سازگاری داخلی هریک از قسمت‌های طراحی‌شده اشاره دارد.» بنابراین به حداقل رساندن اتصال و حداکثر ساختن میزان انسجام باید در نظر گرفته شود. به‌طور کلی در سیستم‌های نرم‌افزاری هدف ساخت سیستمی متشکل از اجزای مستقلی است که

1. Coupling
2. Complexity
3. Chidamber
4. Kemerer

قابلیت‌های سیستم را در همکاری با هم برآورده می‌سازند [۱۵].

حال تعریفی که می‌توان برای معیار پیچیدگی در نظر گرفت، در ساده‌ترین حالت چیزی جز «مشکل در درک، تغییر و نگهداری نرم‌افزار» [۱۶] نخواهد بود. در واقع پیچیدگی یک مسئلهٔ ذهنی است که بر توانایی توسعه‌دهندگان در درک نرم‌افزار تأثیر می‌گذارد. ازاین‌رو برای غلبه بر پیچیدگی، سیستم را به قطعات کوچک‌تر تقسیم می‌کنند تا درک، تغییر و نگهداری ساده‌تر شود [۱۷]. در ادامه به ارائهٔ فرمول‌هایی برای محاسبهٔ میزان این معیارها پرداخته می‌شود.

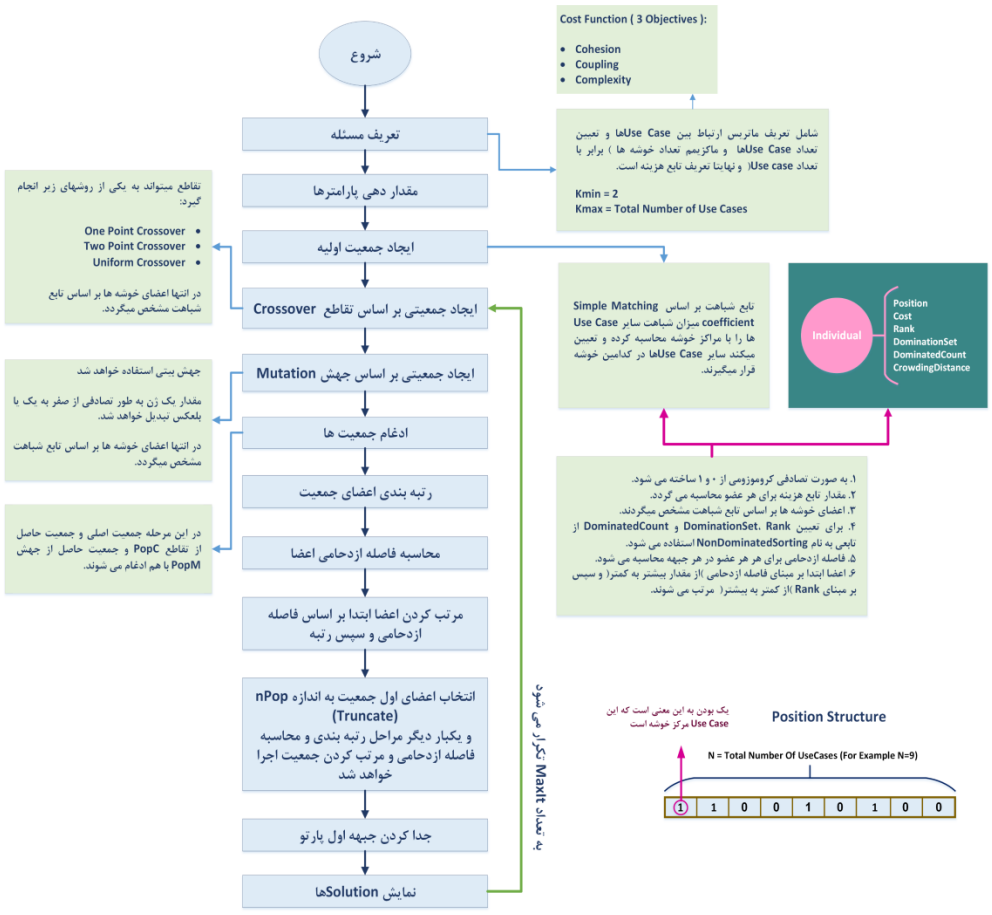
ـ انسجام

در ابتدا باید اشاره کرد که برای محاسبهٔ میزان انسجام یک مؤلفه، مبنا بر شباهت میان موردهای کاربردی یک مؤلفه خواهد بود که این شباهت در ماتریس شباهتی که در بخش ۱.۲ به آن اشاره شد، حساب شده است.

فرمول (۲) [۱۰] برای محاسبهٔ انسجام موجود در یک مؤلفه استفاده می‌شود. m_c تعداد موردهای کاربردی را مشخص می‌کند؛ به‌عبارتی انسجام یک مؤلفه جمع شباهت‌های موجود بین تمامی زوج‌های موردهای کاربردی آن مؤلفه تقسیم بر بیشترین ارتباط بین آن‌هاست.

$$S\left(UC_i, UC_j\right) = \frac{n_{11} + n_{00}}{n_{11} + n_{01} + n_{10} + n_{00}}$$

$$CC\left(cmp_c\right) = \begin{cases} 1 & \text{If } m_c = 1 \\ \dfrac{\sum_{\forall uc_i \in cmp_C} \sum_{\forall uc_j \in cmp_C} S\left(UC_i, UC_j\right)}{\binom{m_c}{2}} & \text{Otherwise} \end{cases} \qquad (۲)$$



شکل (۸): مراحل اجرای روش پیشنهادی

مقدار $CC(cmp_c)$ برای یک مؤلفه در بازهٔ [۰،۱] قرار دارد و اگر یک مؤلفه تنها یک مورد کاربردی داشته باشد، این مقدار برابر ۱ است. هرچه مقدار *CC* برای مؤلفه بیشتر باشد، طبق تعاریف ارائه‌شده نشان‌دهندهٔ بهتر بودن ساختار آن مؤلفه است، اما در نهایت برای محاسبهٔ میزان انسجام کلی نرم‌افزار و مؤلفه‌های تشخیص داده‌شده از فرمول (۳) استفاده می‌کنیم، در این فرمول n نشان‌دهندهٔ تعداد مؤلفه‌های تشخیص داده‌شده است:

$$SoftwareCohesion = \sum_{c=1}^{n} \frac{cc\left(cmp_c\right)}{n} \qquad (۳)$$

ـ اتصال

تاکنون مطالعات بسیاری در جهت تعیین معیار اندازه‌گیری اتصال یک مؤلفه انجام شده است؛ اگرچه بسیاری از آن‌ها در مراحل اولیهٔ طراحی نرم‌افزار قابل قبول نیستند زیرا معمولاً نیازمند متغیرهایی هستند که از کد برنامه استخراج می‌شود. در این مقاله از فرمول (۴) استفاده شده است به این علت که این فرمول با مدل مورد کاربردی سازگار و قابل استفاده است.

