دانشکده فنی و مهندسی

کارشناسی ارشد مهندسی کامپیوتر نرم‌افزار

گروه مهندسی کامپیوتر و فناوری اطلاعات

گزارش درس سمینار

موضوع:

مدیریت پیچیدگی در نرم افزار علمی

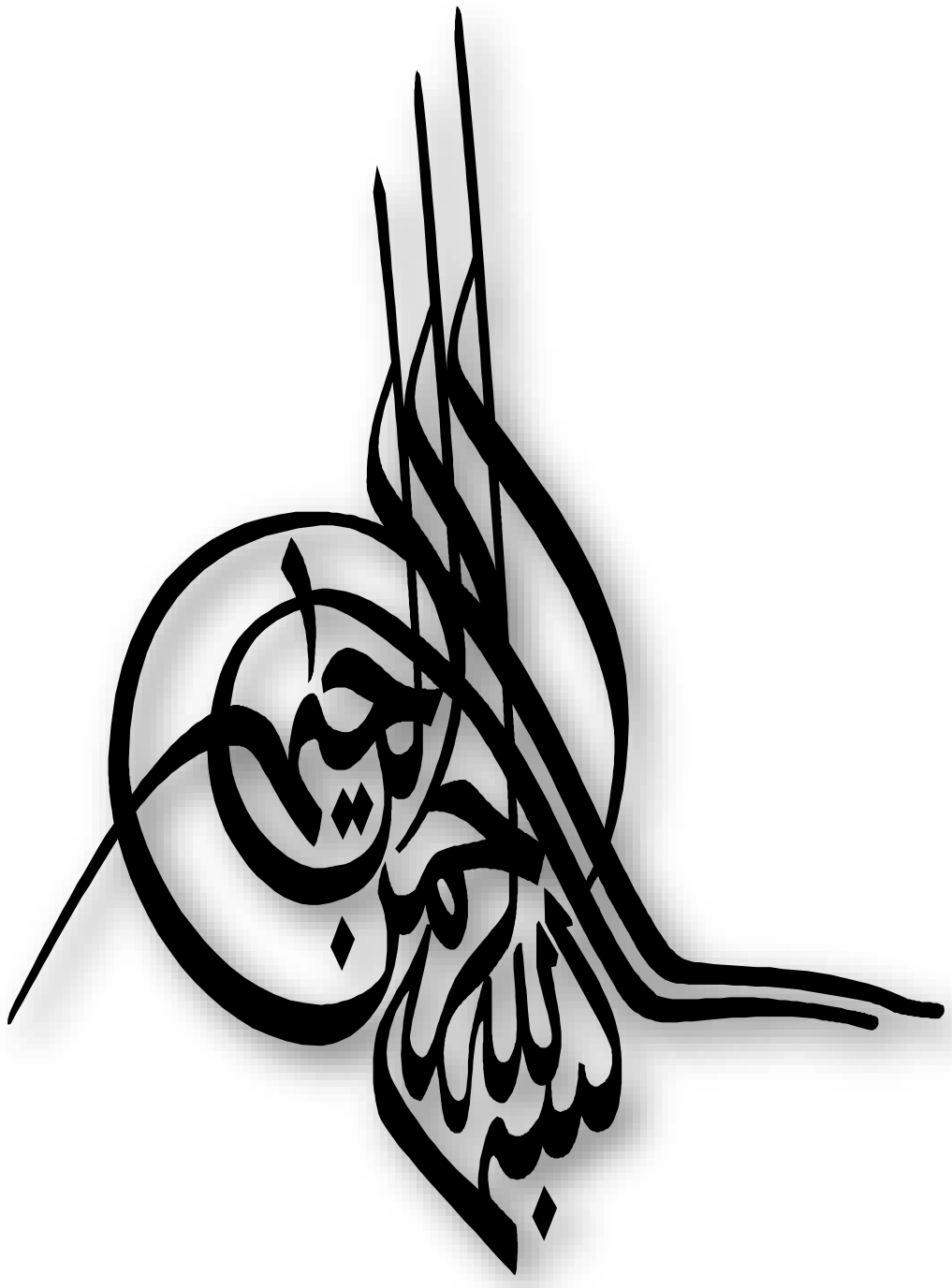
نگارش:

بهاره میرزاخانی

استاد راهنما:

دکتر رضوی ابراهیمی

شهریور ۱۴۰۰


بسم الله الرحمن الرحيم

چکیده

یکی از مزایای مورد انتظار طراحی مدولار ، انعطاف پذیری است. منظور ما از کلمه "انعطاف پذیری" امکان تغییرات شدید در یک ماژول بدون تغییر یا بدون اطلاع از سایر ماژول ها است. بر اساس داده های تکاملی موجود در سیستم های کنترل نسخه ، می توان کیفیت معماری نرم افزار مدولار را تجزیه و تحلیل کرد و تصمیم گرفت که آیا ارزش بازسازی طراحی آن را دارد یا خیر. در این پایان نامه ما این مسئله را با استفاده از یک رویکرد جدید مبتنی بر یک نظریه کلی مدولار که از ماتریس ساختار طراحی (DSM) برای استدلال در مورد ویژگی های کیفیت استفاده می کند ، بررسی می کنیم. با استفاده از روش ما می توانیم توابع را در رده های مختلف طبقه بندی کنیم. این یافته نشان می دهد که تجزیه و تحلیل سطوح مختلف عملکردهای یک سیستم نرم افزاری ممکن است راهنمای توسعه دهندگان در کار چالش برانگیز طراحی مجدد نرم افزار با تشخیص و بازیابی اجزایی باشد که می توانند در پروژه های نرم افزاری دیگر مورد استفاده مجدد قرار گیرند.

کلمات کلیدی:

نرم افزار محاسبات علمی، پیچیدگی در نرم افزار، رابطه و وابستگی، ماتریس ساختار طراحی.

فهرست مطالب

عنوان ..... صفحه

فهرست اشکال

عنوان ..... صفحه

فهرست جداول

عنوان ..... صفحه

فهرست علائم اختصاری

شکل مثلثی مسدود	BTF: Block Triangular Form
ماتریس نقشه برداری دامنه	DMM: Domain Mapping Matrix
ماتریس ساختار طراحی	DSM: Design Structure Matrix
جستجوی موضوعی ناشی از ابرمتن	HITS: Hypertext Topic Search
ماتریس چند دامنه	MDM: Multi-Domain Matrix
تجزیه مقدار منفرد	SVD: Singular Value Decomposition

فصـــل اول

معرفی

امروزه سازمان ها با پیچیدگی های بسیاری در داخل و محیط اطرافشان روبرو هستند که روش های برخورد با پیچیدگی ها و مدیریت آنها به یک مزیت رقابتی کلیدی برای آنها تبدیل شده است. از طرفی مدیریت دانش از راه های بهبود بخشیدن عملکرد سازمانی و بقای سازمان می باشد و بسیاری از سازمان ها سیستم مدیریت دانش را پیاده سازی کرده اند و به آن اهمیت داده اند. سیستم های نرم افزاری را می توان به عنوان شبکه ای از اجزا که با روابط وابستگی به هم پیوسته اند ، مشاهده کرد. در نرم افزار و سایر سیستم های تکنولوژیکی مانند محصول فرآیند یا معماری سازمانی ، برخی از عملکردهای آن توسط الگوی تعامل اجزا یا زیرسیستم ها قابل درک است [۷]. به عنوان مثال ، سیستم های نرم افزاری مدولار امکان ردیابی اشکالات را در تعداد کمی از زیر سیستم ها یا ماژول ها مشخص می کنند.

بسیاری از نرم افزارهای علمی معمولاً توسط متخصصان حوزه نوشته می شوند و برخی از مشکلات محاسبات علمی خاص را برطرف می کنند. به عنوان مثال نرم افزار CSparse پیاده سازی شده در C مربوط به حل سیستم معادله خطی $Ax = b$ است که در آن ماتریس ضریب A پراکنده است. ADOL-C یک سیستم نرم افزاری برای محاسبه مشتقات ریاضی (ضرایب گرادیان ، ژاکوبیان ، هسیان ، تیلور) از یک عملکرد ریاضی است که به عنوان یک برنامه رایانه ای در یک زبان برنامه نویسی (C) در دسترس است. این نرم افزار از تکنیک های تمایز الگوریتمی برای محاسبه مشتقات عددی دقیق (تا ماشین دقیق) برنامه عملکرد در یک نقطه مشخص استفاده می کند.

یک ابزار مناسب برای نشان دادن و تجزیه و تحلیل پیچیدگی معماری این نرم افزارها که در بین مهندسین سیستم و معماران رایج است ، به اصطلاح ماتریس طراحی ساختار (DSM) و ماتریس نقشه برداری دامنه (DMM) و ماتریس چند دامنه (MDM) است [۷]. از آنجا که DSM را می توان با ماتریسی در $R^{m\times n}$ نشان داد ، می توان با روش های جبری خطی پیچیده مانند تجزیه ارزش واحد (SVD) تجزیه و تحلیل کرد. علاوه بر این ، با استفاده از دوگانگی یک ماتریس پراکنده و نمودار آن ، ماتریس پراکنده A را می توان (از طریق جایگزینی P) به شکل محاسباتی سودمندی به نام "شکل مثلثی بلوکی P^T AP "(BTF) که در انجمن DSM به عنوان "تقسیم بندی" نیز شناخته می شود ، مرتب کرد.

پروفسور حسین و گروهش استفاده از DSM را برای مدل سازی و تحلیل پیچیدگی وابستگی سیستم های نرم افزاری علمی پیشنهاد کرده اند [۱۴ ، ۱۵ ، ۱ ، ۱۳]. یکی از انگیزه های اصلی مطالعه کد قدیمی برای تعیین اطلاعات وابستگی از طریق نمودارهای تماس استاتیک بود. اطلاعات جمع آوری شده می تواند برای تجدید ساختار یا استفاده مجدد از اجزاء با تجزیه و تحلیل اطلاعات وابستگی با استفاده از تکنیک های شبکه های پیچیده در حال ظهور مورد استفاده قرار گیرد [۹].

پروفسور حسین و گروهش ویژگی های معماری مجموعه کوچکی از نرم افزارهای علمی نمایشی را مطالعه کرده اند [۱۵]. ابزارهای نرم افزاری مورد مطالعه ، طول مسیر کوتاه تر و گره های کوچک و هزینه های انتشار مشابه نرم افزارهای عمومی مانند سیستم عامل ها را نشان می دهند [۴ ، ۱۹].

به دلایل مختلف ، نرم افزارهای قدیمی ممکن است دارای اسناد فنی کافی نباشند ، به طوری که از دیدگاه قابلیت استفاده ، تشخیص و بازیابی اجزایی که در پروژه های دیگر نرم افزاری مورد استفاده مجدد قرار می گیرند ، دشوار باشد. در این پایان نامه ما سیستم های نرم افزاری را مطالعه می کنیم که به طور خاص برای مشکلات ناشی از برنامه های علمی و مهندسی طراحی شده اند [۱۶].

ابزارهای تجزیه و تحلیل مانند "درک" [۲۱] به ما امکان می دهد ساختار وابستگی نرم افزار را در سطوح مختلف جزئیات مشاهده کنیم: فایل ، کلاس ، عملکرد ، بیانیه و غیره. در کار ما ساختارهای وابستگی برنامه هایی را که در آن تماس گیرنده بین عملکردها ارتباط برقرار می کند ، تجزیه و تحلیل می کنیم. ضبط شده توسط نمودار تماس استاتیک جریان کنترل اساسی در برنامه را به تصویر می کشد.

نمودارهای تماس استاتیک به نمایش وابستگی های مستقیم بین عناصر طراحی (در مورد ما توابع) محدود می شوند. در این مقاله ما علاقه مند به کشف "شباهت" بین عناصر طراحی هستیم. سپس می توانیم از یک متریک تشابه مناسب برای تقسیم بندی عناصر طراحی بین گروه ها یا خوشه ها استفاده کنیم که در آن عناصر در یک گروه به روش خاصی "مشابه" هستند. کاربرد فوری چنین تجزیه ای ، توانایی بازیابی گروهی از توابع "مشابه" از مخزن نرم افزار است. در ترکیب با مفهوم "اهمیت" عناصر طراحی [۱۳] ، هدف ما در این کار گروه بندی یا دسته بندی عناصر طراحی به "ردیف" بر اساس "اهمیت" آنها است.

۱-۱ مقدمه

۱-۱-۱ نرم افزار محاسبات علمی

هزاران تا میلیون ها خط منبع کد ، سیستم های نرم افزاری را به عنوان محصولی پیچیده می سازد. سیستم نرم افزاری به تصمیمات طراحی بستگی دارد محدودیت های داخلی و خارجی مسائل مختلف فنی و غیر فنی [۱۱]. توسعه برنامه های کاربردی نرم افزار محاسبات علمی به عنوان اثبات ابزار مفهومی در نظر گرفته می شود. اما منابع سخت افزاری قوی نرم افزارهای علمی را برای حل و شبیه سازی مشکلات بزرگ تسهیل می کند. توسعه اخیر منابع سخت افزاری قوی تر باعث افزایش تعداد برنامه های علمی می شود که به طور کارآمدتر از برنامه های ساخته شده قبل شبیه سازی می شوند [۱۵]. این نرم افزارهای شبیه سازی بسیار پیچیده هستند و حاوی میلیون ها خط کد کامپیوتر هستند. این برنامه ها سرمایه گذاری قابل توجهی در زمان و دیگر منابع محاسباتی و نیروی

انسانی دارند. قابلیت استفاده مجدد ، کارایی ، قابلیت حمل ، صحت ، استحکام و سهولت استفاده از ویژگی های مختلف نرم افزار علمی است.

یک یا چند ماژول مستقل توسعه یافته یک سیستم نرم افزاری را تشکیل می دهند. ما می توانیم هر ماژول را به عنوان بخشی از نرم افزار در نظر بگیریم. از طریق این ماژول ها می توانیم وابستگی های آنها را پیدا کنیم. به عنوان مثال ما می گوییم ماژول A بستگی به ماژول B دارد وقتی که ماژول A از ماژول B استفاده می کند (فراخوانی می کند)؛ در اینجا ما دو نوع وابستگی نرم افزاری را توصیف می کنیم: استاتیک و پویا. وابستگی های استاتیک از کدی که در حالت اجرا نیست استخراج می شود و از کد منبع به عنوان ورودی استفاده می کند. وابستگی های پویا از کد در حالت اجرا استخراج می شوند و از کد اجرایی و حالت برنامه به عنوان ورودی استفاده می کنند. مشکل وابستگی پویا وجود برخی از زیرروالین است که فقط در زمان اجرا می شوند. مزیت در نظر گرفتن وابستگی استاتیک استفاده از کد منبع به عنوان ورودی است و به وضعیت برنامه وابسته نیست. به همین دلیل است که ما در کار خود وابستگی استاتیک را در نظر گرفته ایم.

۲-۱-۱ معماری نرم افزار

معماری سازمان اساسی یک سیستم است که در اجزای آن روابط آنها با یکدیگر و محیط و اصول راهنمای طراحی و تکامل آن تجسم یافته است [۲۰]. معماری نرم افزاری یک برنامه یا سیستم محاسباتی ، ساختار یا ساختارهای سیستم است که شامل عناصر نرم افزاری است که ویژگی های خارجی آن عناصر و روابط بین آنها را در بر می گیرد [۲].

پیچیدگی سیستم های نرم افزاری بزرگ را می توان با استفاده از معماری نرم افزار به راحتی تشخیص داد. برای یک سیستم نرم افزاری ، معماری آن به عنوان ساختار سطح بالا در نظر گرفته می شود. به عبارت دیگر معماری نرم افزار انتزاعی از یک سیستم پیچیده است. مزایای این انتزاع مانند:

۱. معماری نرم افزار می تواند مبنایی برای تجزیه و تحلیل رفتار سیستم های نرم افزاری باشد.

۲. می تواند هزینه های طراحی را با فراهم آوردن زمینه ای برای استفاده مجدد از عناصر (معماری کامل نرم افزار یا قسمت هایی از آن) که ذینفعان به ویژگی ها یا عملکردهای مشابه نیاز دارند ، کاهش دهد.

۳. تصمیمات اولیه طراحی می تواند گرفته شود که به چرخه عمر توسعه نرم افزار (توسعه ، استقرار و نگهداری) کمک می کند.

۳-۱ - ۱ رابطه وابستگی

یک رابطه وابستگی می تواند بین عناصر یک سیستم اعمال شود تا نشان دهد که تغییر در یک عنصر در صورت وجود وابستگی ممکن است منجر به تغییر عناصر دیگر شود.

رابطه وابستگی را می توان به عنوان یک شبکه پیچیده در نظر گرفت. این نمای شبکه پیچیده در زمینه های متعددی با موفقیت به کار گرفته شده است. به عنوان مثال از قشر مغزی انسان (یک سیستم پیچیده) در مقاله استفاده شده است [۵]. این کار تفاوت های محلی و جهانی بین بیماران بیمار و گروه کنترل را با ارزیابی اندازه گیری قابلیت انتقال شبکه های وزنی گزارش کرد.

۴-۱ - ۱ معیار های نرم افزار

معیارهای نرم فزار از یک دیدگاه به دو دسته ی معیارهای فرآیند و معیارهای محصول تقسیم میشوند. معیارهای فرآیند به عنوان معیارهای مدیریت شناخته شده اند و برای اندازه گیری مورد استفاده قرار میگیرند و از خواص فرایند برای به دست آوردن اندازه نرم افزار استفاده میشود. معیارهای فرآیند شامل معیارهای هزینه ، معیارهای تلاش، معیارهای پیشرفت و معیارهای استفاده مجدد است . معیارهای فرآیند در پیش بینی اندازه سیستم نهایی و تعیین یک پروژه با توجه به برنامه در حال اجرا کمک میکند.

معیارهای محصول نیز به عنوان معیارهای با کیفیت شناخته شده اند و برای اندازه گیری خواص یک نرم افزار استفاده می شوند. معیارهای محصول شامل محصول غیر قابل اطمینان، معیارهای عملکرد، معیارهای قابل استفاده، معیارهای هزینه ، معیارهای اندازه، معیارهای پیچیدگی و معیارهای سبک است . معیارهای محصول در بهبود کیفیت بخش های مختلف سیستم و مقایسه میان سیستم های موجود کمک میکند.

به طور معمول استفاده از معیارهای نرم افزار با اهداف زیر صورت میگیرد:

- مطالعه تطبیقی روش های مختلف طراحی سیستم های نرم افزاری.
- برای تجزیه و تحلیل ، مقایسه و مطالعه زبان های برنامه نویسی مختلف با توجه به ویژگی های آنها.
- مقایسه و ارزیابی توانایی ها و بهره وری از افراد درگیر در توسعه نرم افزار.
- تهیه مشخصات از نرم افزار با کیفیت .
- تأیید انطباق نرم افزار مورد نیاز سیستم ها و مشخصات.
- ساخت استنتاج در مورد تلاش برای طراحی و توسعه سیستم های نرم افزاری.
- گرفتن یک ایده در مورد پیچیدگی کد.