این فرمول میزان اتصال را برای یک مؤلفه اندازه‌گیری می‌کند. در این فرمول $cp(cmp_c)$ نشان‌دهندهٔ تعداد مؤلفه‌هایی است که با مؤلفهٔ cmp_c در ارتباط هستند و *UCMP* تعداد کل مؤلفه‌های سیستم است:

$$CCR\left(cmp_c\right) = \frac{\left|cp\left(cmp_c\right)\right|}{\left|UCMP\right| - 1} \qquad (۴)$$

در فرمول *CCR* دو مؤلفه دارای اتصال هستند اگر میان موردهای کاربردی آن‌ها ارتباطی وجود داشته باشد. مقدار *CCR* برای یک مؤلفه در بازهٔ [۰،۱] قرار دارد و مقدار یک نشان‌دهندهٔ این است که این مؤلفه با تمامی مؤلفه‌های دیگر ارتباط دارد و مقدار صفر مستقل بودن مؤلفه از سایر مؤلفه‌ها را نشان می‌دهد. بنابراین هرچه این مقدار کمتر باشد یعنی مؤلفه بهتر تشخیص داده شده است، اما برای به‌دست‌آوردن مقدار اتصال کل سیستم از فرمول (۵) استفاده شده است:

$$SoftwareCoupling = \sqrt{\frac{\sum_{c=1}^{n}\left(CCR\left(cmp_c\right)\right)^2}{\text{No. Of Use Cases}}} \qquad (۵)$$

ـ پیچیدگی

اگرچه معیارهای زیادی برای محاسبهٔ پیچیدگی نرم‌افزاری وجود دارد، اکثر آن‌ها مانند معیارهای پیچیدگی Object Oriented (OO) ازجمله شاخص چیدامبر- کمرر [۱۴] روش متفاوتی برای توصیف ویژگی‌های CBSD مانند پیچیدگی مؤلفه ندارند. با این حال، چندین معیار برای محاسبهٔ پیچیدگی مؤلفه ارائه شده است و بیشتر آن‌ها در مرحلهٔ اولیهٔ طراحی نرم‌افزار به‌دلیل کمبود اطلاعات قابل استفاده نیستند. یکی از معیارهای مشهور و قابل‌قبول در سطح Use Case Model معیاری تحت عنوان UCP (Use Case Point) [۱۸] است.

UCP معیار اندازه‌گیری پیچیدگی نرم‌افزاری است و در دو مرحله انجام می‌شود که در فرمول ۶ [۱۸] نمایش داده شده است.

$$UCP = UUCP + AUCP \qquad (۶)$$

در مرحلهٔ اول UUCP یا Uncorrected UCP بر اساس میزان وزن هر کنشگر و مورد کاربردی محاسبه می‌شود که مراحل محاسباتی در فرمول‌های ۷، ۸ و ۹ [۱۸] نشان داده شده است.

$$UCP = UAW + UUC \qquad (۷)$$

$$UAW = \sum\left(\text{Complexity Weight}\right)\times\left(\text{\# of Actors Associated with Complexity}\right) \qquad (۸)$$

$$UUCW = \sum\left(\text{Complexity Weight}\right)\times\left(\text{\# of Use Cases Associated with Complexity}\right) \qquad (۹)$$

در مرحلهٔ دوم AUCP یا Adjusted UCP با استفاده از پیچیدگی فنی[1] محاسبه می‌شود. اگرچه در این مقاله به‌علت دشوار بودن به‌دست‌آوردن معیار، پیچیدگی فنی AUCP نادیده گرفته شده است.

Unadjusted Actor Weight (UAW): یک کنشگر می‌تواند یک شخص، یک برنامهٔ نرم‌افزاری و یا یک دستگاه سخت‌افزاری باشد. سپس UAW بر اساس سه نوع کنشگر ذکرشده دارای وزن‌های پیچیدگی متناسب می‌گردد که در جدول (۲) نشان داده شده است.

1. Technical Complexity

سطح پیچیدگی موردهای کاربردی به‌طور مستقیم با تعداد کلاس‌های تجزیه‌وتحلیل در ارتباط است. سپس UUCW بر اساس سه نوع از موردهای کاربردی و میزان وزن پیچیدگی آن‌ها محاسبه می‌شود. جدول (۳) انواع موردهای کاربردی و وزن‌هایشان را نشان می‌دهد.

جدول (۲): انواع کنشگرها و وزن پیچیدگی آن‌ها

وزن	توضیح	نوع
۱	رابط برنامه	ساده
۲	رابط کاربری تعاملی	متوسط
۳	رابط گرافیکی	پیچیده

جدول (۳): انواع موردهای کاربردی و وزن پیچیدگی آن‌ها

وزن	توضیح	نوع
۱	کمتر از ۵ کلاس تجزیه‌وتحلیل دارد	ساده
۲	بین ۵ تا ۱۰ کلاس تجزیه‌وتحلیل دارد	متوسط
۳	بیشتر از ۱۰ کلاس تجزیه‌وتحلیل دارد	پیچیده

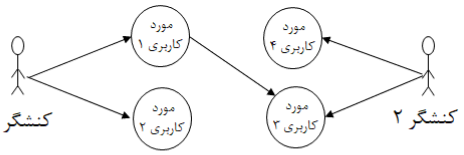
آنچه در نهایت به‌عنوان معیاری برای اندازه‌گیری پیچیدگی مؤلفه در این مقاله استفاده شده است، فرمولی تحت عنوان $ComponentComplexity(\mathrm{cmp_c})$ است که در [۱۳] بر اساس معیار UCP ارائه شده است. معیار مطرح‌شده در فرمول (۱۰) نشان داده شده است.

$$ComponenComplexity\ (cmp_c) = \sum_{\forall UC_i \in cmp_c} \frac{ucp_i}{m_c \times TotalUCP} \qquad (۱۰)$$

در این فرمول، m_c تعداد موردهای کاربردی موجود در مؤلفه، UCP_i و $TotalUCP$ به‌ترتیب برابر با پیچیدگی i امین مورد کاربردی و جمع تمامی UCP_i هاست. مقدار پیچیدگی برای هر مؤلفه در بازهٔ [۰،۱] قرار دارد و هرچه این مقدار برای یک مؤلفه کمتر باشد، به این معنی است که مؤلفه بهتر تشخیص داده شده است. در نهایت برای به‌دست‌آوردن میزان پیچیدگی کلی سیستم از فرمول (۱۱) [۱۹] استفاده خواهد شد:

$$SoftwareComplexity = \sqrt{\frac{\sum_{c=1}^{n}\left(ComponentComplexity\ (cmp_c)\right)^2}{\text{No. Of Use Cases}}} \qquad (۱۱)$$

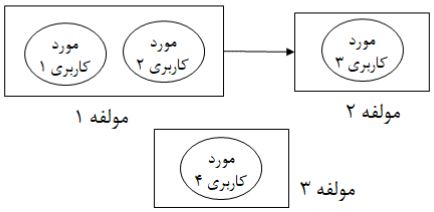
برای نمونه، مثالی در شکل (۹) برای محاسبهٔ معیارهای انسجام، اتصال و پیچیدگی ارائه شده است. در این مثال، ۴ مورد کاربری، ۲ کنشگر و ۳ کلاس موجودیتی وجود دارد که ابتدا روابط و ماتریس ارتباط آن‌ها نمایش داده شده است. سپس یک کروموزوم نمونه برای شناسایی مؤلفه‌ها از روی این موارد کاربری نمایش داده شده است. در این کروموزوم، سه ژن مقدار برابر یک دارند که نشان‌دهندهٔ سه مؤلفه می‌باشد و مورد کاربری ۲ نیز که ژن آن برابر صفر است، چون شباهت بیشتری به مورد کاربری ۱ دارد، در مؤلفه ۱ قرار می‌گیرد.



	کنشگر ۱	کنشگر۲	کلاس ۱	کلاس۲	کلاس۳
مورد کاربری ۱	۱	۰	۱	۰	۰
مورد کاربری ۲	۱	۰	۰	۰	۰
مورد کاربری ۳	۰	۱	۱	۰	۱
مورد کاربری ۴	۰	۱	۰	۱	۱

1	0	1	1

یک نمونه کروموزوم



$$S(UC_1, UC_3) = \frac{1+1}{1+1+2+1}$$
$$S(UC_1, UC_2) = \frac{1+3}{1+3+0+1}$$
$$CC(\mathrm{cmp}_1) = \frac{\frac{4}{5}}{\binom{2}{2}}$$
$$CCR(cmp_1) = \frac{1}{3-1}$$
$$UAW(UC_1) = 3 \times 1$$
$$UUCW(UC_1) = 1 \times 1$$

شکل (۹): مثالی از شیوهٔ محاسبهٔ فرمول‌های انسجام، اتصال و پیچیدگی

در ادامهٔ این مثال، شباهت موارد کاربری ۱ و ۳ و شباهت موارد کاربری ۱ و ۲ نمایش داده شده است. همچنین برای

مؤلفهٔ ۱ که دارای دو مورد کاربری است، فرمول انسجام مؤلفه نمایش داده شده است. همچنین برای مؤلفهٔ ۱، فرمول اتصال ارائه شده است که این مؤلفه فقط با مؤلفهٔ ۳ در ارتباط است و تعداد کل مؤلفه‌های کروموزوم نیز ۳ عدد می‌باشد. در نهایت برای مورد کاربری ۱، پیچیدگی در نظر گرفته شده است؛ که به‌دلیل اینکه فرض شده کنشگر از جنس رابط گرافیکی است، وزن آن ۳ و به‌دلیل اینکه در ماتریس کلاس‌های موجودیتی تعداد کلاس‌ها کمتر از ۵ می‌باشد، پیچیدگی کلاس‌ها ۱ در نظر گرفته شده است. برای مثال دیگر، در شکل (۱۰) شمایی از تقاطع بین دو کروموزوم نمایش داده شده است.

همان طور که در شکل (۴) نمایش داده شده، ساختار کدگذاری کروموزوم‌های روش پیشنهادی مبتنی بر Medoid است که به‌ازای هر مورد کاربری، یک ژن وجود دارد و مقدار آن در صورتی که ۱ باشد، به معنی مرکز خوشه و در غیر این صورت صفر به معنی عدم مرکز خوشه بودن مورد کاربری را نشان می‌دهد. در این ساختار پس از تعیین مراکز خوشه، سایر موردهای کاربری به خوشه‌هایی تعلق می‌گیرند که نزدیک‌ترین فاصله را با مرکز آن خوشه‌ها داشته باشد. همچنین پس از ایجاد راه‌حل نهایی، توابع برازندگی بر اساس فرمول‌های انسجام، اتصال و پیچیدگی ذکرشده محاسبه می‌شود.

شکل (۱۰): مثالی از شیوهٔ تقاطع بین دو کروموزوم

۳. نتایج تجربی

در این بخش به مواردی همچون تعیین متغیرها و ساختار پیاده‌سازی، سیستم مورد مطالعه و نتایج به‌دست‌آمده از پیاده‌سازی روش پیشنهادی روی سیستم مورد مطالعه پرداخته می‌شود.

۱.۳. پیاده‌سازی

پیاده‌سازی انجام‌شده در محیط MATLAB R2016a و روی سیستمی با سیستم عامل Windows 10، پردازندهٔ Core i7 3.4 GHz و ۸ گیگابایت حافظه اجرا شده است.

این الگوریتم دارای ۵ متغیر است که قبل از اجرای الگوریتم تعیین می‌شوند. متغیرهای الگوریتم در جدول (۴) ارائه شده است. توجه کنید که جمعیت اولیه را به‌صورت تصادفی ایجاد شده و برای هر عنصر از کروموزوم مقدار تصادفی یک یا صفر قرار داده می‌شود.

جدول (۴): متغیرهای الگوریتم NSGA-II

متغیر	توضیح	مقدار
N_{pop}	اندازهٔ جمعیت	۲۰۰
i_{max}	تعداد تکرار	۲۳۰
P_{crs}	احتمال تقاطع	۰/۸
P_{mut}	احتمال جهش	۰/۰۱
N_{uc}	تعداد مورد کاربردی	توسط خبره تعیین می‌شود
N_{crs}	تعداد والدین	$2 \times round(P_{crs} \times N_{pop}/2)$
N_{mut}	تعداد جهش‌یافته‌ها	$round(P_{mut} \times N_{pop})$

نویسندگان از عملگر تقاطع تک‌نقطه‌ای و دونقطه‌ای استفاده کرده‌اند. همان طور که در شکل (۸) مشاهده می‌شود، عملگر جهش بعد از عملگر تقاطع اعمال می‌شود. در این مقاله از جهش بیتی استفاده شده است، به این صورت که به طور تصادفی، یک بیت وضعیتش از صفر به یک یا برعکس تغییر خواهد کرد.

بعد از هر تکرار طبق ساختار الگوریتم NSGA-II جبهه‌های مختلف تعیین شده‌اند؛ بهترین جواب‌ها در جبههٔ اول قرار دارند و در نهایت برابر با مقدار جمعیت تعیین‌شده در الگوریتم (در اینجا ۵۰ است) به‌ترتیب از جبههٔ اول تا آخر جایگزین جمعیت قبل خواهند شد و در نهایت در صورتی که تعداد تکرارها به حداکثر اندازهٔ خود رسیده باشند، الگوریتم پایان خواهد یافت. گفتنی است از میان مجموعه جواب‌هایی که در جبههٔ اول پارتو قرار گرفته‌اند، در نهایت یکی بر اساس بیشترین تعداد غلبه بر

سایر جواب‌های موجود در جبهه‌های دیگر به‌عنوان جواب نهایی الگوریتم انتخاب خواهد شد.

۲.۳. مورد مطالعه

سیستم مورد مطالعه در این مقاله برای اجرای الگوریتم و شناسایی مؤلفه‌های آن یک سیستم باشگاه مشتریان بانکی است که شامل ۴۶ مورد کاربرد، ۳ کنشگر و ۱۹ کلاس موجودیتی است (پیوست شمارهٔ ۱ فهرست و روابط آن‌ها را نمایش داده است). تقسیم‌بندی موردهای کاربرد به مؤلفه‌ها توسط خبره در شکل (۱۱) نشان داده شده؛ همان طور که از شکل مشخص است، سامانهٔ باشگاه مشتریان به هفت مؤلفهٔ امتیازات مشتری، الگو تشویق و تنبیه، جایگاه مشتری، مصرف امتیاز، کارت وفاداری، گزارش‌ها و خرید محصولات بیمه تقسیم شده است.