- گرفتن تصمیمات مربوط به تقسیم بیشتر ماژول پیچیده که باید انجام شود یا نه .
- ارائه راهنمایی به مدیر منابع برای بهره برداری مناسب خود.
- مقایسه و مبادلات بین طراحی و توسعه نرم افزار و هزینه های تعمیر و نگهداری.
- ارائه بازخورد به مدیران نرم افزار در مورد پیشرفت و کیفیت در مراحل مختلف چرخه عمر توسعه نرم افزار.
- تخصیص منابع تست ، برای تست کد.

از طرف دیگر معیارهای نرم افزار محدودیت هایی هم دارند. از آن جمله میتوان به موارد زیر اشاره نمود:

- استفاده از متریک های نرم افزاری همیشه آسان نیست و در برخی موارد دشوار و پر هزینه است .
- تأیید و توجیه معیارهای نرم افزار بر اساس داده های تاریخی.تجربی که اعتبار آن به سختی قابل بررسی است .
- برای مدیریت محصولات نرم افزاری مفید هستند ولی برای ارزیابی عملکرد کارکنان فنی کارا نیستند.
- تعریف و استخراج معیارهای نرم افزار دقیق نیست و به طور کلی فرض بر این است که معیارهای نرم افزار استاندارد نیستند و به ابزارهای موجود و محیط کار بستگی دارد.
- اکثر مدل های پیش بینی در برآورد متغیرهای خاصی که اغلب شناخته شده اند تکیه می کنند.
- بسیاری ازمدل های توسعه نرم افزارها احتمالی وتجربی هستند.

۵-۱ - ۱ معیارهای اندازه نرم افزار

- تعداد خط کد

تعداد خط کد یکی از اولین و ساده ترین متریک ها برای محاسبه اندازه برنامه های کامپیوتری است . به طور کلی از این معیار برای محاسبه و مقایسه بهره وری از برنامه نویسان استفاده می شود. هر خط از متن برنامه بدون در نظر گرفتن توضیحات و یا خط خالی یک خط کد در نظر گرفته میشود.

- تعداد نشانه

در این معیارها، یک برنامه کامپیوتری به عنوان یک مجموعه ای از نشانه ها در نظر گرفته شده است که ممکن است براساس اپراتور یا عملوندها طبقه بندی شده باشد. همه معیارهای نرم افزار را می توان درقالب نمادهای اساسی تعریف کرد. این نمادها به عنوان نشانه نامیده می شود.

مقیاس های اساسی عبارتند از:

n۱= تعداد دفعات مشاهده اپراتورهای منحصر به فرد

۲n= تعداد دفعات مشاهده شده از عملوند های منحصر به فرد.

۱N= تعداد دفعات مشاهده کل وقوع اپراتورها.

۲N= تعداد مجموع تکرار عملوند.

- تعداد عملکرد

اندازه بزرگ محصول نرم افزاری را می توان از راه بهتری و از طریق یک واحد بزرگتر به نام ماژول تخمین زد. ماژول را می تـوان به عنوان بخشی از کد که ممکن است به طور مستقل وارد شده تعریف کرد.

به عنوان مثال، اجازه دهید یک محصول نرم افزاری به N ماژول احتیاج داشته باشد. به طور کلی توافق شده که انـدازه مـاژول بایـد حدود ۵۰-۶۰ خط از کد باشد. بنابراین تخمین اندازه این محصول نرم افزاری حدود ۶۰×N خط از کد میباشد.

۶-۱ - ۱ معیارهای علمی نرم افزار

مدل Halstead نیز به عنوان نظریه علم نرم افزار شناخته شده است . بر اساس این فرضیه ساخت و ساز و برنامه شامل یـک فراینـد ذهنی دستکاری اپراتورهای منحصر به فرد (۱n)و عملوند منحصر به فـرد(۲n) میباشـد. ایـن بـدان معنـی اسـت کـه یـک برنامـه از اپراتورهای ۱N و عملوندهای ۲N با انتخاب اپراتورهای منحصر به فرد از ۱n و عملوند های منحصر به فرد ۲n ساخته شده اند. بـا استفاده از این مدل، Halstead از تعدادی از معادلات مربوط به برنامه نویسی مانند سطح برنامه ، تلاش زبان پیاده سازی و غیـره به دست آمده است . ویژگی مهم و جالب این مدل آن است که برنامه را می توان از ویژگی های مختلف مانند اندازه، تلاش و غیره تجزیه و تحلیل کرد.

واژگان برنامه اینگونه تعریف شده اند :

و طول واقعی برنامه ها به عنوان: یکی از این فرایض این نظریه این است که طول یک برنامه به خوبی ساختار یک تابع فقط ۱nو ۲n است .

این رابطه به عنوان معادله پیش بینی طول شناخته شده و تعریف شده است مانند:

طول پیشنهادی برآورد برخی دیگر محققان: برآورد طول برنامه جنسن

این توسط جنسن و وایرون برای زمان واقعی برنامه های کاربردی نوشته شده توسط پاسکال بکار گرفته و معتبر شـده اسـت وحتـی نتایجی دقیق تر از برآورد هالستید کسب کرد.

برآورد طول برنامه جیف [Nz] :

که در آن n به عنوان واژگان برنامه درنظر گرفته شده است .

برآورد طول برنامه بیل مش [Nb] :

= تعدادی ازاپراتورهای منحصربه فرد که شامل عملگرهای اساسی، کلمات وتوابع .روشها.

= تعداد عملوندهای منحصر به فرد.

حجم برنامه (V)

برنامه نویسی لغت منجر به اقدامات یکی دیگر از اندازه ها که ممکن است به عنوان تعریف شود.

حجم پتانسیل (*V)

ممکن است به عنوان تعریف شود که در آن ۱n به معنی حداقل تعداد اپراتورها و ۲n حداقل تعداد عملوند ها است .

۷-۱ - ۱ معیارهای کنترل جریان

- معیار McCabe's Cyclomatic

McCabe برنامه های کامپیوتری را به عنوان یک مجموعه ی قوی متصل به گراف جهت دار تفسیر می کند. گره ها نشان دهنده بخش هایی از کد منبع هستند که هیچ شعبه و کمان نماینده که ممکن است در حین کنترل جریان اجرای برنامه انتقال یابند، ندارند.

مفهوم گراف برنامه برای اندازه گیری استفاده می شود و آن را برای اندازه گیری و کنترل تعدادی از مسیرها از طریق یک برنامه استفاده میکنند. پیچیدگی یک برنامه کامپیوتری را می توان با پیچیدگی توپولوژیک یک گراف ارتباط داد.

McCabe تعداد cyclomatic را پیشنهاد داد، G (V) نظریه گراف به عنوان یک شاخص پیچیدگی نرم افزار ارائه شده است.

تعداد cyclomatic برابر با تعداد مسیرهای خطی مستقل از طریق یک برنامه در نمودار آن است. برای کنترل برنامه ، گرافG، تعدادcyclomatic،G (V) اینگونه در نظر گرفته شده اند.

۲-۱ مشارکت ما

یک نرم افزار ممکن است به دلایل مختلف حاوی اسناد فنی کافی نباشد. بنابراین ممکن است تشخیص و بازیابی اجزایی که می توانند در پروژه های نرم افزاری دیگر مورد استفاده مجدد قرار گیرند ، دشوار باشد. هدف ما بازیابی گروهی از عناصر طراحی "مشابه" از مخزن نرم افزار به "سطوح" است که بر اساس "اهمیت" آنها طبقه بندی شده است.

مشارکت ما برای رسیدن به این هدف در زیر ذکر شده است.

۱. تجزیه و تحلیل ساختار وابستگی نرم افزار با ثبت ارتباط تماس گیرنده بین عملکردها با استفاده از ابزار "درک" [۲۱].

۲. رتبه بندی عناصر طراحی (توابع) با استفاده از مفهوم "اهمیت" آن عناصر طراحی [۱۳].

۳. استفاده از یک متریک تشابه مناسب (شباهت کسینوس) برای تقسیم بندی عناصر طراحی (توابع) بین گروه ها یا خوشه ها.

۴. گروه بندی عناصر طراحی به "ردیف" بر اساس "اهمیت" آنها.

۵. آزمایشات عددی برای نشان دادن نتایج پیاده سازی ما.

۱-۳ پیچیدگی در نرم افزار

بدلیل تفاوت ذاتی بین نرم افزار و سخت افزار پیچیدگی خاصی در ابعاد مختلف از جمله تعریف نرم افزار، طراحی و پیاده‌سازی، تست و نگهداری آن وجود دارد که: با پیچیدگی سیستم‌های طبیعی و محصولات فیزیکی ساخت بشر متفاوت است.

یک خاصیت ذاتی سیستم های نرم افزاری بزرگ است که نمی‌توان این پیچیدگی را از بین برد بلکه باید آنرا کنترل نمود.

انواع پیچیدگی:

intelleictually intractivility (تمردپذیری و اجازه پذیرفتن برای آشفتگی): پیچیدگی بطور ذاتی در ساخت سیستم وجود دارد، پیچیدگی ممکن است از بزرگی سیستم ، یا از واسینگیها، بدعت‌ها و پیاده‌سازی تکنولوژی و . . . بوجود آید.

Management intractivility (تمرد پذیری مدیریتی): پیچیدگی در سازمان و فرآیند بکار گرفته شده در ساخت سیستم، ممکن است از اندازه پروژه (تعداد افردی که در تمام جهات ساخت سیستم درگیر هستند)، وابستگی های پروژه، فاصله جغرافیایی سیستم ها و . . . به عبارتی عوامل تولید کننده نرم افزار غیر قابل کنترل هستند چون سازمان، افراد و فرآیند هستند و ماشین نیستند که کنترل شوند و سرمایه‌های اولیه برای تولید نرم افزار الزاماً ماشین، سرمایه و پول نیست بلکه یکسری عوامل انسانی متغیری هستند که تحت مدیریت قرار می‌گیرند.

۱-۴ سازمان پایان نامه

در مجموع ۵ فصل در این پایان نامه وجود دارد. فصل ۱ فصل مقدماتی است که در آن مشکل و اهمیت حل مسئله را به طور کلی معرفی می کنیم. سپس تعاریف و شرح نرم افزارهای محاسبات علمی و معماری آنها و همچنین مفاهیم رابطه وابستگی را ارائه می دهیم. ما همچنین در مورد مشارکت و پایان نامه خود در این فصل بحث می کنیم.

در فصل ۲ ، ما در مورد استخراج و مدل سازی وابستگی بحث می کنیم. ما برخی از ابزارهایی را که برای استخراج و تجسم نمودارهای تماس استفاده می شود ، توصیف می کنیم. در پایان این فصل ، ما به ابزاری که در پایان نامه ما استفاده شد اشاره می کنیم.

شرح مفصل در مورد مرکزیت اجزا با استفاده از یک روش طیفی در فصل ۳ ارائه شده است. قبل از توضیح این روش ها با چند مثال کوچک ، ما مقادیر ویژه ، بردارهای ویژه ، هاب و قدرت را شرح می دهیم.

فصل ۴ شامل یک بحث کوتاه در مورد روش رویکرد جدید ما برای تجزیه و تحلیل نرم افزار علمی و به دنبال آن شرح مفصل سیستم های مورد نظر است. این سیستم ها شامل CSparse و ADOL-C می باشد. سپس در مورد اجرای الگوریتم بحث می کنیم. در نهایت ما نتایج خود را از آزمایشات گزارش و بحث می کنیم. ما نکات پایانی و دستورالعمل های آتی کار را در فصل ۵ ارائه می دهیم.

فصـــل دوم

استخراج و مدل سازی وابستگی

۱ – ۲ طراحی ساختار ماتریس

طراحی ساختار ماتریس (DSM) یک نمایش ساده فشرده و بصری از یک سیستم یا پروژه در قالب یک ماتریس مربع است [۷]. DSM یک ابزار مدل سازی شبکه است که برای نمایش عناصر متشکل از یک سیستم و برهم کنش های آنها استفاده می شود. با بررسی وابستگی هایی که بین عناصر آن در یک ماتریس مربع وجود دارد ، معماری سیستم ها یا روابط بین عناصر یک سیستم را برجسته می کند.

برای تجزیه و تحلیل یک سیستم ، مدل های DSM را می توان با استفاده از روش های تحلیلی مختلف مانند خوشه بندی و تعیین توالی مجدداً مرتب یا تقسیم بندی کرد.

DSM مزایای زیر را ارائه می دهد:

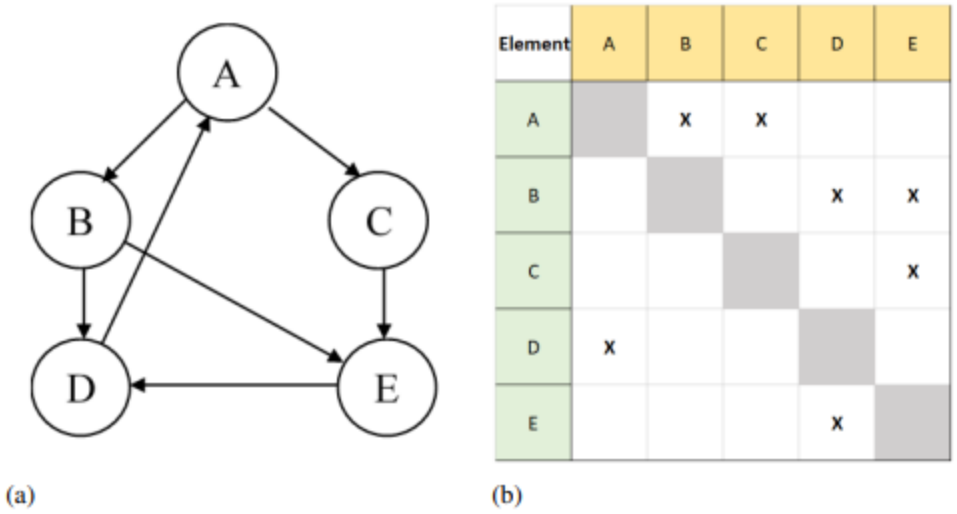
• DSM یک قالب نمایشی فشرده برای سیستم های پیچیده بزرگ ارائه می دهد.

• DSM نمای سطح سیستم را برای طراح سیستم برجسته می کند که از تصمیم گیری بهینه در سطح جهانی پشتیبانی می کند.

• ساختار اساسی یک سیستم پیچیده به دلیل DSM قابل درک می شود.

• DSM با استفاده از ماتریس مربع نمایش داده می شود. بنابراین تعدادی از تجزیه و تحلیل های قدرتمند در نظریه نمودارها و ریاضیات ماتریسی و همچنین روش های تجزیه و تحلیل تخصصی DSM برای DSM کاربرد دارد.

ما برای نشان دادن روابط عنصر از یک مثال ساده استفاده می کنیم (شکل ۲,۱ را ببینید). توجه داشته باشید که سیستم متشکل از پنج عنصر (یا زیر سیستم) است: " "A، "B"، "C"، D"" و "E". ما فرض می کنیم که پنج عنصر به طور کامل سیستم را توصیف می کنند و رفتار آن را مشخص می کنند در حالی که از DSM برای مدل سازی استفاده می کنیم. برای نمایش تصویری این سیستم از فرم گرافیکی استفاده می کنیم. نمودار سیستم با اجازه دادن به یک راس/گره روی نمودار برای نشان دادن یک عنصر سیستم و یک لبه متصل به دو گره برای نشان دادن رابطه بین دو عنصر ساخته می شود. جهت گیری تأثیر از یک عنصر به عنصر دیگر توسط یک پیکان به جای یک پیوند ساده ثبت می شود. به عنوان مثال ما می توانیم یک پیکان از عنصر A به عنصر B. مشاهده کنیم اگر این عناصر به عنوان تابع در نظر گرفته شوند ، می توانیم بگوییم که تابع A تابع B. را فرا می خواند. بنابراین تابع A تماس فرستنده و تابع B تماس گیرنده است. نمودار حاصله گراف جهت دار یا صرفاً دیگراف نامیده می شود (در شکل ۲,۱ الف نشان داده شده است).



شکل ۲,۱: نمونه ای از DSM: الف) نمودار وابستگی ، ب) ماتریس وابستگی

ما می توانیم DSM را با استفاده از فرم ماتریس نشان دهیم. بازنمایی ماتریس یک گراف جهت دار دارای برخی از خصوصیات است مانند دوتایی (بدون وزن) یا می توان آن را به صورت مربع وزن کرد (دارای n سطر و ستون است که n تعداد گره های دیگراف است) k غیر صفر دارد عناصر (k تعداد لبه های دیگراف است).

نام عناصر به عنوان عناوین سطر در پایین ماتریس و در بالای آن به عنوان ستون به همان ترتیب قرار می گیرند (شکل ۲,۱ ب را ببینید). اگر یک لبه (رابطه) از گره x به گره y وجود داشته باشد ، مقدار عنصر [x] [y] با X مشخص می شود. در غیر این صورت مقدار خالی می ماند. عناصر مورب ماتریس تفسیری ندارند اما در برخی موارد به عنوان نماینده خود گره ها در نظر گرفته می شوند. برای ماتریس های دوتایی X 1 است و برای ماتریس های دیگر X به معنی مقدار عددی است.

۲ – ۲ استخراج وابستگی

بررسی وابستگی های برنامه مانند فراخوانی عملکردها برای سیستم های بسیار بزرگ چالش برانگیز است [۲۲]. دو طبقه اصلی برای استخراج نمودارهای فراخوانی وجود دارد: سبک وزن و سنگین وزن. بخشی از کل اطلاعات استاتیک با استخراج سبک وزن ارائه می شود ، از طرف دیگر استخراج کننده های سنگین یک نمودار فراخوانی کامل ارائه می دهند. استخراج کننده های سنگین بار دیگر می توانند در دو نوع سختگیرانه و تحمل پذیر طبقه بندی شوند. مانند کامپایلرها ، وقتی یک خطای واژگانی یا نحوی وجود داشته باشد ، استخراج کننده سنگین وزن متوقف می شود. استخراج کننده مدارا نمودارهای تماس کامل را ارائه می دهد.