شایان ذکر است که خبرگان این مورد مطالعه، شامل یک تیم معماری و طراحی، ۵ نفره با تجربهٔ کاری مجموع ۴۳ سال بوده و این پروژه در یک شرکت نرم‌افزاری وابسته به یکی از بانک‌های خصوصی کشور تهیه و هم‌اکنون نیز در اختیار مشتریان بانک است.

۳.۳. نتایج ارزیابی

پس از اجرای الگوریتم پیشنهادی روی مورد مطالعهٔ ارائه‌شده در بخش ۲.۳، جدول (۵) میانگین توابع هدف (انسجام، اتصال و پیچیدگی) و تعداد مؤلفه‌ها را برای جبههٔ اول پارتو ارائه می‌دهد.

جدول (۵): میانگین مقادیر توابع هدف و تعداد مؤلفه برای جبهه اول

میانگین تعداد مؤلفه	میانگین انسجام سیستم	میانگین اتصال سیستم	میانگین پیچیدگی سیستم
۶/۴	۰/۸۶	۰/۱۵۱	۰/۰۲۸

نتایج حاصل از اجرای الگوریتم پیشنهادی روی داده‌های حاصل از سیستم مورد مطالعه در شکل (۱۲) نمایش داده شده است. همان طور که در شکل (۱۲) مشاهده می‌شود، در جواب‌های به‌دست‌آمده، در بسیاری از نسل‌ها، تضادی مابین مقادیر معیارهای اتصال، انسجام و پیچیدگی وجود دارد. برای نمونه در نسل ۲۱۳، با وجود افزایش انسجام، افزایش اتصال و در نسل ۱۲۹، با وجود افزایش انسجام، کاهش اتصال نیز مشاهده می‌شود. بنابراین این شکل نشان‌دهندهٔ این است که سه معیار مورد استفاده در این مسئله، با یکدیگر تضاد دارند و بر یکدیگر تأثیر متقابل می‌گذارند؛ همچنین استفاده از یک الگوریتم بهینه‌سازی چندهدفه را توجیه می‌کنند.

از میان جواب‌های به‌دست‌آمده در جبههٔ اول پارتو (جواب‌های به‌نمایش‌درآمده در شکل ۱۲) بر اساس میزان تعداد غلبه بر سایر جواب‌های موجود در سایر جبهه‌ها، تنها جوابی که بیشترین غلبه را داشته است به‌عنوان جواب نهایی اجرای الگوریتم انتخاب و مورد ارزیابی قرار خواهد گرفت.

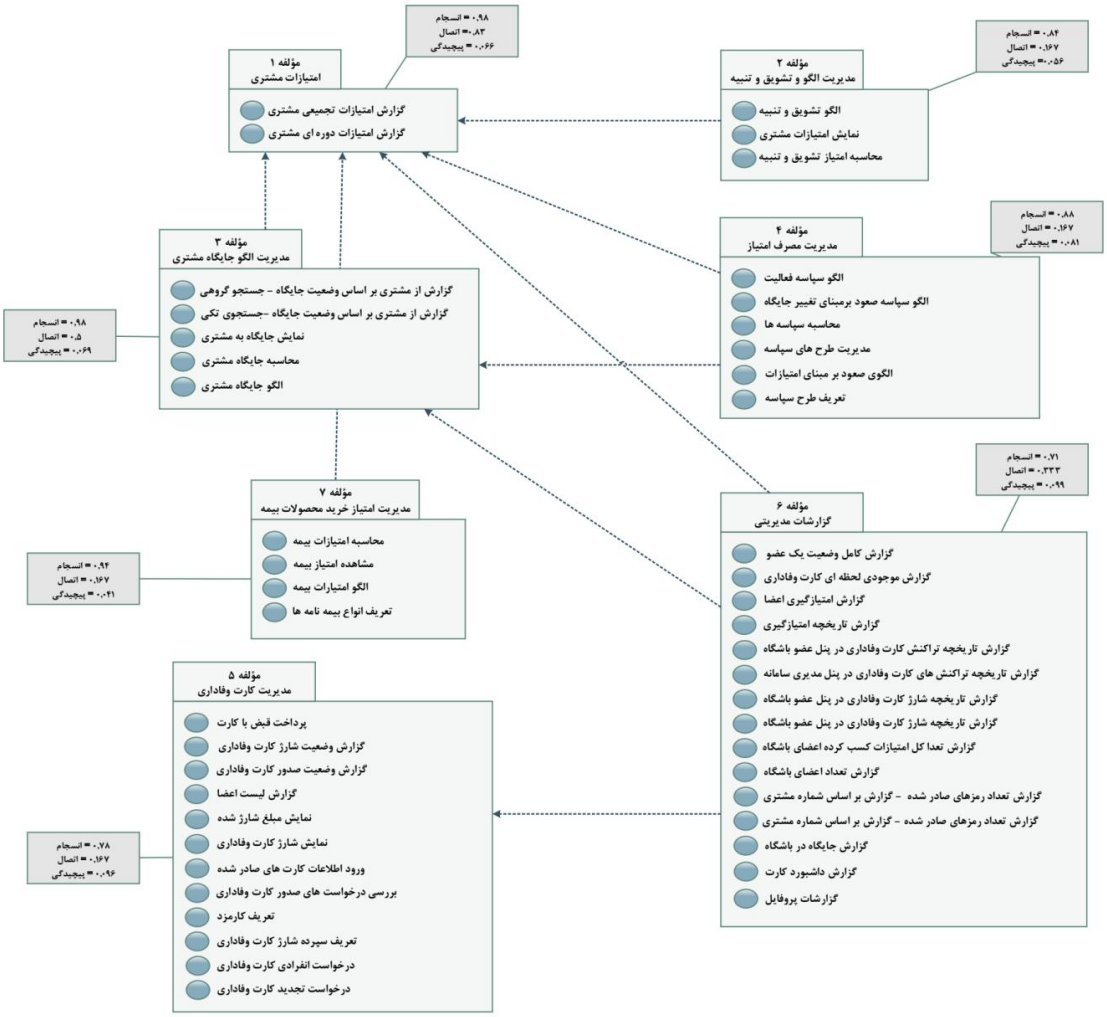
با توجه به توضیحات ارائه‌شده در خصوص انتخاب جواب نهایی از میان جواب‌های موجود در جبههٔ اول تنها یک جواب با غلبه بر سایر جواب‌های موجود در جبهه‌های دیگر انتخاب شده است که مقدار Software Cohesion، Software Coupling و Software Complexity آن در جدول (۶)، مقادیر جزئی انسجام، اتصال و پیچیدگی برای هر مؤلفهٔ موجود در ساختار آن در جدول (۷) و همچنین ساختار مؤلفه‌های جواب نهایی در شکل (۱۳) به نمایش درآمده است. همان طور که در شکل مشاهده می‌شود، ساختار به‌دست‌آمده از اجرای الگوریتم بسیار مشابه ساختار تعیین‌شده توسط خبره است و تفاوت موردهای کاربردی دو ساختار در شکل (۱۳) با رنگ تیره نمایش داده شده است که این تفاوت شامل ۶ مورد کاربرد از ۴۶ مورد کاربرد است.

جدول (۶): مقادیر توابع هدف و تعداد مؤلفه برای جواب نهایی

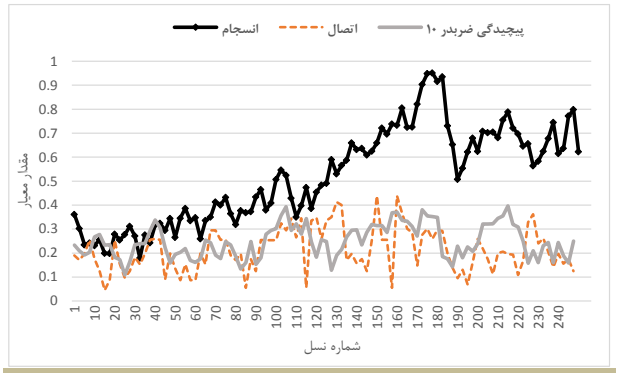
تعداد مؤلفه	انسجام سیستم	اتصال سیستم	پیچیدگی سیستم
۶	۰/۹۰	۰/۱۴۸	۰/۰۲۷۱

جدول(۷): میزان توابع هدف برای هر مؤلفه جواب نهایی به‌دست‌آمده توسط روش پیشنهادی

انسجام مؤلفه	اتصال مؤلفه	پیچیدگی مؤلفه
۰/۹۵	۰/۶	۰/۰۷۹
۰/۹۸	۰/۶	۰/۰۶۹
۰/۹۱	۰/۴	۰/۰۸۱
۰/۹۹	۰	۰/۰۳۳
۰/۷۶	۰/۴	۰/۰۹۹
۰/۸۱	۰	۰/۰۷۸