Understand [۲۱] ابزاری است که می تواند برای استخراج وابستگی استفاده شود. در این پایان نامه از این ابزار استفاده کرده ایم. درک برای تجزیه و تحلیل وابستگی بین مصنوعات نرم افزاری در یک پروژه استفاده می شود. این برنامه از طیف گسترده ای از زبان های برنامه نویسی از جمله جاوا ، کلاس های C#، C++ و بسته های Ada برای اطلاعات وابستگی پشتیبانی می کند و می تواند از API های C++ و PERL به اطلاعات وابستگی دسترسی داشته باشد. با استفاده از این نرم افزار می توانیم کدها را با استفاده از یک ارجاع دقیق متصل کرده و آنها را با استفاده از نمایش های گرافیکی تجسم کنیم. تجزیه و تحلیل هر کد منبع به معنی تجزیه و تحلیل واحدهای مختلف آن است زیرا این واحدها دارای ویژگی های متمایزی هستند که بر روی کد منبع منعکس می شود. Understand دارای ویژگی های معماری است که به ما کمک می کند مجموعه های سلسله مراتبی واحدهای کد منبع را ایجاد کنیم. وابستگی بین این واحدها را می توان از مرورگر وابستگی مشاهده کرد و توسط نمودارهای وابستگی تجسم کرد. با این حال ، ما می توانیم پارامترهای مختلفی مانند ، گره ها ، فایل ها ، کلاس ها ، بسته ها و رابط ها را برای مشاهده وابستگی بین واحدهای مختلف کد منبع با استفاده از مرورگر وابستگی مشاهده کنیم. این ابزار دارای ویژگی های زیر برای مشاهده وابستگی ها است [۲۱]:

• مرور سریع وابستگی ها برای فایل ها و درک معماری ها

• فهرست موجودیت های "وابسته" و "وابسته به" پرونده ها و معماری ها

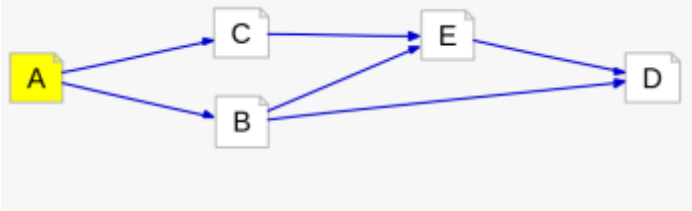
• خروجی صفحه گسترده روابط وابستگی

• داک Dependency Browsing که همه اطلاعات وابستگی را نشان می دهد

برای توضیح مختصر وابستگی هایی که می توان با استفاده از Understand استخراج کرد ، می توانیم یک کد منبع نمایشی را که در شکل ۲,۵ نشان داده شده است در نظر بگیریم. اکنون می توانیم این مثال را برای نشان دادن وابستگی های مختلف نشان دهیم:

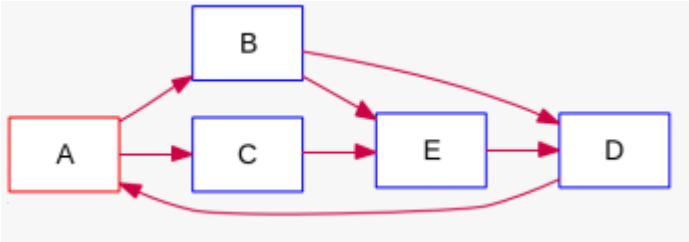
• شامل وابستگی: نمودار Include Dependency فایل هایی را نشان می دهد که برای پیاده سازی مورد نیاز است. برای مثال یک فایل "xyz.c" به فایل دیگری وابسته است "xyz.h" بدین معناست که "xyz.c" فایل "xyz.h" را در بر گرفته است. این وابستگی را می توان با استفاده از Understand پیدا کرد.

• وابستگی به تماس: با استفاده از درک شکل ۲,۲ رابطه تماس ها از A تا D. را نشان می دهد. ما می توانیم بررسی کنیم که آیا بین دو عنصر سیستم رابطه وجود دارد یا خیر. سپس با استفاده از ابزار Understand می توانیم شکل ۲,۳ را نیز مشاهده کنیم که در آن فایل ها/ عملکردهای با لبه های خروجی به فایل ها/ توابع با لبه های ورودی وابسته هستند. به عنوان مثال با مشاهده شکل ۲,۳ می توان گفت که تابع A وابسته به توابع B و C است ، زیرا از A تا B و C حاشیه ای وجود دارد. جزئیات این فراخوانی ها یا وابستگی ها را می توان در مرورگر اطلاعات یافت.



شکل ۲,۲: رابطه تماس ها

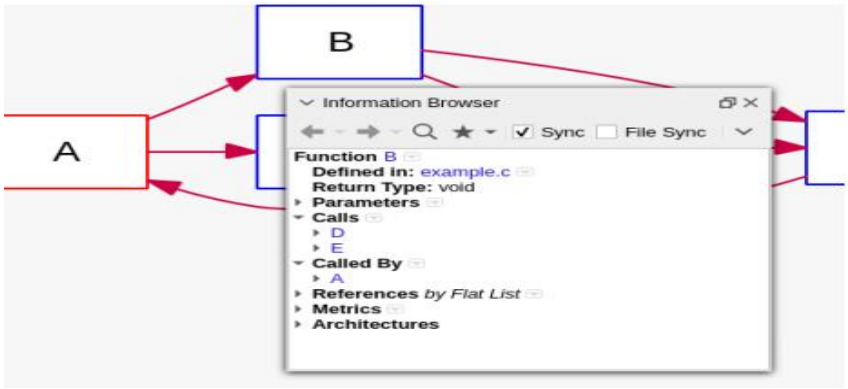
• وابستگی اولیه: وابستگی init بر مقداردهی اولیه یک شی تمرکز می کند.



شکل ۲,۳: وابستگی تابع

• تنظیم وابستگی: با استفاده از Understand می توانیم وابستگی های مجموعه را نیز بیابیم. فرض کنید یک تابع از یک فایل مقدار یک شی را از فایل های مختلف تعیین می کند. سپس می گوییم یک وابستگی بین این دو فایل وجود دارد.

• از وابستگی استفاده می کند: نمودار وابستگی که کاربردهای مختلف بین دو فایل را نشان می دهد، به عنوان مثال با مشاهده نمودار وابستگی استفاده می توان گفت که چند بار یک فایل/ تابع از فایل/ تابع دیگری استفاده می کند. شکل ۲,۴ برخی از اطلاعات استفاده از تابع B. را نشان می دهد. در اینجا ما می توانیم B توابع D و E را فراخوانی کنیم و توسط تابع A. فراخوانی شده اند. از نمای گرافیکی می توانیم ببینیم چند بار دیگر توابع را فرا می خواند.



شکل ۲,۴: شرح عملکرد

File Edit View Search Tools Documents

```
#include<stdio.h>
int x=0;

void E()
{
  printf("E");
  D();
}
void D()
{
  printf("D");
  A();
}
void C()
{
  printf("C");
  E();
}
void B()
{
  printf("B");
  D();
  E();
}
void A()
{
  if (x!=0) retun;
  x++;
  printf("A");
  B();
  C();
}
```

شکل ۲٫۵: یک مثال شبه برای وابستگی به تابع

۳ – ۲ روابط ، ماتریس ها و نمودارها

تجزیه و تحلیل شبکه های پیچیده بزرگ در تجزیه و تحلیل شبکه های اجتماعی رایج است و از این رو نمایش آن ها دغدغه اصلی محققان شده است. علاوه بر این یک نرم افزار علمی همچنین دارای اطلاعات زوجی بین واحدهای خود (ماژول ها) است که نیاز به نمایش زوجی دارد.

زمینه تجزیه و تحلیل اطلاعات جفتی از سه ساختار ریاضی بسیار مرتبط برای نشان دادن آنها استفاده می کند: روابط ، نمودارها و ماتریس ها.

۱ – ۳-۲ روابط

رابطه دودویی R را می توان مجموعه ای از جفت های مرتب (x، y) تعریف کرد. برای بیشتر روابط مفید ، عناصر جفت های مرتب شده به طور طبیعی به نوعی مرتبط یا مرتبط هستند. این رابطه (جفت های مرتب شده) دو مجموعه را به هم مرتبط می کند و شامل یک نقشه برداری است.

به عنوان مثال ، یک رابطه در یک تابع نیز یافت می شود. در اینجا y = f (x) = 2x تابعی از همه اعداد زوج است. نماد معادله فقط یک دست کوتاه برای شمارش همه جفت های ممکن در رابطه مانند ، {(۱،۲) ، (۲،۴) ، (۳،۶) ، ...} است.

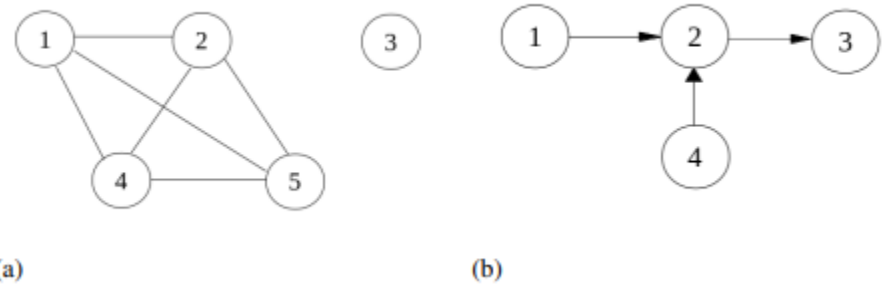
مثال نشان داده شده در شکل ۲،۵ را می توان به صورت R = {(A,B) ، (A، C)، (B، D)، (B، E)، (C، E)، (A، D)، (D، E)}.

۲ – ۳-۲ نمودارها

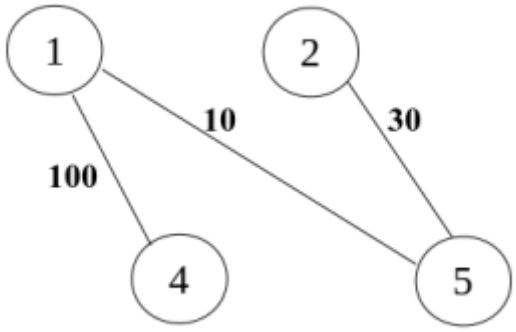
اجازه دهید G (V,E)، یک گراف باشد که V مجموعه محدود رأس ها و E مجموعه ای از لبه ها است که رابطه زوجی بین رئوس را در V نشان می دهد. چند دسته از نمودارها وجود دارد و ما آنها را با توجه به نیاز خود برای تجزیه و تحلیل استفاده می کنیم.

• نمودارهای جهت دار و غیرمستقیم: نمودارهای جهت دار (که دیگراف نیز نامیده می شوند) گرافی است که از مجموعه ای از رأس هایی که توسط لبه هایی به هم متصل شده اند تشکیل شده است که در آن لبه ها جهت مربوط به خود را دارند. این نمودارها از جفت مرتب شده تشکیل شده اند. ما می توانیم از آنها برای نشان دادن روابط غیر متقارن مانند نمودارهای تماس استفاده کنیم. از طرف دیگر یک گراف غیرمستقیم از جفت های بی نظم تشکیل شده است که در آن همه لبه ها دو طرفه هستند. آنها برای روابطی که لزوماً متقارن هستند استفاده می شوند. شکل ۲،۶ (الف) نمونه ای از یک گراف غیر مستقیم و شکل ۲،۶ (ب) نمونه ای از یک گراف جهت دار است.

• نمودارهای ارزشمند و بدون ارزش: در نمودارهای دارای ارزش ، لبه ها دارای مقادیری هستند که مشخصات روابط را نشان می دهد ، مانند قدرت ، مدت زمان ، ظرفیت ، جریان و غیره. تماس گرفت. نمودارهای بدون ارزش هیچ ارزشی برای لبه ها بیان نمی کنند. شکل ۲،۷ نمونه ای از نمودار ارزشمند است که در آن هر لبه مقداری روی خود دارد. از طرف دیگر شکل ۲،۶ را می توان به عنوان نمونه ای از دو نمودار بدون ارزش در نظر گرفت.



شکل ۲,۶: نمونه ای از نمودار: الف) گراف غیرمستقیم ، ب) نمودار جهت دار



شکل ۲,۷: نمونه ای از نمودار با ارزش

• نمودارهای بازتابی و غیر بازتابی: نمودارهای بازتابی به حلقه های خود اجازه می دهند. این یک راس است که می تواند یک لبه برای خود داشته باشد. به عنوان مثال اگر یک تابع خود را فراخوانی کند (تابع بازگشتی) ، یک لبه از آن راس به خود (حلقه خود) وجود خواهد داشت.

• چند گراف: اگر بین دو راس بیش از یک لبه وجود داشته باشد ، آن نمودار چند گراف نامیده می شود. اما به جای استفاده از چند گراف ترجیح می دهیم از نمودارهای با ارزش استفاده کنیم. به عنوان مثال اگر تابع A دو بار تابع B را فراخوانی کند ، برچسب ۲ را در لبه بین A و B قرار می دهیم.

اکنون ما نیاز به برخی از تعاریف اولیه داریم.

درجه یک راس تعداد راس هایی است که در مجاورت آن راس قرار دارند. با این حال ، تعداد لبه هایی است که در آن راس قرار می گیرند. به عنوان مثال در شکل ۲,۶ (a)، Vertex 4 دارای درجه ۳ است. یک راس صفر درجه ایزوله نامیده می شود Vertex 3 در شکل ۲,۶ (a) یک راس جدا شده است. رأسی با درجه ۱ آویز نامیده می شود. رأس ۱ در شکل ۲,۶ (b) چنین رأسی است.

در یک دیگراف (شکل ۲,۶ (b) را ببینید) درجه نامساوی یک قوس تعداد قوس (یا لبه ها) است که از سایرین به آن راس وارد می شود در حالی که درجه بالاتر تعداد قوس از آن راس به بقیه است. در شکل ۲,۶ (b)، Vertex 2 دارای درجه ۲ درجه بالاتر و درجه ۱ است.

اگر مسیری از هر راس به هر راس دیگر وجود داشته باشد ، نمودار متصل می شود. حداکثر زیرگراف متصل را جزء می نامند. نمودار نشان داده شده در شکل ۲,۶ (الف) دارای دو جزء است: {۱،۲،۴،۵} و {۳}. حداکثر زیرگراف یک زیرگراف است که برخی از ویژگی های مشخص شده (مانند اتصال) را برآورده می کند و بدون نقض ویژگی نمی توان به آن راس اضافه کرد.

۳ – ۳-۲ماتریس ها

وابستگی های نشان داده شده توسط یک نمودار را می توان با ماتریسی با ابعاد مناسب نشان داد. دلیل استفاده از دو روش مختلف نمودار و ماتریس برای نمایش اطلاعات یکسان این است که یک مبادله وجود دارد. نمودارها نسبت به ماتریس ها بصری تر هستند ، اما وقتی تعداد گره ها و لبه ها افزایش می یابد ، درک آنها دشوار است. چند ده گره می تواند برای تولید یک نمودار بسیار پیچیده کافی باشد. از طرف دیگر نمودارهای بزرگ و پیچیده را می توان با یک ماتریس بسیار کارآمد نشان داد.

$$X = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad (2.1)$$

در شکل ۲,۸ ما DSM را با توجه به مثال نشان داده شده در شکل ۲,۵ نشان می دهیم. در اینجا ردیف ها مربوط به توابع تماس فرستنده و ستون ها مربوط به توابع تماس گیرنده است.

	A	B	C	D	E
A		1	1	0	0
B	0		0	1	1
C	0	0		0	1
D	1	0	0		0
E	0	0	0	1	

شکل ۲,۸: نمونه ای از DSM

نمایش ماتریس معادل در رابطه ۲,۱ ارائه شده است. ورودی $X_{ij} = 1$ به این معنی است که تابع در Row_i تابع را در $Column_j$ فراخوانی می کند.

اگر می خواهیم رابطه در جهت دیگری باشد ، می توانیم ماتریس X را به جابجا کرده و X^T را بدست آوریم (به رابطه ۲,۲ مراجعه کنید). وقتی X متقارن است $X^T = X$ داریم.

$$X^{\top} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad (2.2)$$

در ابتدای این فصل ما در مورد DSM بحث کرده ایم. DSM یک ماتریس مربعی است که برای نشان دادن دقیق اطلاعات مشابه در نمودار یا ماتریس مجاورت استفاده می شود. در این پایان نامه ما از دوگانگی یک نمودار و ماتریس برای نمایش و محاسبه موثر اطلاعات کمی در مورد معماری یک سیستم نرم افزاری با استفاده از نظریه نمودار و جبر خطی استفاده می کنیم.

فصـــل ســـوم

تجزیه و تحلیل وابستگی

زمانی که درمورد پیچیدگی نرم افزار بحث می کنیم باید توجه داشته باشیم که پیچیدگی مفهوم ثابتی نیست و باید در شرایط مختلف مورد بررسی قرار گیرد.از این رو می گوییم پیچیدیگی نرم افزار اصطلاح غیر استانداری است زیرا از محدوده مشخصی برخوردار نیست، امکان دارد نرم افزار بزرگی با خطوط زیادی از کد و پیمانه های مختلف و مرتبط را پیچیده و همینطور نرم افزار کوچکی با الگوریتمی سخت را نیز پیچیده ارزیابی نماییم .برای همین است که نمی توان پیچیدگی را از یک بعد مورد بررسی قرار داد. موسسه IEEEتعریفی از پیچیدگی ارائه داده است که به شرح زیر می باشد" :میزان سختی در درک یک سیستم و یا مؤلفه که طراحی و پیاده سازی شده است ".از این رو باید عنوان کرد که پیچیدگی نرم افزار به فاکتورهایی از اندازه گیری تاثیر گذار در هزینه ، نگهداری و توسعه اطلاق می شود.البته افرادی چون "زئوس" این نظریه را زیر سوال برده و پیچیدگی را از دید روانشناسی نرم افزار مورد بررسی قرار داده و دسته بندی نموده اند Zeus پیچیدگی روانشناسی نرم افزار را به سه بخش زیر تقسیم نموده :

۱-پیچیدگی در مساله

۲-پیچیدگی در طراحی سیستم

۳-پیچیدگی رویه ای

مثلا" درمورد پیچیدگی در مساله باید عنوان داشت که این امر به پیچیدگی کلی الگوریتم مساله بر میگرد.یعنی ممکن است یک سیستم پیچیدگی ذاتی کم یا بیشتری داشته باشد که در میزان پیچیدگی کلی آن تاثیرگذار خواهد بود.همینطور می توان به پیچیدگی موجود در طراحی سیستم اشاره داشت که به دو قسمت: ساختاری و داده ای تقسیم می شود .پیچیدگی ساختاری به مفهوم کلی اتصال اشاره دارد و اتصال بین پیمانه های درونی در زبان برنامه نویسی را مورد بررسی قرار می دهد. به عنوان مثال ؛ در یک زبان برنامه نویسی (فرض کنید) C ++ ارتباطاتی بین پیمانه های درونی زبان + کلاسها و کتابخانه ها وجود دارد که به خودی خود پیچیدگی هایی را به دنبال خواهد داشت .پیچیدگی داده ای به مفهوم دیگری با عنوان "چسبندگی" اشاره دارد و چسبندگی های درون ساختاری را مورد بررسی قرار می دهد .پیچیدگی ساختاری و داده ای ارتباط مبتنی بر IN-FAN و OUT-FAN در پیمانه ها و متغییرهای خروجی/ورودی را مورد بررسی قرار می دهند .آنگونه که اشاره شد پیچیدگی های ساختاری و داده ای به روابط بین پیمانه های درون سیستمی نرم افزار اشاره دارند حال آنکه در زبان های پیش گام و با به کارگیری OOP یا برنامه نویسی شیء گرا از میزان این پیچیدگی ها تا حد زیادی کاسته شده است.