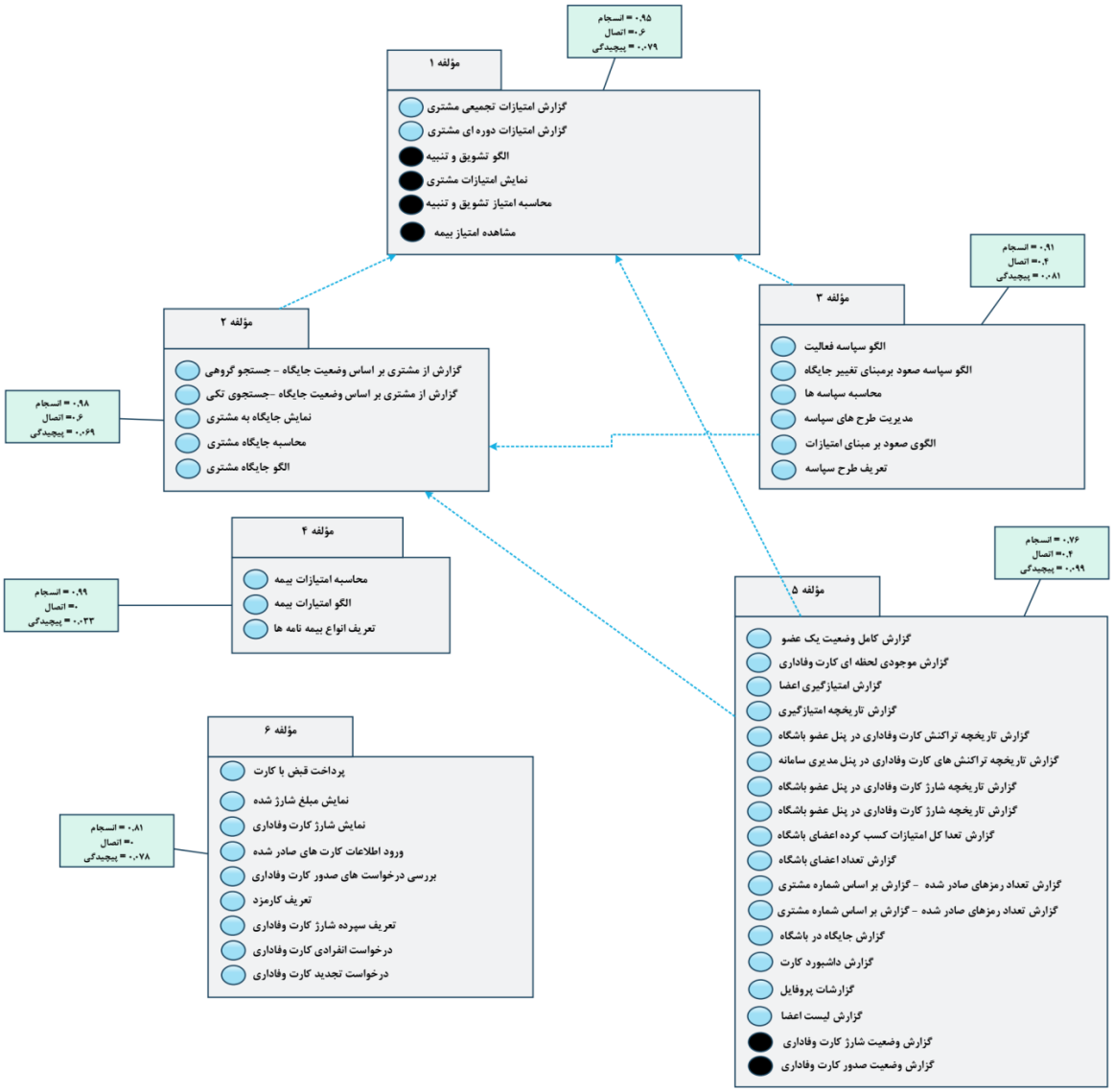


شکل (۱۱): ساختار مؤلفه‌های تعیین‌شده توسط خبره برای سیستم مورد مطالعه

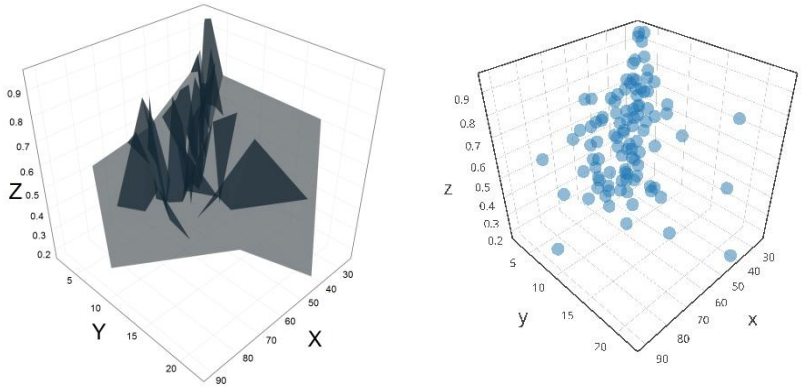


شکل (۱۲): میانگین جبهۀ اول پارتو بر هر معیار

برای دید بهتر، نمایش سه‌بعدی هریک از سه معیار استفاده‌شده برای میانگین جبهۀ اول پارتو در شکل (۱۴) نمایش داده شده است. البته در این شکل برای نمای گرافیکی بهتر مقادیر معکوس معیار اتصال و پیچیدگی نشان داده شده‌اند. بنابراین در این شکل هرچه مقدار هر سه معیار بیشتر باشد، به جواب‌های بهینه‌تری دست می‌یابیم. در شکل (۱۴)، همان طور که مشاهده می‌شود، معیار انسجام با اتصال و اتصال با پیچیدگی دارای تناقض هستند و زمانی که بیشترین مقدار انسجام (محور Z) در یک راه‌حل ارائه شده، در همان راه‌حل کمترین اتصال (یا به تعبیری بیشترین معکوس اتصال- محور Y) حاصل شده است. برای نمونۀ دیگر، زمانی که بیشترین پیچیدگی در یک راه‌حل وجود دارد (یا به‌تعبیری کمترین معکوس پیچیدگی- محور X)، در همان راه‌حل کمترین میزان اتصال (یا به‌تعبیری بیشترین معکوس اتصال- محور Y) حاصل شده است. بنابراین این نتایج نشان می‌دهد که بین توابع مختلف برازندگی، مصالحه وجود دارد و استفاده از یک روش چندهدفه بسیار مؤثر است.



شکل (۱۳): ساختار مؤلفهٔ جواب نهایی به‌دست‌آمده توسط الگوریتم پیشنهادی



شکل (۱۴): مقایسهٔ معیارهای Z (نشان‌دهندهٔ معیار Software Cohesion)، Y (نشان‌دهندهٔ معکوس معیار Software Coupling) و X (نشان‌دهندهٔ معکوس معیار Software Complexity) برای میانگین جبهه‌های اول پارتو

در این مقاله به‌جز مقادیر توابع هدف که خود معیاری در جهت ارزیابی جواب به‌دست‌آمده است، نتیجهٔ نهایی حاصل‌شده را با ساختار مؤلفه‌های تعیین‌شده توسط خبره مقایسه خواهیم شد. گفتنی است نویسندگان برای ارزیابی مؤلفه‌های تشخیص داده‌شده توسط الگوریتم با مؤلفه‌های تعیین‌شده توسط خبره، از فرمولی کمک گرفته‌اند به نام معیار کیفیت [۲۰] ، که در فرمول (۱۲) ارائه شده است:

$$Q(A, ExpertComponent) = \left(1 - \frac{Mojo(A, ExpertComponent)}{n}\right) \times 100 \qquad (۱۲)$$

در این فرمول، *Mojo* کمترین تعداد موردهای کاربردی را که نیاز است تغییر کنند تا راه‌حل به‌دست‌آمده توسط الگوریتم تبدیل به راه‌حل تعیین‌شده توسط خبره شود، محاسبه می‌کند و n هم تعداد کل موردهای کاربردی سیستم است. هرچه میزان Q برای یک راه‌حل بالاتر باشد، نشان‌دهندهٔ بیشترین میزان شباهت به راه‌حل تعیین‌شده توسط خبره است.

جداول (۸) و (۹) مقادیر توابع هدف و مقدار معیار کیفیت را برای پاسخ منتخب و نظر خبره نمایش می‌دهد.

جدول (۸): مقادیر توابع هدف ساختار جواب تعیین‌شده و خبره

ساختار مؤلفه	انسجام سیستم	اتصال سیستم	پیچیدگی سیستم
NSGA-II	۰/۹۰	۰/۱۴۸	۰/۰۲۷۱
خبره	۰/۸۷	۰/۱۵۷	۰/۰۲۸۹

جدول (۹): میزان توابع هدف برای هر مؤلفهٔ جواب نهایی

تعیین‌کننده	معیار کیفیت	مجموع میزان تفاوت موردهای کاربردی در ساختار الگوریتم با خبره
NSGA-II	٪۸۷/۲	۶
خبره	٪۱۰۰	۰

در خصوص نتایج جدول (۹) نیز به این نتیجه می‌توان رسید که جواب به‌دست‌آمده تا حد زیادی به ساختار تعیین‌شده توسط خبره شباهت دارد و تنها با تغییر شش مورد کاربردی، جواب به‌دست‌آمده توسط الگوریتم پیشنهادی، تبدیل به ساختار مؤلفهٔ تعیین‌شده توسط خبره می‌شود. در جدول (۱۰) روش استفاده‌شده با سایر روش‌های ارائه‌شده در زمینهٔ تشخیص مؤلفه مقایسه شده است. از میان روش‌های ارائه‌شده روش پیشنهادی این مقاله با روش‌های خوشه‌بندی [۳] ، شبکهٔ عصبی و خوشه‌بندی فازی [۱۰]، مبتنی بر PSO [۹]، SBLCI [۲۱]، SCI-GA [۱۳] و CCIC [۱۲] مقایسه شده است؛ علت این انتخاب هم نزدیکی ایده و فرآورده‌های مورد استفاده در روش‌ها بوده است. در روش SBLCI [۲۱] مؤلفه‌های منطقی یک سیستم را از مدل‌های تجزیه و تحلیل آن با استفاده از الگوریتمی پیشنهادی بر مبنای الگوریتم ژنتیک شناسایی می‌کند به این ترتیب که با طرحی تکراری ابتدا مؤلفه‌های سطح بالا و سپس زیرمجموعه‌های سطح پایین را برای هر مؤلفه شناسایی می‌کند؛ لذا در ارزیابی انجام‌شده (جدول ۱۰) فقط سطح اول مؤلفه‌ها در نظر گرفته شده است.

جدول (۱۰): مقایسهٔ توابع هدف و معیار کیفیت با روش‌های دیگر

روش	انسجام سیستم	اتصال سیستم	پیچیدگی سیستم	معیار کیفیت
روش پیشنهادی مبتنی بر NSGA-II	۰/۹۰	۰/۱۴۸	۰/۰۲۷۱	٪۸۷/۲
CCIC [۱۲]	۰/۸۱	۰/۱۸۱	۰/۰۲۹۹	٪۷۹/۶
SCI-GA [۱۳]	۰/۷۴	۰/۲۰۵	۰/۰۳۵۵	٪۷۲/۵
SBLCI [۲۱]	۰/۷۷	۰/۱۹۴	۰/۰۳۰۸	٪۶۷/۹
مبتنی بر PSO [۹]	۰/۷۱	۰/۲۱۸	۰/۰۳۴۵	٪۶۹/۵
شبکهٔ عصبی و خوشه‌بندی فازی [۱۰]	۰/۷۹	۰/۱۸۵	۰/۰۳۱۰	٪۷۴/۱
خوشه‌بندی [۳]	۰/۵۵	۰/۳۸۵	۰/۰۶۷۲	٪۵۷/۲

در روش‌های خوشه‌بندی [۳] ، شبکهٔ عصبی و خوشه‌بندی فازی [۱۰]، مبتنی بر PSO [۹] و SCI-GA [۱۳] از موردهای کاربری برای ارزیابی استفاده شده است اما در روش CCIC [۱۲]، که با استفاده از کلاس‌های تجزیه و تحلیل، مؤلفه‌های منطقی نرم‌افزار را شناسایی می‌کند، در ارزیابی انجام‌شده (جدول ۱۰)، از کلاس‌های تحلیلی مورد مطالعه برای شناسایی مؤلفه‌ها استفاده کردیم.