باید اشاره داشت که اغلب پیچیدگی های درونی مانند آنچه گفته شد درصد پایین و ضریب اندکی از پیچیدگی نرم افزار را شامل می شوند و در شکل گیری بحران نرم افزار آنچنان تاثیر گذار نیستند.

پیچیدگی رویه ای: این پیچیدگی به ساختار کلی برنامه ، خطوط کد ، حلقه ها و شرط ها و طریقه به کار گیری آنها اشاره می کند .به صورت کلی پیچیدگی بخش غیرقابل اجتناب در تولید نرم افزار است یعنی به هر نحو میزانی از پیچیدگی در نرم افزار وجود دارد اما مشکل بحران نرم افزار زمانی رخ می دهد که این پیچیدگی ها از یک حد بیشتر شوند. وظیفه توسعه دهندگان نرم افزار این است که این پیچیدگی ها را تا حد ممکن کاهش دهد و این یعنی ارائه راهکار و رویه ای مناسب تا یک الگوریتم هر چه ساده تر و مفید تر پیاده سازی شود.

نمونه کد

در ادامه به مثال زیر توجه کنید که در آن یک روش رویه ای در کنترل پیچیدگی اعمال شده است.برنامه زیر دو عدد را مقایسه کرده و عدد بزرگتر را درون متغییری با نام Max میریزد.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int a,b,Max;
    if(a>b)
        Max=a;
    else
        Max=b;

}
```

الگوی اول : این الگو دو عدد را با استفاده از statement if یا همان دستور شرطی if برسی می کند و به پاسخ میرسد،حالا به الگوی زیر توجه کنید:

```
int _tmain(int argc, _TCHAR* argv[])
{
    int a,b,Max;
    (a>b)?Max=a:Max=b;
}
```

در مثال فوق همان برنامه اول منتهی با رویه ای مناسبتر برای جلوگیری از پیچیدگی نوشته شده است و از دستور شرطی یک خطی استفاده شده است .این مثال الگویی ساده برای پی بردن به این نکته است که در بحران نرم افزار ، کنترل پیچیدگی امکان پذیر و راهبردی است .فراموش نکنیدم که هرچه مدل ما حاوی اطلاعات بیشتری باشد، طبیعتا مدل ما پیچیده تر خواهد بود.

ماهیت پیچیدگی نرم افزار

همانطور که در بحث های ابتدایی و الزام آور بودن اصول اندازه گیری نرم افزار (measurement) مطرح شد، شناخت ماهیت نرم افزار باعث می شود از عملیات کورکورانه و به قولی آزمون و خطا جلوگیری شود. عدم رعایت این اصل(شناخت) یکی از مهمترین عوامل پیچیدگی می باشد.زمانی که برنامه ساز در مورد برنامه پیش رو شناخت کافی نداشته باشد و بر ماهیت کلی آن اشراف نداشته باشد قطعا نرم افزار تولیدی او نیز کارایی بالایی نخواهد داشت و امکان بروز اختلالات غیرمرقبه در حین کار با نرم افزار بالا می رود به گونه ای که عدم رضایت مشتریان را به دنبال خواهد داشت.

اجزای کلیدی در پیچیدگی

پیچیدگی برچسبی است که ما به وجود متغییرهای وابسته موجود در یک سیستم نسبت می دهیم،هرچه متغییرها بیشتر باشد وابستگی بین آنها بیشتر می شود. اتصالات بین متغییرها ویژگی های زیادی برای آن واحد می طلبد که منجر به بروز پیچیدگی بیشتر خواهد شد . پیچیدگی چند جزء کلیدی دارد که به شرح زیر می باشد:

۱-مقیاس :مقدار اجزاء سیستم نرم افزاری

۲- تنوع:گستره اجزاء متنوع تشکیل دهنده سیستم

۳- اتصال : ارتباط بین مؤلفه ها را فراهم می کند

مقیاس : مقیاس به تنهایی مساله نیست و در صورت زیاد بودن مقیاس با استفاده از داده های آماری می توان آن را تعیین و کنترل نمود.

تنوع : این جزء انواع اجزائی را که باید تحلیل شوند افزایش می دهد. هرچه تنوع بیشتر شود تلاش بیشتری برای درک سیستم لازم داریم.

اتصال : با افزایش مقیاس قاعدتا تعداد اتصالات و تعامل بین اجزاء بیشتر خواهد شد و این امر نیز به نحوی موجب پیچیدگی می شود.

بررسی معیارهای پیچیدگی

پیچیدگی به صورت کم و بیش وجود دارد و وجود خواهد داشت به صورت کلی پیچیدگی دارای رابطه مستقیم با گستردگی پروژه می باشد. پر واضح است که هر مقدار که تعداد خطوط کد پروژه بیشتر باشد قاعدتا" از مدل ها ، کلاس های زیادی برخوردار بوده و پیچیدگی ذاتی آن بیشتر است.حالا کافیست مطالب قبلی را به یاد بیاوریم "پیچیدگی به صورت کامل قابل از بین بردن نیست ، از این رو باید آن را کنترل نمود ".

پیچیدگی نرم افزار دارای ویژگی های انحصاری و بارزی به شرح زیر می باشد:

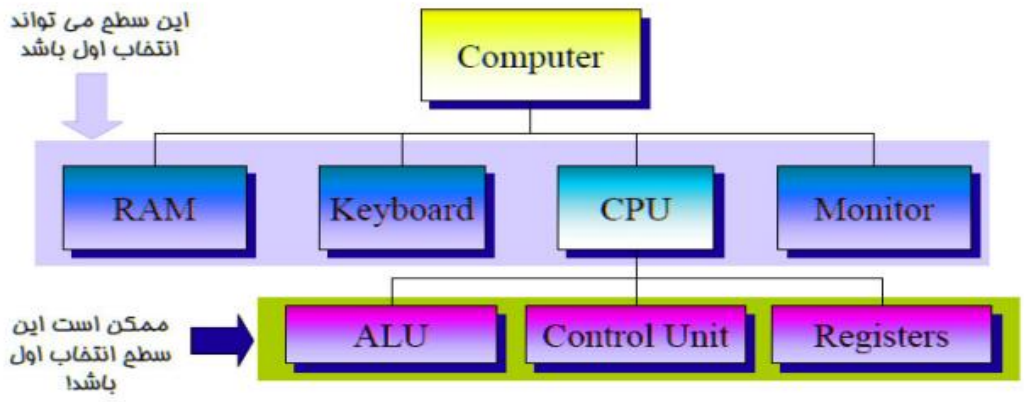
- با پیچیدگی سخت افزاری در وسایل فیزیکی و مکانیکی تفاوت دارد و از یک مفهوم منطقی در نرم افزار نشآت می گیرد
- یک خاصیت ذاتی در نرم افزارهای بزرگ می باشد
- پیچیدگی حوزه مساله
  ۱- نیازمندی های گوناگون و متضاد
  ۲- ارتباط بین کاربر و نرم افزار
  ۳- تغییر نیازها
- پیچیدگی در فرایند تولید
- عدم انعطاف پذیری سیستم و استاندارد نبودن اجزاء آن
- مشکل در توصیف رفتارهای سیستم های گسسته

پیچیدگی عارضی

پیچیدگی ذاتی عموما مشخصه هر مساله نرم افزاری و الگوریتم های مربوط به آن می باشد، یعنی اگر مساله ای ساده را نیز در نظر بگیریم شامل راه حلی می باشد ، همین راه حل هر چند ساده یک پیچیدگی ذاتی برای نرم افزار محسوب می شود. در مقابل پیچیدگی ذاتی ، پیچیدگی عارضی قرار گرفته است که از انتخاب صورت گرفته در ساخت سیستم ناشی می شود، که به دلایل بروز آن به صورت کامل اشاره شد. به صورت اصولی باید گفت عملی ترین و بهترین راه برای کنترل پیچیدگی عارضی این است که ابتدا مساله را درست تعریف کنیم سپس به دنبال راه حل(چگونگی) انجام مساله برویم .تمام مواردی که در سرفصل های قبلی پیچیدگی مطرح شد به نوع عارضی اشاره دارند و با به کارگیری راهکارهای ذکر شده تا حدود زیادی می توان در مقابل بروز اینگونه پیچیدگی ها مصون بود. البته برخی راهکارها در مقابله با بروز پیچیدگی عارضی وجود دارد که در ادامه به آن اشاره خواهیم داشت .

پیچیدگی سلسله مراتبی (Hierarchy)

در اغلب سیستم ها ، پیچیدگی به صورت سلسله مراتبی ظاهر می شود. در اینگونه سیستم ها ارتباط بین اجزای درونی سیستم قویتر از ارتباط بین خود زیر سیستم هاست و این یکی از ویژگی های پیچیدگی در ساختار سلسله مراتبی می باشد . شکل زیر نمایانگر یک سیستم پیچیده سلسله مراتبی می باشد.



سیستم های سلسله مراتبی معمولا از تعداد کمی از زیر سیستم های مشخص و متفاوت تشکیل می شوند که این زیر سیستم ها به صورت های گوناگون و ترتیب های مختلف ظاهر می شوند .

باید توجه داشت که سیستم های پیچیده ای که حتی با وجود گستردگی به خوبی عمل می کنند دارای زیر سیستم های ساده و کار آمد و هدفمند می باشند که در تعامل با سایر زیر سیستم ها بوده و از پیچیدگی کنترل شده ای برخوردار می باشند. در غیر این صورت و اگر سیستم دارای زیر سیستم های اصولی و مهندسی شده نباشد ، قطعا ماهیت کلی سیستم به خطر افتاد و نرم افزار مورد نظر کار نخواهد کرد.

شیء گرایی و نقش آن در کنترل پیچیدگی نرم افزار یکی دیگر از مواردی که موجب کاسته شدن از پیچیدگی می شود به کار گیری روش های شیء گرایی(Oriented Object) می باشد. شیء گرایی دارای ۴ اصل کلی می باشد که عبارتند از:

۱-تجرید

۲-محصور سازی

۳-واحد بندی

۴-سلسله مراتب

تجرید(Abstraction)

به صورت کلی به فرآیند تشخیص خصیصه های مهم یک پدیده گویند، در تجرید باید از پرداختن به ویژگی های موقت و غیر مهم جلوگیری کرد و خصیصه ها و ویژگی های مهم یک پدیده را مدنظر قرار دهیم. مثلا یک دانشجو را در نظر بگیرید(پدیده) ویژگی های مهم برای صدور کارنامه برای یک دانشجو شامل مواردی مانند : تعداد واحد ها ، نمرات ، شماره دانشجویی ، شماره شناسنامه می باشد که پس از اعمال اصل تجرید بدست خواهد آمد .در صورتی که ما اصل تجرید را رعایت نکنیم امکان استفاده از ویژگی

های موقت و غیرضروری مثل :شماره تلفن همراه ، وضعیت جسمانی و... را فراهم کرده ایم که این امر به خودی خود موجب بروز پیچیدگی بیشتر خواهد شد .به زبان ساده اصل تجرید مؤلفه های غیرضروری که از اهمیت کمتری برخوردار هستند را حذف و به ویژگی های مؤثر در طراحی صحیح الگوریتم و رسیدن به روشی مناسب برای حل مساله می پردازد.

محصورسازی(Encapsulation)

محصورسازی موجب می شود تا پدیده از اثرات مخرب میحط خارجی به دور بماند و همینطور طریقه دسترسی سایر اشیاء به آن پدیده را نیز کنترل نماییم. محصورسازی با کنترل راه های دسترسی به یک پدیده (شیء) باعث جلوگیری از تاثیرات منفی و احتمالی بر روی پدیده می شود و همینطور از گسترش خطاهای رخ داده در درون پدیده به سایر اشیاء جلوگیری می کند .از جمله سایر نقش های محصورسازی در کنترل پیچیدگی این است که به دلیل ثبات در واسط ارتباطی (Interface) می توان تغییرات زیادی بر روی آن اعمال کرد و همینطور امکان استفاده مجدد را به پدیده می دهد .در کپسوله سازی ، محصور سازی از دو اصطلاح ماژول باز و ماژول بسته استفاده می شود. درواقع به ماژولی که برای اعمال تغییرات آماده باشد باز و به ماژولی که امکان تغییر در آن وجود نداشته باشد ، ماژول بسته می گوییم. در عملیات Encapsulation هر دو ماژول باز و بسته کاربردی و لازم می باشند .

واحد بندی(Modularity)

یک سیستم را در نظر بگیرید که واحد های کوچکتر آن هر یک به صورت واحدهایی جداگانه دسته بندی شده باشند. به این دسته ها ماژول گفته می شود. سیستم هایی با دسته بندی (Module)و با حداقل ارتباط را سیستم های ماژولار یا واحد بندی شده گویند .به عنوان مثال از فایل ها در زبان زبان C ++ ، یونیت ها در دلفی و پاسکال ،کامپوننتها در NET. و جاوا را می توان به عنوان ماژول نام برد. واحد بندی دارای دو خصیصه اصلی می باشد:

۱ – انسجام :درجه ارتباط عملکردهای داخلی ماژول را گویند.

۲ -وابستگی : درجه ارتباط ماژول ها با یکدیگر را مشخص می کند.

۳ – استقلال یا Independent

۴- Well defined interfaces یا واسط های خوش تعریف

درواقع واحد بندی با شکستن مساله به زیر مساله ها از بروز پیچیدگی جلوگیری می کنند و موجب نظام مند شدن مساله برای کنترل پیچیدگی می شود.البته مواردی که باید در واحد بندی مورد توجه قرار بگیرند این است که

۱ - ماژول بندی نباید با تعداد خیلی زیاد و یا خیلی کم باشد

۲- معیاری مشخص برای شکستن مساله باید لحاظ شود تا ماژول بندی مؤثر واقع شود.

سلسله مراتب(Hierarchy)

به مرتب ساختن Abstract ها در سطوح مختلف سلسله مراتب گفته می شود. باید توجه داشت که نمودار سلسله مراتبی دارای دو ویژگی زیر می باشد :

ساختار کلاس : سلسله مراتبA-IS

ساختار شیء :سلسله مراتبOF-PART

در کل باید گفت فرمول شکل اصلی یک سیستم پیچیده به این صورت می باشد:

شکل اصلی سیستم پیچیده =خواص پنجگانه + سلسله مراتب + A-IS سلسله مراتبOF-PART

سلسله مراتب A-IS :این سلسله مراتب برای مشخص نمودن وراثت ها مورد استفاده قرار میگیرد از لحاظ گفتاری می توان مثال زیر را مطرح کرد ANIMAL A IS CAT : این مشخص کننده وراثتی می باشد که گربه از خانواده حیوانات میگیرد.

سلسله مراتب OF-PART : بیانگر قسمتی از چیزی می باشد ،از لحاظ گفتاری می توان به این صورت نوشت BODY OF : PART IS HAND، این مثال مشخص کنند شیء یا قطعه ای از یک کلیت می باشد .پیاده سازی در قالب سلسله مراتب ما را در دریافت بهتر مساله و درک بهتر آن کمک کرده لذا موجب کاستن پیچیدگی می شود.

می توان ویژگی های استفاده از شیء گرایی را در کنترل پیچیدگی چنین ارزیابی کرد:

۱- منظم کردن فرآیند تولید نرم افزار

۲- بالا رفتن درک مساله در سیستم های نرم افزاری

۳- ارائه مدلی قدرتمندتر برای کنترل هرچه بیشتر پیچیدگی نرم افزاری

۴- کاهش هزینه های تولید (زمانی – مالی)

۵- استفاده مجدد و انعطاف پذیری بیشتر همینطور افزودن امکان پشتیبانی آسانتر

در این پایان نامه هدف ما این است که گروه های عملکردهای مهم (اهمیت نسبی اجزاء در نرم افزار علمی) را پیدا کنیم که در آن یک گروه مشابه ترین عملکردها را دارد. به جای استفاده از الگوریتم های خوشه بندی ، ما از رتبه بندی مرکز و قدرت استفاده می کنیم تا توابع را به ترتیب در نظر بگیریم و سپس گروه ها را با ارزیابی شباهت کسینوس آنها ایجاد کنیم. رتبه بندی مرکز و اقتدار (با استفاده از روش های طیفی) بر مقادیر ویژه نمایش ماتریس شبکه ها و جمع آوری اطلاعات جهانی در مورد ساختار تکیه می کند. در این فصل در مورد روش HITS که برای تجزیه و تحلیل خود استفاده کرده ایم بحث خواهیم کرد.

۱- ۳ ارزش های ویژه و بردارهای ویژه

اگر A یک ماتریس $n \times n$ باشد ، یک بردار غیر صفر x در R^n نامیده می شود اگر Ax یک مقیاس مقیاس x باشد که

$$Ax = \lambda x \qquad (3.1)$$

برای برخی از مقیاسهای λ مقیاس λ ارزش ویژه A و x گفته می شود که بردار ویژه A مربوط به λ است.

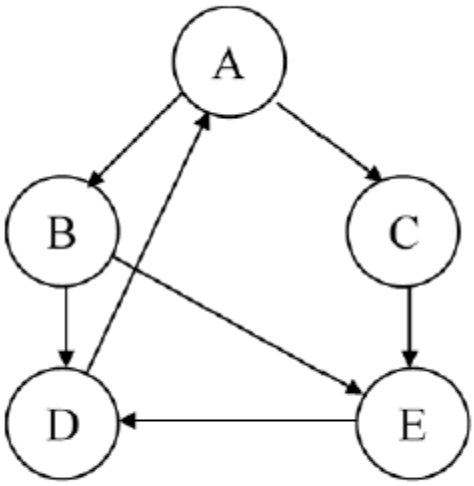
به عنوان مثال ، بردار $x = \begin{bmatrix}1\\2\end{bmatrix}$ بردار ویژه $A = \begin{bmatrix}4 & 0\\12 & -2\end{bmatrix}$ مربوط به ارزش ویژه λ = 4 می باشد

$$Ax = \begin{bmatrix}4 & 0\\12 & -2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}4\\8\end{bmatrix} = 4x$$

برای یافتن مقادیر ویژه یک ماتریس $n \times n$ ، معادله ۳,۱ را به صورت $Ax = \lambda Ix$ یا به شرح زیر بازنویسی می کنیم:

$$(\lambda I - A)x = 0 \qquad (3.2)$$

بنابراین معادله ۳,۲ دارای راه حل غیر صفر است اگر و فقط اگر $\det(\lambda I - A) = 0$. در اینجا می توانیم با یک مثال این اصطلاحات ریاضی را مورد بحث قرار دهیم. برای سادگی ، مثال ذکر شده در فصل قبل را به خاطر می آوریم.



(a)

	A	B	C	D	E
A		1	1	0	0
B	0		0	1	1
C	0	0		0	1
D	1	0	0		0
E	0	0	0	1	

(b)

شکل ۳,۱: یک مثال: الف) نمودار ، ب) ماتریس مجاور معادل یا DSM