همان طور که در جدول (۱۰) مشاهده می‌شود، روش پیشنهادی مبتنی بر NSGA-II این مقاله توانسته با به‌کارگیری ایدهٔ بهینه‌سازی چندهدفه نسبت به سایر روش‌های مشابه بهینه‌سازی تک‌هدفه، نتایج بهتری از نگاه معیارهای اتصال، انسجام و پیچیدگی به دست بیاورد؛ به‌علاوه اینکه خروجی

این روش به طراحی خبره نزدیکی بیشتری نسبت به سایر روش‌های مقایسه‌شده دارد.

در جدول (۱۱)، روش‌های شناسایی مؤلفهٔ معماری که مبتنی بر تکاملی هستند، از لحاظ تعداد تکرار، زمان همگرایی و متوسط کرموزوم‌های غیرتکراری به کل در هر نسل، مقایسه شده‌اند. نتایج این ارزیابی نشان می‌دهد که روش پیشنهادی با وجودی که دارای زمان بیشتری نسبت به بعضی از روش‌های مشابه برای همگرایی است، توانسته تنوع بیشتری در کرموزوم‌ها در هر نسل ایجاد کند.

جدول (۱۱): مقایسهٔ تعداد تکرار و زمان همگرایی و متوسط نسبت کرموزوم‌های غیرتکراری به کل در هر نسل

روش	متوسط تعداد تکرار همگرایی	متوسط زمان همگرایی (ثانیه)	متوسط نسبت کرموزوم‌های غیرتکراری به کل در هر نسل
روش پیشنهادی مبتنی بر NSGA-II	۳۲۱	۳۱۹۵۸	٪۵۸/۳
CCIC [۱۲]	۱۲۵۳	۷۵۲۶۹	٪۴۵/۲
SCI-GA [۱۳]	۷۵۲	۲۸۷۹۰	٪۲۸/۱
SBLCI [۲۱]	۱۱۲۱	۴۵۲۲۱	٪۳۳/۲
مبتنی بر PSO [۹]	۱۸۵	۹۵۳۶	٪۳۸/۷

۴. نتیجه‌گیری

در این مقاله به بررسی استفاده از یکی از الگوریتم‌های چندهدفه با نام NSGA-II در تشخیص مؤلفه‌های منطقی یک سیستم نرم‌افزاری پرداخته شده است؛ علت این امر عدم استفاده از الگوریتم‌های تکاملی چندهدفه در این حوزه بوده است؛ اگرچه ساختار مسئلهٔ تشخیص مؤلفهٔ خود نیز یک مسئلهٔ چندهدفه است، اهداف اصلی این الگوریتم بهبود مقادیر انسجام، اتصال و پیچیدگی ساختار مؤلفه‌ای یک سیستم نرم‌افزاری در مقایسه با ساختار تعیین‌شده توسط خبره می‌باشد.

بررسی کارایی روش استفاده‌شده نشان می‌دهد این الگوریتم به‌رغم پیچیدگی پیاده‌سازی آن در این حوزه کارآمد بوده و می‌تواند جواب‌های بهینه و تا حد زیادی نزدیک به نظر خبره را به دست آورد. پیش‌بینی می‌شود که سایر الگوریتم‌های تکاملی چندهدفه نیز بتوانند جواب‌هایی بهینه و مطلوب در حوزهٔ مسئلهٔ تشخیص مؤلفه به دست آورند. به عنوان کارهای آتی، برای شناسایی مؤلفه‌های معماری نرم‌افزار، می‌توان معیارهای دیگر مؤثر در معماری نرم‌افزار مانند قابلیت استفادهٔ مجدد مؤلفه‌ها، کارایی، قابلیت اطمینان را در شناسایی مؤلفه‌ها به‌صورت تابع برازندگی جدید اضافه کرد. همچنین بیشتر کارهای انجام‌شده در معماری در حوزهٔ منظر منطقی معماری بوده است و به‌عنوان کار آتی دیگر، می‌توان دو منظر مؤلفه و اتصال در زمان اجرا و منظر استقرار معماری را توسط روش‌های جست‌وجوبنیان پیشنهاد داد و جواب‌های بهینه را برای آن منظرها شناسایی کرد.

پیوست (۱): ارتباط بین موردهای کاربرد و کنشگر ها و کلاس های موجودیتی سامانه باشگاه مشتریان

		نام کنشگر یا کلاس موجودیت																							
نام مورد کاربرد	شماره	مدیر سامانه	سامانه	عضو باشگاه	امتیاز	مشتری	کارت وفاداری	سپرده	طرح سپاسه	الگوی سپاسه صعود	الگوی سپاسه فعالیت	الگو جایگاه	الگو تنبیه و تشویق	جایگاه	گروه	نردبان	هدیه	شمعه	کارمزد کارت	کارمزد تجدید کارت	گروه اطلاعاتی	بیمه نامه	الگو امتیازات نامه
		AC۱	AC۲	AC۳	EC۱	EC۲	EC۳	EC۴	EC۵	EC۶	EC۷	EC۸	EC۹	EC۱۰	EC۱۱	EC۱۲	EC۱۳	EC۱۴	EC۱۵	EC۱۶	EC۱۷	EC۱۸	EC۱۹
گزارش امتیازات تجمیعی مشتری	UC۱	۱			۱	۱	۱	۱															
گزارش امتیازات دوره ای مشتری	UC۲	۱			۱	۱	۱	۱															
الگو تنبیه و تشویق	UC۳	۱											۱	۱	۱	۱							
محاسبه امتیاز تشویق و تنبیه	UC۴		۱		۱								۱	۱	۱								
نمایش امتیاز مشتری	UC۵			۱	۱	۱																	
الگو جایگاه مشتری	UC۶	۱				۱						۱		۱	۱	۱							
محاسبه جایگاه مشتری	UC۷		۱		۱	۱						۱		۱	۱	۱							
نمایش جایگاه به مشتری	UC۸			۱		۱						۱		۱	۱	۱							
گزارش از مشتری بر اساس وضعیت جایگاه (جستجوی گروهی)	UC۹	۱			۱	۱								۱	۱	۱							
گزارش از مشتری بر اساس وضعیت جایگاه (جستجوی تکی)	UC۱۰	۱			۱	۱								۱	۱								
الگوی سپاسه فعالیت	UC۱۱	۱			۱						۱			۱	۱	۱	۱						
الگوی سپاسه صعود (بر مبنای امتیازات)	UC۱۲	۱			۱					۱					۱		۱						
الگوی سپاسه صعود (بر مبنای جایگاه)	UC۱۳	۱								۱				۱	۱	۱	۱						
تعریف طرح سپاسه	UC۱۴	۱							۱	۱	۱			۱	۱	۱	۱						
مدیریت طرح های سپاسه	UC۱۵	۱			۱				۱	۱	۱			۱	۱	۱	۱						
محاسبه سپاسه	UC۱۶		۱		۱					۱	۱			۱	۱		۱						
درخواست انفرادی کارت وفاداری	UC۱۷			۱		۱	۱	۱										۱	۱				
پرداخت کارمزد	UC۱۸			۱		۱	۱											۱	۱				
عملیات درخواست تجدید کارت	UC۱۹			۱		۱	۱	۱										۱	۱	۱			
گزارش لیست اعضا	UC۲۰	۱				۱	۱								۱								
بررس درخواست های صدور کارت وفاداری و عودت کارمزد به مشتری	UC۲۱	۱				۱	۱	۱										۱	۱				
تعیین کارمزد صدور کارت و صدور تجدید کارت وفاداری	UC۲۲	۱					۱								۱				۱	۱			
ورود اطلاعات کارت های صادر شده	UC۲۳	۱				۱	۱								۱								
نمایش شارژ امتیاز کارت وفاداری	UC۲۴	۱				۱	۱		۱								۱						
نمایش مبلغ شارژ شده در کارت وفاداری	UC۲۵			۱		۱	۱		۱								۱						
تعریف سپرده های باشگاه مشتریان	UC۲۶	۱					۱	۱	۱														
گزارش وضعیت شارژ کارت وفاداری	UC۲۷	۱				۱	۱								۱								
گزارش وضعیت صدور کارت وفاداری	UC۲۸	۱				۱	۱	۱									۱						
گزارشات پروفایل	UC۲۹	۱				۱	۱							۱	۱						۱		
گزارش امتیازگیری اعضا	UC۳۰	۱			۱	۱	۱							۱	۱								
گزارش کامل وضعیت یک عضو	UC۳۱	۱				۱	۱							۱	۱						۱		
گزارش تاریخچه تراکنش کارت های وفاداری در پنل مدیر سامانه	UC۳۲	۱				۱	۱																
گزارش تاریخچه شارژ کارت های وفاداری در پنل مدیر سامانه	UC۳۳	۱				۱	۱																
گزارش تغییرات تعداد اعضای باشگاه	UC۳۴	۱													۱								
گزارش تغییرات نمودار کل امتیازات کسب کرده اعضا در باشگاه	UC۳۵	۱													۱								