از مثال نشان داده شده در شکل ۳,۱ ما DSM را به عنوان ماتریس A به عنوان معادله زیر دریافت می کنیم:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad (3.3)$$

در بخش ۳,۳ ما در مورد الگوریتم HITS از [۱۷] بحث خواهیم کرد. برای توصیف آن الگوریتم ما به دو ماتریس ویژه $B1 = AA^T$ و $B2 = A^TA$ نیاز داریم که A^T ترانسپورس ماتریس A است (ماتریس A یک ماتریس بولی است). در الگوریتم HITS ما بزرگترین ارزش ویژه و بردار ویژه مربوط به ماتریس های B_1 و B_2 را محاسبه می کنیم. این ماتریس های B_1 و B_2 ماتریس متقارن هستند. ماتریس های متقارن دارای مقادیر ویژه (واقعی) هستند.

فرمان [V,λ] =eig(B_1) در Octave ماتریس مورب λ از مقادیر ویژه و ماتریس V را برمی گرداند که ستون های آنها بردارهای اختصاصی مربوطه هستند به طوری که B_1* V = V* λ .

$$V = \begin{bmatrix} 0.00000 & -0.00000 & 0.00000 & 1.00000 & 0.00000 \\ 0.57735 & 0.00000 & -0.00000 & 0.00000 & 0.81650 \\ -0.57735 & 0.00000 & -0.70711 & 0.00000 & 0.40825 \\ 0.00000 & -1.00000 & 0.00000 & 0.00000 & 0.00000 \\ -0.57735 & 0.00000 & 0.70711 & 0.00000 & 0.40825 \end{bmatrix} \quad (3.4)$$

$$\lambda = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix} \quad (3.5)$$

به طور مشابه ما ماتریس بردارهای اختصاصی V را که با مقادیر ویژه λ ماتریس B_2 مرتبط است ، محاسبه می کنیم.

$$V = \begin{bmatrix} 0.00000 & 1.00000 & 0.00000 & -0.00000 & 0.00000 \\ -0.70711 & 0.00000 & 0.00000 & 0.70711 & 0.00000 \\ 0.70711 & 0.00000 & 0.00000 & 0.70711 & 0.00000 \\ 0.00000 & 0.00000 & -0.70711 & 0.00000 & 0.70711 \\ 0.00000 & 0.00000 & 0.70711 & 0.00000 & 0.70711 \end{bmatrix} \quad (3.6)$$

طیف DSM نمودار تماس و بردارهای اختصاصی مربوطه می توانند انبوهی از اطلاعات ساختاری در مورد شبکه زیرین را همانطور که در این پایان نامه نشان می دهیم ، نشان دهند. روش رتبه بندی طیفی که در این پایان نامه (HITS) استفاده می شود از بردارهای اختصاصی مرتبط با مقادیر ویژه انتخابی DSM مرتبط بدست می آید.

۲- ۳ مرکز و اقتدار

در شبکه Hubs و Authorities دو نوع گره مهم هستند. یک نمودار وابستگی همچنین می تواند به عنوان یک شبکه که در آن توابع به عنوان گره در نظر گرفته می شوند و وابستگی های آنها به عنوان قوس در نظر گرفته می شود. هاب ها گره هایی هستند که به گره های زیادی از نوع مهم اشاره می کنند ، جایی که مقامات این گره های مهم هستند. به عنوان مثال شکل ۳,۱ یک شبکه بین پنج عنصر را نشان می دهد که در آن یک تابع در صورت فراخوانی توابع دیگر ، هاب است و هنگامی که یک تابع با توابع دیگر فراخوانی می شود ، یک تابع است. از اینجا یک تعریف دایره ای به دست می آید: مراکز خوب آنهایی هستند که به بسیاری از مقامات خوب اشاره می کنند و مقامات خوب آنهایی هستند که بسیاری از مراکز خوب به آنها اشاره می کنند [۳].

۳-۳ جستجوی موضوعی ناشی از ابرمتن (HITS)

جستجوی موضوعی ناشی از ابرمتن (HITS) یک الگوریتم است که توسط کلاینبرگ [۱۷] استاد گروه علوم کامپیوتر در کرنل توسعه یافته است. این الگوریتم از ساختار پیوند وب برای کشف و رتبه بندی صفحات مربوط به یک موضوع خاص استفاده کرده است. رتبه بندی HITS متکی بر یک روش تکراری است که به یک راه حل ثابت همگرا می شود. به گفته کلاینبرگ ، به هر گره در شبکه i دو وزن غیر منفی اختصاص داده می شود: وزن قدرت (x_i) و وزن هاب (y_i). در ابتدا به هر x_i و y_i یک مقدار غیرجدی دلخواه داده می شود. سپس وزن ها با استفاده از رابطه ۳,۷ و معادله ۳,۸ برای $k = 1,2,3,...$ به روز می شوند.

$$x_i^{(k)} = \sum_{j:(j,i)\in E} y_j^{(k-1)} \qquad (3.7)$$

$$y_i^{(k)} = \sum_{j:(i,j)\in E} x_j^{(k)} \qquad (3.8)$$

- **وزن قدرت به روز رسانی**: در اینجا ما از رابطه ۳,۷ استفاده می کنیم. k^{th} در گره تکرار i یک وزن جدید $X_i^{(k)}$ تعیین می شود که برابر است با مجموع $y_j^{(k-1)}$ که مجموع آن بر روی هر گره j که به گره i اشاره می کند ، می گذرد. برای همه گره های نمودار ما از این مرحله استفاده می کنیم ، یعنی برای i=1,2,...,n (n تعداد گره های شبکه است).
- **به روز رسانی وزن هاب**: از معادله ۳,۸ در اینجا استفاده می شود. وزن hub جدید $y_i^{(k)}$ مجموع $X_i^{(k)}$ است که مجموع آن بر روی گره های j که گره i به آنها اشاره می کند ، می گذرد. این برای همه گره های نمودار تکرار می شود.

توجه داشته باشید که وزن های هاب از وزن های قدرت فعلی محاسبه می شوند که در آن وزن های مجاز از وزنهای هاب قبلی محاسبه شده است.

از روشی که در بالا توضیح داده شد ، ما رابطه وابستگی طبیعی بین هاب ها و مقامات را مشاهده می کنیم. مقدار y (هاب) یک گره بزرگ است اگر گره به تعداد زیادی از گره ها با مقادیر x بزرگ (مقادیر) اشاره کند و بالعکس [۱۸].

ما باید تمام هاب و مقادیر قدرت را برای همه گره ها پس از هر تکرار عادی کنیم تا

$$\sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\sum_{i=1}^{n} y_i^2} = 1.$$

اکنون ، در تکرار k برای n گره ، می توانیم مقدار hub و authority را از نظر بردارها نشان دهیم. اگر $\vec{x}_k$ نشان دهنده بردار مقادیر قدرت و $\vec{y}_k$ نشان دهنده بردار مقادیر مرکز در تکرار k باشد ، برای n گره داریم

$$\vec{x_k} = \begin{bmatrix} x_k(1) \\ x_k(2) \\ \vdots \\ x_k(n) \end{bmatrix} \qquad (3.9)$$

and

$$\vec{y_k} = \begin{bmatrix} y_k(1) \\ y_k(2) \\ \vdots \\ y_k(n) \end{bmatrix} \qquad (3.10)$$

اگر k = 0 ، سپس با استفاده از معادلات ۳,۹ و ۳,۱۰ می توانیم $\vec{x}_0$ و $\vec{y}_0$ را مقداردهی کنیم ، به عنوان

$x_0(1) = x_0(2) = \ldots = x_0(n) = (1 = \sqrt{n})$ و $y_0(1) = y_0(2) = \ldots = y_0(n) = (1 = \sqrt{n})$.

بگذارید A یک ماتریس مجاورت نمودار هدایت شده G. باشد سپس با استفاده از معادله ۳,۱۱ و ۳,۱۲ می توانیم الگوریتم گفته شده در بالا را نشان دهیم.

$$\vec{x}_k = c_k A^\top \vec{y}_{(k-1)} \qquad (3.11)$$

$$\vec{y}_k = c'_k A \vec{x}_k \qquad (3.12)$$

c_k و c'_k از رابطه ۳,۱۱ و معادله ۳,۱۲ به ترتیب ثابت های نرمال سازی هستند. در تکرار k اینها به گونه ای انتخاب می شوند که مجموع مربع وزن های قدرت و وزن توپی برابر ۱ باشد. با در نظر گرفتن این معادلات ، اکنون می توانیم روش HITS را با استفاده از معادلات زیر نشان دهیم [۱۷]:

$$\vec{x}_k = c_k c'_{(k-1)} A^\top A \vec{x}_{(k-1)} \quad \text{for } k > 1 \qquad (3.13)$$

$$\vec{y}_k = c'_k c_k A A^\top \vec{y}_{(k-1)} \quad \text{for } k > 0 \qquad (3.14)$$

بنابراین می توان گفت که HITS یک روش قدرت تکراری برای محاسبه بردار ویژه غالب برای AA^T و $A^T A$ است [۱۰]. بردار ویژه غالب ، ستون ماتریس V (بردار ویژه مربوطه) است که با ارزش ویژه مطابقت دارد.

نمرات هاب و نمرات اقتدار توسط ورودی های بردار ویژه غالب AA^T و $A^T A$ به ترتیب تعیین می شوند [۳].

مجدداً می توان شکل ۳,۱ را به عنوان مثال برای توضیح الگوریتم HITS در نظر گرفت که در آن ماتریس مجاورت آن نمودار در رابطه ۳,۳ آمده است.

جدول ۳,۱: Hub and Authority Ranking (نمرات مربوط به بردار ویژه غالب است)

Node	Hub Score	Hub Rank	Authority Score	Authority Rank
A	0.00000	3	0.00000	2
B	0.81650	1	0.00000	2
C	0.40825	2	0.00000	2
D	0.00000	3	0.70711	1
E	0.40825	2	0.70711	1

بردارهای ویژه AA^T و $A^T A$ مربوط به بزرگترین ارزش ویژه $\lambda_{max} = 3$ (به رابطه ۳,۵ مراجعه کنید) رتبه بندی هاب ها و مقامات (با استفاده از الگوریتم HITS) را که در جدول ۳,۱ نشان داده شده است ، به دست می آورند. در معادله ۳,۵ مشاهده می کنیم که ارزش ویژه غالب در ستون 5^{th} است و از این رو ستون پنجم V در معادلات ۳,۴ و ۳,۶ بردارهای ویژه غالب مربوطه هستند. ما اینها را به ترتیب نمره ای برای محاسبه رتبه بندی مرکز و رتبه در نظر می گیریم. در اینجا رتبه بندی گره های A تا E برای هاب ها {۳؛ ۱؛ ۲؛ ۳؛ ۲} و رتبه بندی مقامات {۲؛ ۲؛ ۲؛ ۱؛ ۱} است.

فصـــل چهارم

روش و نتایج

بسیاری از نرم افزارها معمولاً توسط متخصصان حوزه نوشته می شوند و مشکلات خاصی را برطرف می کنند. بیشتر اوقات این نرم افزارها به دلایل مختلف دارای اسناد فنی کافی نیستند. بنابراین از دیدگاه قابلیت استفاده ، تشخیص و بازیابی اجزایی که می توانند در پروژه های نرم افزاری دیگر مورد استفاده مجدد قرار گیرند ، دشوار است. تجزیه و تحلیل چنین نرم افزاری برای محققان چالش برانگیزتر است.

در این پایان نامه ما وابستگی های بین توابع ترسیم شده توسط نمودارهای فراخوانی استاتیک را برای دسته بندی توابع در گروه های "توابع مشابه" (با استفاده از وابستگی های پنهان به شرح زیر) با توجه به "اهمیت" آنها (با رتبه بندی طیفی توابع با استفاده از الگوریتم HITS) تجزیه و تحلیل می کنیم. سیستم نرم افزاری توابع هاب که "مهمترین" طبقه بندی شده اند ، توابعی هستند که خدمات اصلی را به کاربران نهایی ارائه می دهند. توابع مربوطه مهمترین "ارائه دهندگان خدمات" را برای توابع هاب نشان می دهند.

یک تابع Hub به طور مستقیم در اجرای عملکرد اصلی نرم افزار مشارکت می کند. نمونه هایی از توابع Hub در CSparse عبارتند از cs lusol (یک سیستم خطی را با ماتریس ضریب نامتقارن حل می کند) و cs lsolve (یک سیستم خطی مثلثی پایین را حل می کند). از طرف دیگر توابع که وظیفه ارائه خدمات پشتیبانی به سیستم نرم افزاری را بر عهده دارند ، توابع Authority نامیده می شوند. نمونه هایی از عملکرد Authority در CSparse عبارتند از cs realloc (تغییر اندازه یک بلوک از حافظه) و cs انجام شده (فضای کار را آزاد می کند و یک ماتریس پراکنده را برمی گرداند).

در نمودار فراخوانی اگر تابع i تابع j را فرا می خواند ، می گوییم که به j بستگی دارد. این نوع وابستگی صریح است زیرا می توان آن را مستقیماً از کد منبع استخراج کرد. فرض کنید تابع i و j هر دو تابع k را فراخوانی می کنند. به طور شهودی به این معنی است که توابع i و j به نوعی ارتباط دارند (بسته به زمینه). این یک مثال از وابستگی "پنهان" است که از نمودار تماس قابل تشخیص نیست. ما این نوع وابستگی های پنهان را از ماتریس محصول $B = A^* A^T$ محاسبه می کنیم. مقدار غیر صفر B (i,J) دلالت بر "وابستگی پنهان" بین توابع i و j دارد.

در این کار ما ساختار وابستگی نرم افزار (رابطه تماس گیرنده بین توابع) را با استفاده از ابزار "درک" [۲۱] و "شباهت" بین عناصر طراحی (توابع) با استفاده از یک متریک شباهت مناسب (شباهت کسینوس) کشف می کنیم. سپس با مفهوم "اهمیت" عناصر طراحی [۱۳] ترکیب می کنیم و عناصر طراحی را بر اساس "اهمیت" آنها در "طبقه" طبقه بندی می کنیم. این فصل تأثیر وابستگی های عملکردی بر معماری نرم افزار و همچنین تأثیر کشف "شباهت" بین عناصر طراحی را مورد بحث قرار می دهد. روش شناسی و نتایج تجربی کار جدید ما نیز در این فصل مورد بحث قرار گرفته است.

۱ – ۴ روش شناسی

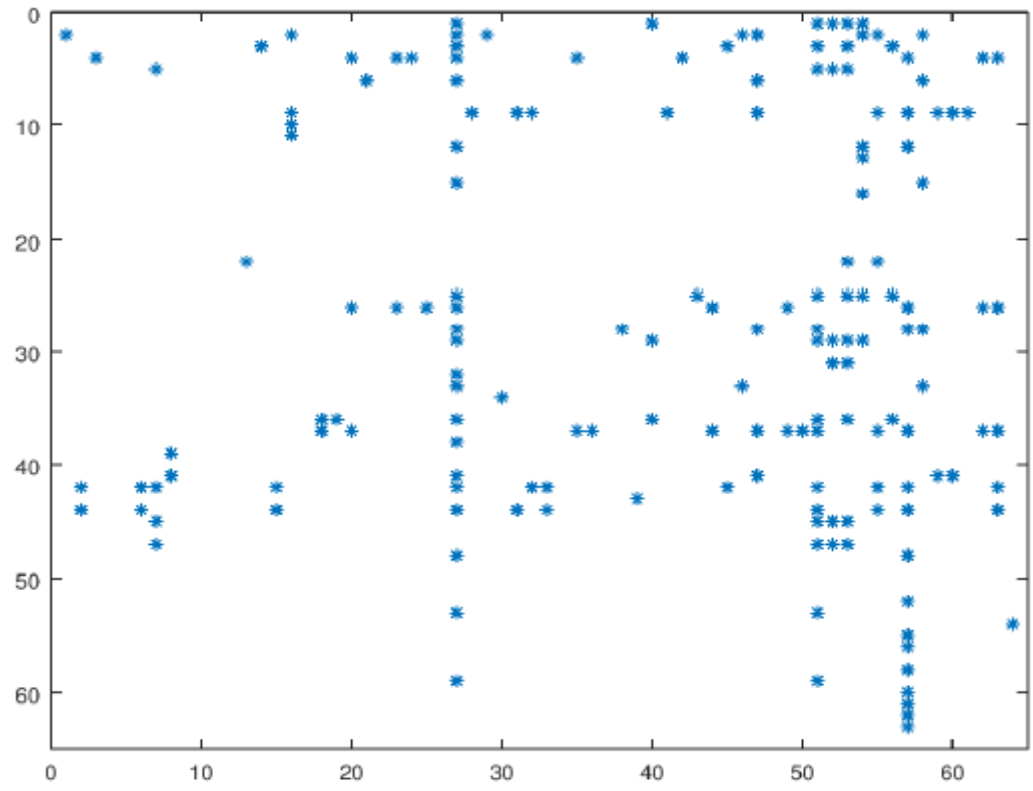
این بخش را با توصیف روش استخراج وابستگی و سپس ساخت DSM شروع می کنیم. با استفاده از ابزار تجزیه و تحلیل "درک" [۲۱] ما می توانیم ساختار وابستگی نرم افزار را مشاهده و استخراج کنیم. ما برای تجزیه و تحلیل ساختار وابستگی برنامه ها ، ارتباط بین تماس گیرنده بین عملکردها را استخراج می کنیم. روش یافتن "اهمیت" عناصر طراحی [۱۳] محاسبه رتبه بندی hub و اقتدار است که لیستی از تماس گیرندگان مهم (هاب ها) و تماس گیرندگان مهم (مقامات) را ارائه می دهد. سپس روش کشف "شباهت" بین عناصر طراحی را مورد بحث قرار می دهیم. هدف ما این است که عناصر طراحی را بر اساس "اهمیت" که با استفاده از یک شبه کد ارائه شده است "طبقه بندی" کنیم.

زیرساخت های محاسباتی برای عملیات تحقیقاتی [۸] (COIN-OR) یکی از بزرگترین و گسترده ترین جوامع منبع باز برای نرم افزارهای تحقیقاتی علمی است. ما پروژه های نرم افزار منبع باز از COIN-OR را مطالعه کردیم. به عنوان مثال نرم افزار CSparse و ADOL-C که در C اجرا شده اند به ترتیب در بخش های ۴,۲,۱ به طور مختصر شرح داده می شود. سپس نتایج این نرم افزار که از آزمایشات خود بدست آورده ایم در بخش ۴,۳ مورد بحث قرار می گیرد.

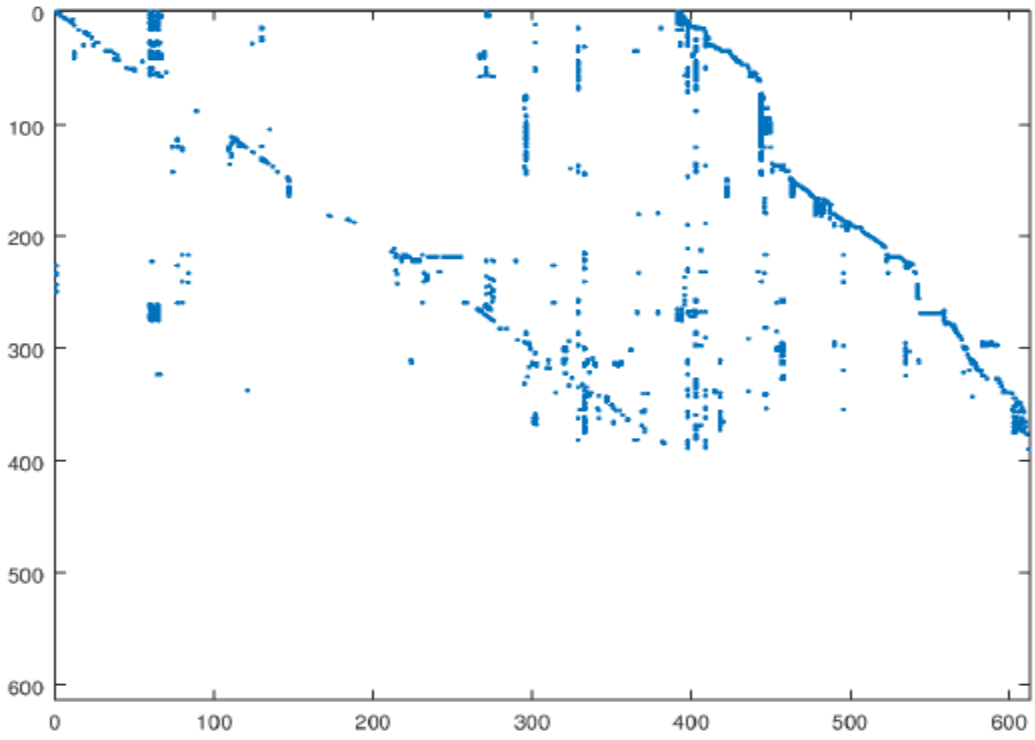
۱-۱-۴ استخراج وابستگی ها

در بخش ۲,۲ مقدماتی در مورد استخراج وابستگی ها ارائه شده است.

در این بخش ما نحوه استخراج وابستگی های فراخوانی بین عملکردها (برای CSparse و ADOL-C) با استفاده از "درک" SciTool [۲۱] را مورد بحث قرار می دهیم. ما قبلاً در مورد این ابزار در بخش قبلی بحث کرده ایم. ما می توانیم این نرم افزارها (CSparse و ADOLC) را با استفاده از Octave تجسم کنیم. شکل ۴,۱ و شکل ۴,۲ ماتریس های CSparse و ADOL-C را به ترتیب با استفاده از Octave ترسیم می کند.



شکل ۴,۱: ماتریس پراکنده CSparse



شکل ۴,۲: ماتریس پراکنده ADOL-C

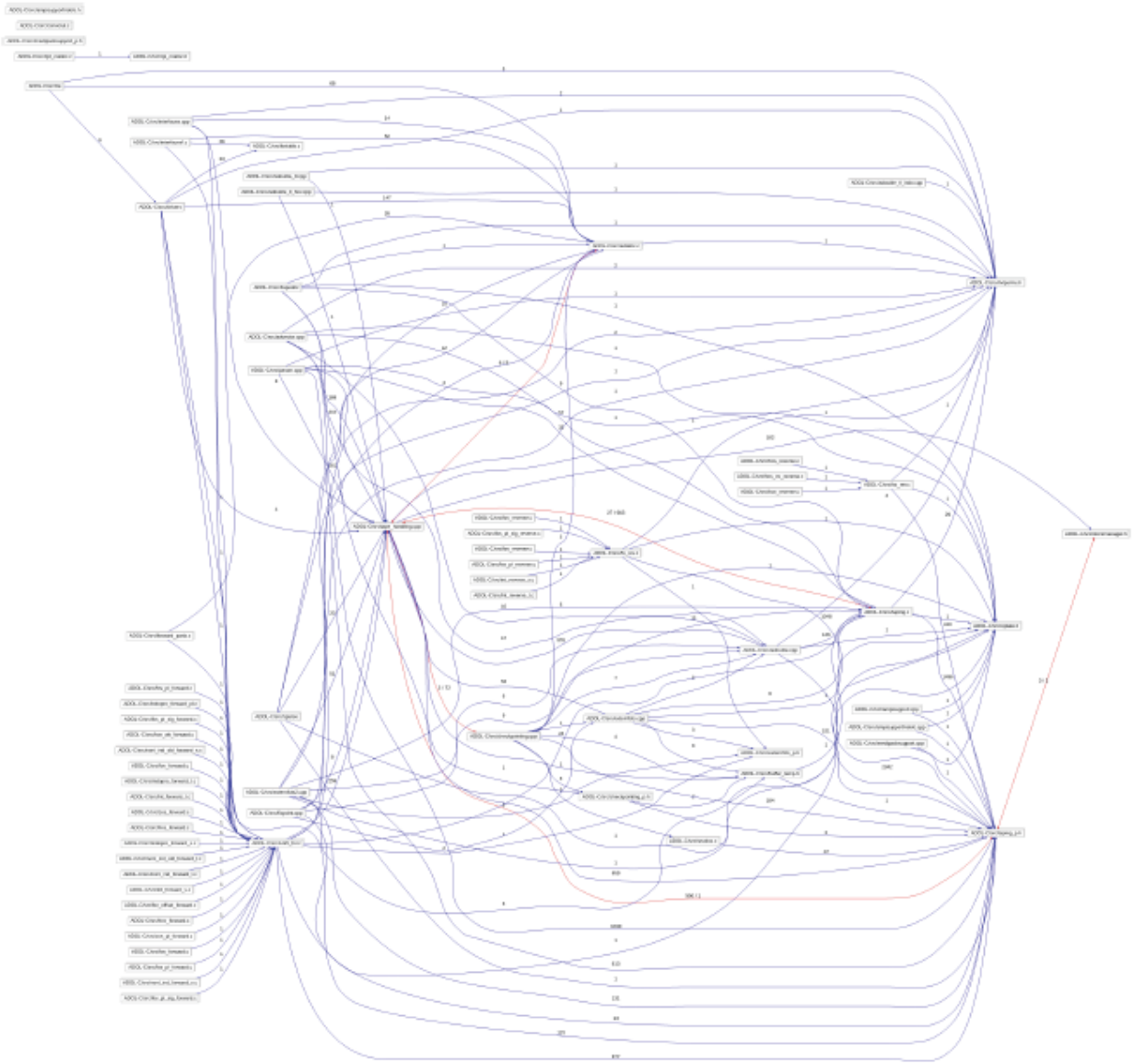
برای استخراج وابستگی ها ابتدا بسته کامل (آخرین نسخه) نرم افزار سیستم (CSparse و ADOL-C) را بارگیری می کنیم. سپس بسته را در "SciTool "Understand وارد می کنیم. سپس می توان نمودارهای تماس را تجسم کرد (به شکل ۴,۳ و شکل ۴,۴ مراجعه کنید). سپس هر تابع را بازرسی کرده و جزئیات آن را از گزینه description مشاهده می کنیم. در این توضیحات ما می توانیم لیستی از توابع را که توسط آن تابع خاص فراخوانی شده اند و همچنین اطلاعات زیادی را بیابیم. ما همچنین می توانیم ماتریس های وابستگی را در قالب های مختلف فایل صادر کنیم.

۴ - ۱ - ۲ ساخت DSM ها

ما DSM را با وابستگی های ایستا ایجاد می کنیم. DSM یک نمایش بصری از یک سیستم و به شکل یک ماتریس مربع است. ما پروژه CSparse با ۶۴ عملکرد و پروژه ADOL-C را با ۶۱۲ عملکرد بررسی کرده ایم. بنابراین CSparse دارای DSM با ماتریس X64۶۴ و ماتریس DSM 612X612 برای ADOL-C است. این ماتریس ها دوتایی هستند زیرا ما ماتریس های وزنی را در نظر نگرفتیم. این بدان معناست که حتی اگر یک تابع i تابع دیگر j را بیش از یک بار فراخوانی کند ، فرض می کنیم DSM [i, j] = 1 است.

شکل ۴,۳: نمودار وابستگی CSparse

شکل ۴,۵ DSM را برای CSparse نشان می دهد. ترتیب توابع (در سطرها یا ستون ها) در این ماتریس باینری X64۶۴ را می توان از ضمیمه A یافت. ما می بینیم که DSM [2, 1] = 1 به معنی تابع cs و فراخوانی تابع cs افزودن است. سلول های خالی 0 در نظر گرفته می شوند. ما برای ADOL-C نیز DSM مشابه (ماتریس دوتایی X612۶۱۲) داریم.

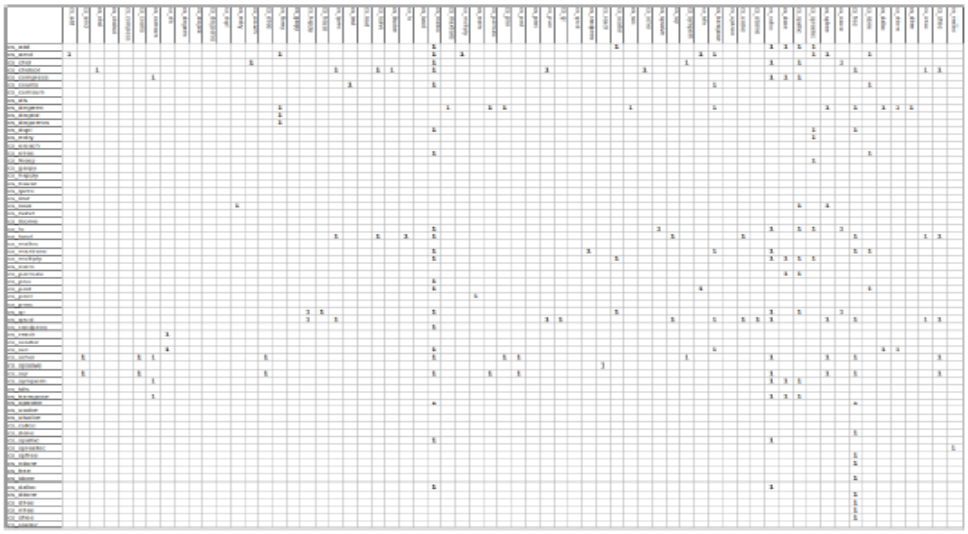


شکل ۴,۴: نمودار وابستگی ADOL-C

۳ - ۱ - ۴ مرکز محاسبات و رتبه بندی قدرت

در فصل ۳ با استفاده از یک مثال کوچک ، ما برخی از روش های تجزیه و تحلیل وابستگی بین توابع را مورد بحث قرار داده ایم. برای مثال محاسبه ارزش ویژه و بردار ویژه از ماتریس مشخص شده ، هاب ها و مقامات را مشخص می کند و از روش HITS برای رتبه بندی توابع استفاده می کند. در این بخش ما اهمیت و روش محاسبه مرکز و رتبه بندی قدرت را برای رویکرد خود شرح می دهیم.

برای یک ماتریس مشخص ، الگوریتم HITS دو ماتریس $B_1 = AA^T$ و $B_2 = A^T A$ را محاسبه می کند. ماتریس های B_1 و B_2 به ترتیب نقش مهمی در محاسبه رتبه بندی مرکز و رتبه بندی قدرت دارند.



شکل ۴,۵: DSM از CSparse

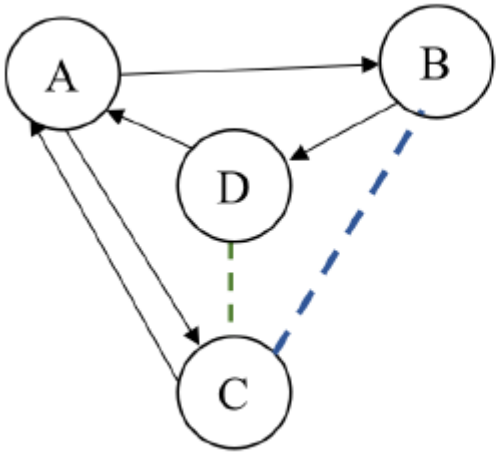
شکل ۴,۶ نمونه ای از نمودار فراخوانی را نشان می دهد که برای آن دریافت می کنیم ،

$$B_1 = AA^\top = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \mathbf{1} \\ 0 & 0 & \mathbf{1} & 1 \end{bmatrix} \tag{4.1}$$

$$B_2 = A^\top A = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

بنابراین مقدار مورب ماتریس B_1 , diag (B_1) = [۲؛ ۱؛ ۱؛ ۱] نشان دهنده خارج از توابع و مقدار مورب دیاگرام ماتریس B_2, (B_2) = [۲؛ ۱؛ ۱؛ ۱] نشان دهنده عدم تابع توابع است به علاوه بر این در معادله ۴,۱ و ۴,۲ برخی از مقادیر خارج از مورب را مشاهده می کنیم (به عنوان متن پررنگ گزارش شده است). معادله ۴,۱ بیان می کند که تابع C و تابع D رابطه غیرمستقیمی دارند

هرچند که هر دو تابع A را می نامند (خط نقطه سبز در شکل ۴,۶). از طرف دیگر معادله ۴,۲ بیان می کند که تابع B و تابع C رابطه ای غیر مستقیم دارند هرچند که هر دو با تابع A (خط نقطه آبی در شکل ۴,۶) نامیده می شوند.



	A	B	C	D
A	0	1	1	0
B	0	0	0	1
C	1	0	0	0
D	1	0	0	0

شکل ۴,۶: نمونه ای از Hub و Authority

بنابراین ماتریس های محاسبه B_1 و B_2 رابطه وابستگی معنایی می دهد که در آن DSM فقط رابطه وابستگی نحوی بین توابع را ارائه می دهد.

فرض کنید ما DSM خود را از کد منبع داده شده که A است ساخته ایم. حالا ماتریس $B_1 = AA^T$ و $B_2 = A^T A$ را محاسبه کنید ، جایی که A^T ترانسپورس ماتریس A است.

سپس با استفاده از Octave می توان ماتریس مورب λ از مقادیر ویژه و ماتریس V را پیدا کرد که ستونهای آنها بردارهای اختصاصی مربوطه هستند به طوری که $B_i * V = V * \lambda$ جایی که B_i B_1 و B_2 است. ما به دنبال ارزش ویژه غالب λ هستیم که بیشترین مقدار λ را در خود دارد. ستون مربوط به V بردار ویژه غالب است که به عنوان مقدار مرکز (برای B_1) و مقدار قدرت (برای B_2) در نظر گرفته می شود [۱۳]. ما توابع را به ترتیب نزولی با توجه به مقادیر آنها مرتب می کنیم (بخش ۳,۱ را ببینید) و از این رو رتبه قدرت و رتبه بندی توابع را بدست می آوریم.

در جداول ۴,۱ و ۴,۲ ما ۵ مقام برتر و هاب را با توجه به مقادیر آنها برای هر دو نرم افزار CSparse و ADOL-C گزارش کرده ایم.

جدول ۴,۱: Hub and Authority رتبه پنج عملکرد اول پروژه CSparse

Rank	Hub	Hub Name	Authority	Authority Name
1	42	*cs_schol*	27	*cs_malloc*
2	44	*cs_sqr*	51	*cs_calloc*
3	37	*cs_qrsol*	57	*cs_free*
4	28	*cs_maxtrans*	53	*cs_spalloc*
5	29	*cs_multiply*	63	*cs_sfree*

جدول ۴,۲: Hub and Authority رتبه پنج عملکرد اول پروژه ADOL-C

Rank	Hub	Hub Name	Authority	Authority Name
1	36	*tape_doc*	60	*myalloc1*
2	4	*jacobian*	403	*fprintf*
3	58	*forward*	64	*myfree1*
4	29	*inverse_Taylor_prop*	329	*adolc_exit*
5	9	*hessian*	61	*myalloc2*

۴ - ۱ - ۴ شباهت های محاسبه کسینوس

شباهت با کسینوس معیاری است که برای تعیین میزان شباهت توابع به کار می رود. از نظر ریاضی ، کسینوس زاویه بین دو بردار را که در یک فضای چند بعدی پیش بینی شده اند اندازه گیری می کند. در این زمینه دو بردار (دو ردیف DSM) ماتریسی هستند که اطلاعات تماس دو تابع را شامل می شوند. هنگامی که در یک فضای چند بعدی ترسیم می شود که هر بعد با عملکردی در سیستم مطابقت دارد ، شباهت کسینوس جهت (زاویه) توابع را نشان می دهد و نه اندازه. شباهت کسینوس مزیت دارد زیرا حتی اگر دو تابع مشابه از نظر رتبه بندی با هم فاصله داشته باشند ، باز هم می توانند زاویه کوچکتری بین آنها وجود داشته باشد. هرچه زاویه کوچکتر باشد ، شباهت بیشتر است.

معادله ۴,۳ cos (Θ) (شباهت کسینوس) بین بردارهای $\vec{a}$ و $\vec{b}$ را نشان می دهد.

$$\cos(\theta) = \frac{\vec{a}\vec{b}}{\|\vec{a}\|\|\vec{b}\|} \qquad (4.3)$$

جایی که $\|\vec{a}\|$ و $\|\vec{b}\|$ بردارهای یک اندازه هستند ، هنجار اقلیدسی این بردارها هستند و n اندازه این بردارها و تعداد عناصر موجود در سیستم است.

DSM شامل اطلاعات تماس بین توابع است که در آن سطرها توابع تماس گیرنده (هاب) و ستون ها توابع تماس گیرنده (اقتدار) هستند. وقتی شباهت کسینوس بین دو تابع تماس گیرنده را محاسبه می کنیم ، دو ردیف متناظر از DSM را به عنوان بردارهای $\vec{a}$ و $\vec{b}$ انتخاب می کنیم. دوباره هنگامی که شباهت کسینوس بین دو تابع callee را محاسبه می کنیم ، دو ستون متناظر از DSM را به عنوان بردارهای $\vec{a}$ و $\vec{b}$ انتخاب می کنیم.

در رویکرد ما DSM و ماتریس $B_1 = AA^T$ (hub) و $B_2 = A^T A$ (اقتدار) را برای انتخاب بردارها برای بررسی شباهت در نظر می گیریم. اهمیت ماتریس های B_1 و B_2 در بخش ۴,۱,۳ مورد بحث قرار گرفته است ، جایی که ما نشان داده ایم که ماتریس های B_1 و B_2 اطلاعات مهم تری نسبت به DSM دارند. بنابراین ما رویکرد خود را برای هر دو نوع ماتریس آزمایش کردیم.

از رابطه ۴,۳ مشاهده می کنیم که برای محاسبه محصول نقطه بین دو بردار ($\sum_{i=1}^{n} \vec{ai}\ \vec{bi}$) مجموع حاصل ضرب ورودی های مربوطه دو ردیف یا ستون انتخاب شده را محاسبه می کند.

ما می دانیم $\cos(0°) = 1$ و $\cos(90°) = 0$. بنابراین اگر دو بردار متعامد باشند (مشابه هم نباشند) مقدار $\cos(\Theta)$ 0 و اگر دو بردار موازی (مشابه) باشند ، مقدار $\cos(\Theta)$ 1 خواهد بود. اما اگر مقدار $\cos(\Theta)$ بین ۰ و ۱ باشد ، می توانیم آستانه ای برای تشخیص شباهت تعیین کنیم. ما الگوریتم خود را برای مقدار آستانه مختلف بین ۰ و ۱ آزمایش کرده ایم. به عنوان مثال ، 0 .0 ، 0 0. 1 ، 9. 0 ،..... 2 .0 ، 1.

علاوه بر این شباهت های بین توابع برای هاب ها و مقامات را محاسبه کرده ایم. برای مرکز دو بردار سطر از ماتریس و برای مقامات دو بردار ستون از ماتریس انتخاب می کنیم.

اکنون شباهت های کسینوس بین پنج تابع اول (با توجه به رتبه آنها به جدول ۴,۱ و جدول ۴,۲ مراجعه کنید) را با توجه به هاب ها و مقامات (برای پروژه های CSparse و ADOL-C) محاسبه می کنیم.

جدول ۴,۳ و جدول ۴,۴ شباهت های کسینوس بین توابع را نشان می دهد که در آن $\vec{a}$ و $\vec{b}$ توابع پروژه ها را نشان می دهند (پیوست A و B را ببینید). در اینجا مقدار $\cos(\Theta)$ را بین ۰ و ۱ بدست می آوریم. بنابراین با تعیین یک آستانه می توان گفت که آیا دو تابع مشابه هستند یا خیر.

جدول ۴,۳: شباهت های کوزین با پنج عملکرد اول (با توجه به مرکز و رتبه اقتدار) پروژه CSparse

Hub Functions			Authority Functions		
$\vec{a}$	$\vec{b}$	cos(θ)	$\vec{a}$	$\vec{b}$	cos(θ)
cs_schol	*cs_sqr*	0.982167	*cs_calloc*	*cs_spalloc*	0.873418
cs_sqr	*cs_maxtrans*	0.798325	*cs_free*	*cs_sfree*	0.861267
cs_schol	*cs_maxtrans*	0.776425	*cs_malloc*	*cs_calloc*	0.848427
cs_sqr	*cs_qrsol*	0.722491	*cs_malloc*	*cs_sfree*	0.678954
cs_schol	*cs_qrsol*	0.696213	*cs_malloc*	*cs_free*	0.674013
cs_maxtrans	*cs_multiply*	0.69317	*cs_malloc*	*cs_spalloc*	0.636215
cs_qrsol	*cs_maxtrans*	0.691092	*cs_calloc*	*cs_sfree*	0.619324
cs_schol	*cs_multiply*	0.590327	*cs_calloc*	*cs_free*	0.557076
cs_sqr	*cs_multiply*	0.577527	*cs_spalloc*	*cs_sfree*	0.262881
cs_qrsol	*cs_multiply*	0.400871	*cs_free*	*cs_spalloc*	0.220262

جدول ۴,۴: شباهت های کوزین با پنج عملکرد اول (با توجه به مرکز و رتبه اقتدار) پروژه ADOL-C

Hub Functions			Authority Functions		
$\vec{a}$	$\vec{b}$	cos(θ)	$\vec{a}$	$\vec{b}$	cos(θ)
jacobian	*hessian*	0.953637	*myalloc1*	*myfree1*	0.970463
inverse_Taylor_prop	*hessian*	0.89385	*fprintf*	*adolc_exit*	0.944289
jacobian	*inverse_Taylor_prop*	0.822776	*myfree1*	*myalloc2*	0.879903
tape_doc	*forward*	0.68348	*myalloc1*	*myalloc2*	0.865987
jacobian	*forward*	0.529026	*myalloc1*	*adolc_exit*	0.359106
forward	*hessian*	0.504676	*myalloc1*	*fprintf*	0.317976
forward	*inverse_Taylor_prop*	0.407995	*adolc_exit*	*myalloc2*	0.243542
tape_doc	*hessian*	0.315302	*myfree1*	*adolc_exit*	0.221187
tape_doc	*jacobian*	0.295122	*fprintf*	*myalloc2*	0.211266
tape_doc	*inverse_Taylor_prop*	0.234508	*fprintf*	*myfree1*	0.190357

۵ – ۱ – ۴ الگوریتم

الگوریتم کامل رویکرد ما برای یافتن توابع مشابه در سطوح مختلف در زیر ارائه شده است.

الگوریتم ۱: گروه بندی توابع مشابه (DSM A)