گزارش داشبورد کارت	UC۳۶	۱														۱	۱						
گزارش تاریخچه امتیازگیری	UC۳۷				۱		۱	۱									۱	۱					
گزارش تاریخچه تراکنش کارت های وفاداری در پنل عضو باشگاه	UC۳۸				۱											۱	۱						
گزارش تاریخچه شارژ کارت های وفاداری در پنل عضو باشگاه	UC۳۹				۱		۱	۱								۱	۱	۱					
گزارش تعداد رمزهای صادر شده (گزارش بر اساس کد شعبه)	UC۴۰	۱					۱										۱					۱	
گزارش تعداد رمزهای صادر شده (گزارش بر اساس شماره مشتری)	UC۴۱	۱															۱						
گزارش لحظه ای موجودی کارت وفاداری	UC۴۲				۱											۱	۱						
تعریف انواع بیمه نامه	UC۴۳	۱																			۱		
الگو امتیازات بیمه	UC۴۴	۱						۱													۱	۱	۱
محاسبه امتیازات بیمه	UC۴۵		۱					۱	۱								۱					۱	
مشاهده امتیاز	UC۴۶			۱				۱	۱								۱					۱	۱

مراجع

[1] Gholamshahi S. and Hasheminejad S. M. H., "*Software component identification and selection: A research review*", Software: Practice and Experience, vol. 49, no. 1, pp. 40-692, 2019.

[2] Kim J., Park S., and Sugumaran V., "*DRAMA: a framework for domain requirements analysis and modeling architectures in software product lines*", Journal of Systems and Software, vol. 81 no. 1, pp. 37-55, 2008.

[3] Shahmohammadi G., Jalili S., and Hasheminejad S. M. H., "*Identification of system software components using clustering approach*", J Object Technol, vol. 9, no. 6, pp. 77-98, 2010.

[4] Albani A., Overhage S. and Birkmeier D., "*Towards a systematic method for identifying business components*", In Proceedings of CBSE, LNCS 5282, vol. 1, pp. 262–277, 2008.

[5] Lee S.D., Yang Y.J., Cho E.S., Kim S.D. and Rhew S.Y., "*COMO: A UMLBased Component Development Methodology*", In Proceedings of the 6th Asia Pacific Software Engineering Conference, Washingt on, DC, USA, IEEE Computer Society, Los Alamitos, pp. 54-61, 1999.

[6] Ganesan R. and Sengupta S., "*O2BC: a Technique for the Design of Component-Based Applications*", In Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, pp. 46-55, 2001.

[7] Hamza H. S., "*A Framework for Identifying Reusable Software Components Using Formal Concept Analysis*", in Proceeding of the 6th International Conference on Information Technology: New Generations, pp. 813-818, 2009.

[8] Räihä O., "*A survey on search-based software design*", Computer Science Review, vol. 4, pp. 203-249, 2010.

[9] Hashemi S. H. and Shahmohammadi G. R., "*Detection system software components using a hybrid algorithm*", ANDRIAS J., vol. 40, no. 2, pp. 57-63, 2015.

[10] Ahmadzadeh M., Shahmohammadi G. R. and Shayesteh M., "*Identification of software systems components using a self-organizing map competitive artificial neural network based on cohesion and coupling*", ANDRIAS J., vol. 40, no. 3, pp. 642-651, 2016.

[11] Deb K., Pratap A., Agarwal S. and Meyarivan T., "*A fast and elitist multiobjective genetic algorithm: NSGA-II*", In IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, 2002.

[12] Hasheminejad S.M.H. and Jalili S., "*CCIC: Clustering analysis classes to identify software components*", In Journal of Information and Software Technology, vol. 57, pp. 329-351, 2015.

[13] Hasheminejad S. M. H. and Jalili, S., "*SCI-GA: Software Component Identification using Genetic Algorithm*", In Journal of Object Technology (JOT), vol. 12, no. 2, pp. 1-34, 2013.

[14] Chidamber S. R. and Kemerer C. F., "*A metrics suite for object oriented design*", In Journal IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994.

[15] Gorton I., *Essential software architecture*, vol. 14. Springer, 2006.

[16] Zuse H., *Software complexity: measures and methods*, Walter de Gruyter & Co., 1990.

[17] Bruegge B. and Dutoit A. H., *Object-Oriented Software Engineering Using UML, Patterns and Java*, Prentice Hall, 2004.

[18] Karner G., *Resource Estimation for Objectory Projects*, Objectory Systems, 1993.

[19] AlSharif M., Bond W.P. and Al-Otaiby T.,

"*Assessing the Complexity of Software Architecture*", In Proceedings of the 42nd annual Southeast regional conference, ACM, pp. 98-103, 2004.

[20] Tzerpos V. and Holt R.C., "*MoJo: A distance metric for software clustering*", In Proceedings of the 6th Working Conference on Reverse Engineering, pp. 187-193, 1999.

[21] Hasheminejad S.M.H. and Jalili, S., "*An Evolutionary Approach to Identify Logical Components*", In Journal of Systems and Software, vol. 96, no.1, pp. 24–50, 2014.