```
1  N ← Number of functions
2  threshold ← a numeric value between 0 and 1
3  k ← 0                                                    ▷ Number of tiers
4  while N > 0 do
5      h ← List of top 5 elements in hub ranking order
6      a ← List of top 5 elements in authority ranking order
7      U ← [ ]          ▷ U is the list of elements to be removed from A after each iteration
8      for i ← 1 to 5 do
9          hub_Similarity[i] ← 0          ▷ Store similarity between hub elements from h
10         aut_Similarity[i] ← 0      ▷ Store similarity between authority elements from a
11     for i ← 1 to 4 do
12         for j ← i + 1 to 5 do
13             hub_Similarity[i] ← hub_Similarity[i] + cosineSimilarity(h[i], h[j])
14                                 ▷ cosineSimilarity() is a function as Equation 4.3
15             hub_Similarity[j] ← hub_Similarity[j] + cosineSimilarity(h[i], h[j])
16     for i ← 1 to 5 do
17         hub_Similarity[i] ← hub_Similarity[i]/4          ▷ Calculating average similarity
18     k ← k + 1
19     for i ← 1 to 5 do
20         if hub_Similarity[i] >= threshold then
21             Include h[i] in T_k
22             U ← U ∪ h[i]
23     for i ← 1 to 4 do
24         for j ← i + 1 to 5 do
25             aut_Similarity[i] ← aut_Similarity[i] + cosineSimilarity(a[i], a[j])
26             aut_Similarity[j] ← aut_Similarity[j] + cosineSimilarity(a[i], a[j])
27     for i ← 1 to 5 do
28         aut_Similarity[i] ← aut_Similarity[i]/4          ▷ Calculating average similarity
29     k ← k + 1
30     for i ← 1 to 5 do
31         if aut_Similarity[i] >= threshold then
32             Include a[i] in T_k
33             U ← U ∪ a[i]
34     Remove all i ∈ U from A
35     N ← N − |U|
36 return T_1, T_2, ..., T_k                    ▷ T_i is a tier containing similar functions
```

در این الگوریتم DSM A ورودی است و لیستی از سطوح دارای عملکردهای مشابه خروجی است (T1، T2، ..... Tk). آستانه یک مقدار عددی از پیش تعریف شده بین ۰ تا ۱ است. در بخش ۴،۱،۳ ما نحوه محاسبه توزیع مرکز و قدرت توابع را مورد بحث قرار دادیم. با پیروی از روش مشابه در مرحله ۵ ، یک لیست (h) از ۵ عملکرد برتر را با توجه به رتبه بندی هاب دریافت می کنیم و در مرحله ۶ یک لیست (a) از ۵ عملکرد برتر را با توجه به رتبه بندی بدست می آوریم.

اکنون نحوه محاسبه سطوح هاب را توضیح می دهیم. در مراحل ۱۱ تا ۱۵ شباهت کسینوس بین این ۵ توابع هاب را که در بخش ۴،۱،۴ توضیح داده شده است محاسبه می کنیم. سپس میانگین شباهت هر یک از توابع هاب را در مرحله ۱۷ محاسبه می کنیم. در مرحله ۲۰ الگوریتم بررسی می کند که آیا میانگین شباهت تابع h [i] (یعنی شباهت هاب [i]) بیشتر یا مساوی با آستانه از پیش

تعریف شده است یا خیر. اگر شرط را برآورده کند ، تابع در ردیف T_k قرار می گیرد و همچنین در U ذخیره می شود تا پس از تکرار فعلی از ماتریس A حذف شود ، در غیر این صورت هیچ کاری برای آن تابع انجام نمی دهد. به طور مشابه برای سطوح اقتدار ، مراحل ۲۳ تا ۳۳ را دنبال کردیم. بنابراین در یک تکرار حلقه while دو ردیف را یکی برای هاب و دیگری برای اقتدار محاسبه می کنیم. الگوریتم تا زمانی ادامه می یابد که هیچ توابع وجود نداشته باشد یا چنین سطحی با شرایط تعیین شده محاسبه نشود.

۴ – ۲ تنظیمات

این بخش جزئیات مربوط به تنظیمات مطالعه ای را که در کار خود استفاده می کنیم آورده است. در اینجا ،در مورد سیستم های هدف خود و انتخاب آستانه برای آزمایش های خود بحث می کنیم.

۴ - ۲ - ۱ سیستم های هدف

برای آزمایش دو نرم افزار پیاده سازی شده در CSparse, C/C ++ نسخه 5,6,0 و ADOL-C نسخه 2.7.2 را انتخاب می کنیم. این نرم افزارهای علمی برای محاسبه مشتقات عددی دقیق (تا ماشین دقیق) از برنامه عملکرد در یک نقطه مشخص استفاده می شوند. نرم افزار CSparse به حل سیستم معادله خطی $Ax = b$ می پردازد که در آن ماتریس ضریب A پراکنده است [۶].
از طرف دیگر ADOL-C یک سیستم نرم افزاری برای محاسبه مشتقات ریاضی (ضرایب گرادیان ، ژاکوبیان ، هسیان ، تیلور) یک تابع ریاضی است [۱۲].

CSparse

CSparse یک پروژه است که شامل روش های مستقیم برای سیستم های خطی پراکنده است. مشکلات زیادی در زمینه محاسباتی وجود دارد که به حل سیستم های پراکنده معادلات خطی می پردازد. برای حل موثر این مشکلات ، ما نیاز به دانش عمیق الگوریتم های نظریه و ساختار داده های موجود در کتابخانه های نرم افزار ماتریس پراکنده داریم. CSparse اصول اولیه الگوریتم های ماتریس پراکنده را برای ارائه زمینه لازم ارائه می دهد [۶]. این پروژه بسته ماتریس پراکنده قابل بارگیری است که الگوریتم ها و قضایای ارائه شده [۶] را نشان می دهد. برای کار با این پروژه ، کاربر باید در مورد بسته های نرم افزاری بزرگتر و پیچیده تر و همچنین یک ایده قوی در مورد MATLAB و زبان برنامه نویسی C داشته باشد. برای درک بیشتر در مورد این پروژه (سیستم های خطی پراکنده) پیشنهاد می کنیم از [۶] ایده بگیرید.
توابع توسط نویسنده نرم افزار به شرح زیر طبقه بندی می شوند: ابزار اولیه ، اصلی ، ثانویه ، کاربرد ثانویه ، ابزار سوم و سوم [۶].

ADOL-C

نرم افزار ADOL-C در C ++/C پیاده سازی شده است [۱۲]. این بسته ارزیابی مشتقات اول و بالاتر توابع بردار نوشته شده در C ++/C را تسهیل می کند. با استفاده از C، C ++، Fortran یا هر زبان دیگری که می توان با C پیوند داد ، همه می توانند از همه روال های موجود در این بسته استفاده کنند.

مقادیر عددی عاری از بردارهای مشتق بدون خطا را می توان با زمان اجرای کارآمد و فضای کوچک توسط برنامه ارزیابی عملکرد داده شده محاسبه کرد. ماتریس های مشتق شده توسط ستون ها به صورت سطر یا در قالب پراکنده بدست می آیند. برای منحنی های راه حل تعریف شده توسط معادلات دیفرانسیل معمولی ، روالهای خاصی ارائه شده است که بردارهای ضریب تیلور و ژاکوبیان آنها را با توجه به بردار حالت فعلی ارزیابی می کند. برای توابع صریح یا ضمنی ، تنسورهای مشتق با پیچیدگی بدست می آیند که فقط در درجه خود به صورت درجه دوم رشد می کند. محاسبات مشتق شامل مقدار قابل توجهی از داده اما همیشه قابل پیش بینی است. به صورت متوالی به این داده ها دسترسی داده می شود و بنابراین می تواند به صورت خودکار در پرونده های خارجی صفحه بندی شود.

۲ - ۲ - ۴ انتخاب آستانه

در الگوریتم ۱ ما در مورد یک آستانه (مرحله ۲ در الگوریتم ۱) صحبت کرده ایم که به تصمیم گیری در مورد اینکه آیا یک تابع باید در ردیف قرار گیرد یا خیر (مراحل ۲۰ و ۳۱) کمک می کند. مقدار آستانه باید بین ۰ تا ۱ انتخاب شود زیرا می دانیم که مقدار $\cos(\Theta)$ از ۰ تا ۱ متغیر است. در این تز ارزش شباهت کسینوس ($\cos(\Theta)$) نمی تواند منفی باشد زیرا بردارها مثبت از آنجا که بردارها از ماتریس وابستگی انتخاب می شوند (وابستگی نمی تواند ارزش منفی داشته باشد) بردارها همیشه در ربع اول قرار دارند. اگر مقدار $\cos(\Theta)$ آنها نزدیک به ۱ باشد دو تابع بیشتر شبیه هم هستند (توجه: اگر دو بردار مشابه یا موازی باشند به این معنی است که زاویه آنها 0^o است بنابراین $\cos(0^o) = 1$). از طرف دیگر ، اگر مقدار $\cos(\Theta)$ بین دو بردار نزدیک به 0 باشد ، آنها بیشتر شباهت ندارند.

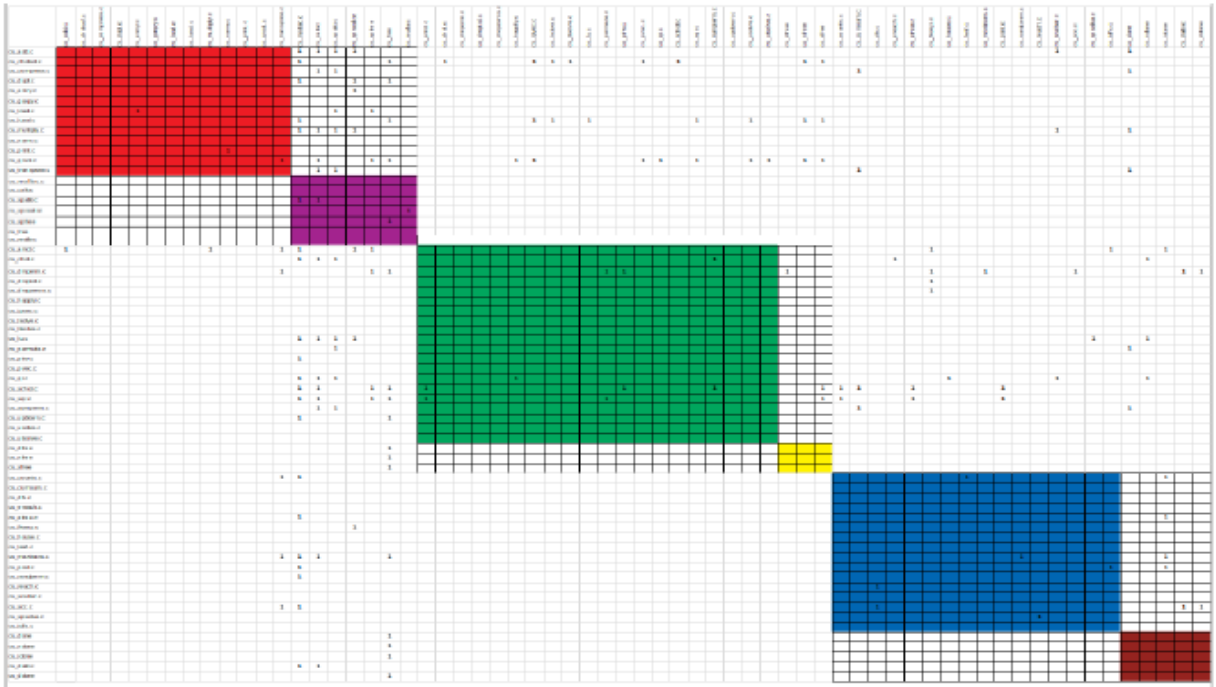
۳ – ۴ نتایج

در این بخش ما نتایج آزمایشات عددی روی پروژه های منتخب را ارائه می دهیم. نرم افزار آزمایشات از زیرساخت های محاسباتی برای عملیات (COIN OR) [۸] به دست آمده است. آزمایشات با استفاده از رایانه ای با پردازنده Intel Xeon 3.4 گیگاهرتز ۸ گیگابایت RAM و لینوکس انجام شد. زبان پیاده سازی GNU Octave بود و کد با کامپایلر نسخه 4.2.2 کامپایل شد.

نتایج آزمایش بسته انتخابی CSparse در شکل های ۴,۸ ، ۴,۹ ، ۴,۱۰ و ۴,۱۱ گزارش شده است. در اینجا شکل های ۴,۸ و ۴,۹ نتایج (عملکردهای سطوح مختلف) را نشان می دهد که در آن ما DSM را برای انتخاب توابع برای بررسی شباهت آنها در نظر گرفتیم. از طرف دیگر جداول ۴,۱۰ و ۴,۱۱ نتایج را نشان می دهند که در آن AA^T و A^TA (به ترتیب برای هاب و اقتدار) برای انتخاب توابع برای بررسی شباهت آنها در نظر گرفته شده است.

به طور مشابه نتایج آزمایش برای نرم افزار انتخابی ADOL-C در شکل های ۴,۱۲ ، ۴,۱۳ ، ۴,۱۴ و ۴,۱۵ گزارش شده است. در اینجا در همه موارد ما ماتریس AA^T و A^TA (به ترتیب برای hub و autority) را برای انتخاب توابع برای بررسی شباهت آنها در نظر گرفتیم. ما نتایج را به صورت جداگانه برای آستانه = 0.4 در شکل های ۴,۱۴ و ۴,۱۵ گزارش کردیم. اکنون نتایج مربوط به نرم افزار ADOL-C را مورد بحث قرار می دهیم. شکل های ۴,۱۲ و ۴,۱۴ نشان می دهد که برای مقادیر مختلف آستانه ، تعداد سطوح متفاوتی برای توابع هاب دریافت می کنیم. باز هم شکل های ۴,۱۳ و ۴,۱۵ نشان می دهد که برای مقادیر مختلف آستانه ، تعداد سطوح متفاوتی برای توابع اقتدار بدست می آوریم.

۱ - ۳ - ۴ بحث



شکل ۴,۷: DSM از CSparse با پارتیشن های ارائه شده [۶]. رنگهای اولیه ، اصلی ، ثانویه ، ثانویه ، ثانویه ، ثالث و ثالث با رنگهای قرمز ، بنفش ، سبز ، زرد ، آبی و قهوه ای مشخص می شوند.

شکل ۴,۷ DSM است که پارتیشن ارائه شده در [۶] را نشان می دهد. در اینجا پارتیشن ها اصلی (رنگ قرمز) ، کاربرد اولیه (رنگ بنفش) ، ثانویه (رنگ سبز) ، کاربرد ثانویه (رنگ زرد) ، درجه سوم (رنگ آبی) و درجه سوم (رنگ قهوه ای) هستند. ما می بینیم که توابع اولیه ثانویه و سوم بیشتر توابع دیگر را فرا می خواند. طبق روش HITS [۱۷] این توابع مرکز مهم هستند. از سوی دیگر ما مشاهده می کنیم که خدمات آب و برق بیشتر با توابع دیگر فراخوانی می شوند و طبق روش HITS می توان آنها را به عنوان مقامات در نظر گرفت.

Hub tiers	Threshold			
	0.3		0.4	
	Name	Category	Name	Category
1	cs_schol cs_sqr cs_maxtrans	Secondary Secondary Tertiary	cs_schol cs_sqr	Secondary Secondary
2	cs_qrsol cs_cholsol cs_lusol	Primary Primary Primary	cs_qrsol	Primary
3	cs_multiply cs_add cs_lu cs_qr cs_symperm	Primary Primary Secondary Secondary Secondary	cs_multiply cs_add cs_transpose cs_symperm cs_compress	Primary Primary Primary Secondary Primary
4	cs_dmperm cs_amd cs_dropzeros cs_droptol	Secondary Secondary Secondary Secondary		
5	cs_post cs_counts cs_etree	Tertiary Tertiary Tertiary		

شکل ۴,۸: توابع CSparse، Tiers of Hub از DSM انتخاب شده اند

Authority tiers	Threshold			
	0.3		0.4	
	Name	Category	Name	Category
1	cs_malloc cs_calloc cs_sfree	Primary_Util Primary_Util Secondary_Util	cs_malloc cs_calloc	Primary_Util Primary_Util
2	cs_free cs_transpose cs_spfree cs_nfree cs_ipvec	Primary_Util Primary Primary_Util Secondary_Util Secondary	cs_free cs_nfree cs_ipvec cs_sfree	Primary_Util Secondary_Util Secondary Secondary_Util
3	cs_spalloc cs_done cs_sprealloc cs_scatter	Primary_Util Tertiary_Util Primary_Util Tertiary	cs_spalloc cs_done	Primary_Util Tertiary_Util
4	cs_fkeep cs_ddone cs_dalloc cs_scc cs_pinv	Tertiary Tertiary_Util Tertiary_Util Tertiary Secondary	cs_spfree cs_ddone cs_dalloc	Primary_Util Tertiary_Util Tertiary_Util

شکل ۴,۹: توابع CSparse، Tiers of Authority از DSM انتخاب شده اند

شکل های ۴,۸ و ۴,۱۰ نشان می دهد که برای مقادیر مختلف آستانه ، تعداد سطوح متفاوتی برای توابع هاب دریافت می کنیم. این ردیف ها همچنین با مقوله ای که در CSparse ذکر شده مطابقت دارد. به عنوان مثال در شکل ۴,۸ Hub Tier 5 شامل سه عملکرد است: cs post cs count و cs etree که در [۶] به عنوان توابع درجه سوم طبقه بندی شده اند (شکل ۴,۷). به طور مشابه ، نمودارهای ۴,۹ و ۴,۱۱ نشان می دهد که برای مقادیر مختلف آستانه ، تعداد مختلفی از سطوح توابع اقتدار را بدست می آوریم. این ردیف ها همچنین با مقوله ای که در CSparse ذکر شده مطابقت دارد. به عنوان مثال در شکل ۴,۹ مرجع درجه ۱ شامل سه عملکرد است: cs malloc cs calloc و cs sfree که در [۶] به عنوان توابع مفید طبقه بندی شده اند (شکل ۴,۷). بنابراین این روش تقریب خوبی از پارتیشن نویسندگان ایجاد می کند.

Hub tiers	Threshold									
	0.3		0.4		0.5		0.6		0.7	
	Name	Category	Name	Category	Name	Category	Name	Category	Name	Category
1	cs_schol.c cs_sqr.c cs_qrsol.c cs_maxtrans.c cs_multiply.c	Secondary Secondary Primary Tertiary Primary	cs_schol cs_sqr cs_qrsol cs_maxtrans cs_multiply	Secondary Secondary Primary Tertiary Primary	cs_schol cs_sqr cs_qrsol cs_maxtrans cs_multiply	Secondary Secondary Primary Tertiary Primary	cs_schol cs_sqr cs_qrsol cs_maxtrans	Secondary Secondary Primary Tertiary	cs_schol cs_sqr cs_maxtrans	Secondary Secondary Tertiary
2	cs_dmperm.c cs_amd.c cs_scc.c cs_counts.c cs_post.c	Secondary Secondary Tertiary Tertiary Tertiary	cs_dmperm cs_amd cs_scc cs_counts cs_post	Secondary Secondary Tertiary Tertiary Tertiary	cs_add cs_transpose cs_compress cs_symperm cs_lu	Primary Primary Primary Secondary Secondary	cs_multiply cs_add cs_transpose cs_symperm cs_compress	Primary Primary Primary Secondary Primary		
3	cs_transpose.c cs_compress.c cs_symperm.c cs_add.c cs_permute.c	Primary Primary Secondary Primary Secondary	cs_transpose cs_compress cs_symperm cs_add cs_permute	Primary Primary Secondary Primary Secondary	cs_dmperm	Secondary	cs_cholsol cs_lusol cs_dupl cs_nfree	Primary Primary Primary Secondary_Util		
4					cs_qr cs_chol cs_permute cs_load	Secondary Secondary Secondary Primary	cs_dmperm	Secondary		
5					cs_amd cs_post cs_counts cs_etree	Secondary Tertiary Tertiary Tertiary	cs_amd cs_post cs_counts cs_etree	Secondary Tertiary Tertiary Tertiary		

شکل ٤،١٠: توابع CSparse ، Tiers of Hub از AA^T انتخاب شده اند

Authority tiers	Threshold									
	0.3		0.4		0.5		0.6		0.7	
	Name	Category	Name	Category	Name	Category	Name	Category	Name	Category
1	cs_malloc cs_calloc cs_free cs_spalloc cs_sfree	Primary_Util Primary_Util Primary_Util Primary_Util Secondary_Util	cs_malloc cs_calloc cs_free cs_spalloc cs_sfree	Primary_Util Primary_Util Primary_Util Primary_Util Secondary_Util	cs_malloc cs_calloc cs_free cs_sfree	Primary_Util Primary_Util Primary_Util Secondary_Util	cs_malloc cs_calloc cs_sfree	Primary_Util Primary_Util Secondary_Util	cs_malloc cs_calloc	Primary_Util Primary_Util
2	cs_usolve cs_lu cs_nfree	Secondary Secondary Secondary_Util	cs_usolve cs_lu cs_nfree	Secondary Secondary Secondary_Util	cs_usolve cs_lu cs_nfree cs_pvec cs_ltsolve	Secondary Secondary Secondary_Util Secondary Secondary	cs_spalloc cs_done cs_scatter cs_cumsum	Primary_Util Tertiary_Util Tertiary Tertiary	cs_sfree cs_ipvec cs_nfree	Secondary_Util Secondary Secondary_Util
3	cs_done cs_cumsum cs_scatter	Tertiary_Util Tertiary Tertiary	cs_done cs_scatter	Tertiary_Util Tertiary	cs_fkeep cs_spfree cs_ddone cs_dalloc	Tertiary Primary_Util Tertiary_Util Tertiary_Util	cs_free cs_lsolve cs_ipvec cs_nfree	Primary_Util Secondary Secondary Secondary_Util		
4					cs_idone cs_tdfs	Tertiary_Util Tertiary	cs_fkeep cs_spfree cs_ddone cs_dalloc	Tertiary Primary_Util Tertiary_Util Tertiary_Util		
5							cs_sprealloc	Primary_Util		
6							cs_ndone cs_house cs_happly	Tertiary_Util Tertiary Secondary		

شکل ٤،١١: توابع CSparse، Tiers of Authority از $A^T A$ انتخاب شده اند

Hub tiers	Threshold	
	0.3	0.5
1	tape_doc jacobian forward inverse_Taylor_prop hessian	jacobian forward inverse_Taylor_prop hessian
2	operator + operator / operator * operator - operator <	tape_doc filewrite_start grow reverse hov_ti_reverse
3	inverse_tensor_eva jac_solv tensor_eval hov_ti_reverse read_params	operator + pow operator / operator * operator -
4	ADTOOL_AMPI_popGSVinfo ADTOOL_AMPI_popReduceInfo ADTOOL_AMPI_popGSinfo	filewrite_ampi filewrite filewrite_end put_op_reserve
5	openTape initNewTape filewrite_start init_rev_sweep getTapeInfos	jac_solv inverse_tensor_eva tensor_eval
6		sparse_jac bit_vector_propagation sparse_hess

شکل ۴,۱۲: توابع ADOL-C، Tiers of Hub از AA^T انتخاب شده اند

Authority tiers	Threshold	
	0.3	0.5
1	myalloc1 fprintf myfree1 adolc_exit myalloc2	myalloc1 myfree1 myalloc2
2	loc next_loc ADOLC_PUT_LOCINT put_op ADOLC_PUT_VAL	adolc_exit malloc
3	free malloc MINDEC	loc next_loc ADOLC_PUT_LOCINT put_op ADOLC_PUT_VAL
4	TAPE_AMPI_read_MPI_Comm allocatePack TAPE_AMPI_read_MPI_Datatype TAPE_AMPI_read_int unpackDeallocate	
5	myfroo2 myfree3 myalloc3 spread1 hos_forward	
6	fail begin cnd empty	

شکل ۴,۱۳: ADOL-C ، توابع Tiers of Authority از $A^T A$ انتخاب شده اند

For Threshold = 0.4			
Hub tiers	Functions	Hub tiers	Functions
1	jacobian forward inverse_Taylor_prop hessian	7	init_rev_sweep init_for_sweep put_op_reserve get_op_block_f
2	operator + operator / operator * operator - operator <	8	reg_ext_fct reg_timestep_fct
3	inverse_tensor_eva jac_solv tensor_eval hov_ti_reverse read_params	9	cp_fov_reverse cp_fos_reverse function vec_jac gradient
4	tape_doc ADOLC_TLM_AMPI_Bcast ADOLC_TLM_AMPI_Allgatherv ADOLC_TLM_AMPI_Scatterv ADOLC_TLM_AMPI_Gatherv	10	readConfigFile close_tape filewrite_end cleanUp taylor_close
5	ADTOOL_AMPI_popGSVinfo ADTOOL_AMPI_popReduceInfo ADTOOL_AMPI_popGSinfo	11	lie_covector lie_gradientcv lie_bracket reverse large_jacobian
6	openTape initNewTape filewrite_start getTapeInfos	12	grow free_loc ADTOOL_AMPI_copyActiveBuf ensure_block ADTOOL_AMPI_allocateTempBuf

شکل ۴,۱۴: ADOL-C ، توابع Hub از AA^T ، آستانه =0.4 انتخاب شده اند

For Threshold = 0.4					
Authority tiers	Functions	Authority tiers	Functions	Authority tiers	Functions
1	myalloc1 fprintf myfree1 adolc_exit myalloc2	6	begin end empty	11	fclose strlen strcmp clearTapeBaseNames strncpy
2	loc next_loc ADOLC_PUT_LOCINT put_op ADOLC_PUT_VAL	7	fail fread fseek fopen	12	BW_AMPI_Barrier BW_AMPI_Wait BW_AMPI_Recv BW_AMPI_Send
3	free malloc MINDEC	8	cp_hov_forward; cp_fov_forward cp_hos_forward dummy edfoo_iarr_wrapper_fov_reverse	13	assert emplace_front erase_after before_begin
4	TAPE_AMPI_read_MPI_Datatype TAPE_AMPI_read_int TAPE_AMPI_read_MPI_Comm allocatePack unpackDeallocate	9	spread1 pack1 zos_forward pack2	14	edfoo_iarr_wrapper_zos_forward edfoo_iarr_wrapper_function edfoo_iarr_wrapper_fov_forward edfoo_wrapper_fov_reverse edfoo_wrapper_fos_reverse
5	myfree2 myfree3 myalloc3 hos_forward	10	cp_hov_reverse cp_hos_reverse cp_fos_forward edfoo_iarr_wrapper_fos_reverse edfoo_iarr_wrapper_fos_forward		

شکل ۴,۱۵: ADOL-C ، توابع Authority از $A^T A$ ، آستانه =0.4 انتخاب شده اند

فصــل پنجم

جمع بندی

در این پایان نامه ، ما روشی را برای گروه بندی یا دسته بندی عناصر طراحی نرم افزارهای علمی در "ردیف" بر اساس "اهمیت" آنها ارائه کرده ایم. ابتدا ساختار وابستگی نرم افزار را با استفاده از ابزار "Understand" تجزیه و تحلیل کرده و DSM ساخته ایم. DSM نمودار فراخوانی ابزار مناسبی را ارائه می دهد تا بتوان از تکنیک های جبری خطی برای شناسایی تماس فرستندگان و تماس گیرندگان مهم از طریق محاسبه نمایی ماتریس استفاده کرد. با استفاده از مفهوم "اهمیت" عناصر طراحی [۱۳] ، آن عناصر طراحی را رتبه بندی کردیم. سپس ما "شباهت" را در بین عناصر طراحی کشف کردیم. در نهایت با استفاده از "شباهت" بین عناصر طراحی ، آنها را در ردیف های مختلف گروه بندی می کنیم. علاوه بر استفاده از DSM ، ما همچنین از ماتریس $B1 = AA^T$ و $B2 = A^T A$ برای انتخاب توابعی که بین توابع وابستگی معنایی دارند ، استفاده کردیم. ما الگوریتم خود را بر روی نرم افزار CSparse و ADOL-C که در C اجرا شده است اعمال کردیم.

مدیریت دانش بر پیچیدگی سازمانی و حوزه های آن تاثیر دارد. اجزای مدیریت دانش (کسب دانش و ذخیره دانش، انتقال و کاربرد دانش) بر پیچیدگی سازمانی تاثیر معناداری دارند. پس می توان گفت مدیریت دانش هم یکی از ابزارهای کاهش و مدیریت پیچیدگی محسوب می شود.

یکی از دلایل اصلی که برنامه های توسعه نرم افزار در نهایت با شکست مواجه میشوند، تخمین ضعیف اندازه نرم افزار است . اندازه عامل مهمی در تعیین هزینه ، زمان و تلاش است . تخمین حجم ، یک فعالیت پیچیده است ، که نتایج آن باید به طور مداوم در سراسر چرخه حیات نرم افزار به روز شده باشد. اقدامات تشخیص حجم نرم افزار شامل تعداد خط کد منبع ، تعداد نقاط تابع و تعداد نقاط ویژه می باشد. پیچیدگی، تابعی از اندازه است ، که تا حد زیادی تحت تاثیر خطاهای طراحی و نقص پنهان هستند که منجر به مشکلات کیفیت و افزایش هزینه ها خواهد شد. پیچیدگی را باید به طور مداوم اندازه گیری، ردیابی و کنترل کرد. معیارهای اندازه گیری نرم افزار جنبه های مختلفی از پیشرفت و پیچیدگی نرم افزار را دارد در نتیجه نقش مهمی در تجزیه و تحلیل و بهبود کیفیت نرم افزارخواهد داشت . معیارهای نرم افزار اغلب به معیارهای محصولات و معیارهای فرآیند طبقه بندی شده اند.

با استفاده از الگوریتم ما می توانیم همه عملکردهای مهم سیستم را دسته بندی کنیم. این دسته به کاربر سیستم کمک می کند تا عملکردهای قابل استفاده (هاب) خود را شناسایی کند. یک سیستم خوب باید مرتباً به روز شود. اما به دلایل مختلف ممکن است نرم افزارهای قدیمی حاوی اسناد فنی کافی نباشند ، بنابراین از دیدگاه قابلیت استفاده ، تشخیص و بازیابی اجزایی که در پروژه های دیگر نرم افزاری مورد استفاده مجدد قرار می گیرند ، دشوار است. بنابراین با استفاده از رویکرد ما ، توسعه دهنده یک سیستم می تواند توابع مهم قدرت را که توسط مراکز مهم نامیده می شوند ، شناسایی کند. این یافته نشان می دهد که تجزیه و تحلیل

سطوح مختلف عملکردهای یک سیستم نرم افزاری ممکن است راهنمای توسعه دهندگان در کار چالش برانگیز طراحی مجدد نرم افزار با تشخیص و بازیابی اجزایی باشد که می توانند در پروژه های نرم افزاری دیگر مورد استفاده مجدد قرار گیرند.

از نظر تحقیقات بیشتر ، ما می خواهیم مشکلات آزمایشی شبکه های بزرگتر را شامل شود. جالب است که مشکلات حوزه های مختلف علمی مانند SNAP و DIMACS10 را برای شناسایی گروه های گره های مهم ذکر کنیم. در این جا ما ماتریس های بدون وزن را در نظر گرفته ایم زیرا برای شناسایی شباهت فقط به اطلاعات مربوط به وابستگی نیاز داریم. برای تحقیقات بیشتر می توان الگوریتم خود را برای ماتریس های وزنی و بدون وزن مقایسه کرد. همچنین در آینده ما می خواهیم روش خود را در کتابخانه نرم افزار کد قدیمی اعمال کنیم که در آن اسناد بسیار کمی وجود دارد یا هیچ گونه اسناد و مدارکی در مورد پروژه در دسترس نیست. روشی که ما در این پایان نامه توسعه دادیم احتمالاً برای این نوع کتابخانه ها مفید خواهد بود.

مراجع

[1] Lazima Ansari, Shahadat Hossain, and Ahamad Imtiaz Khan. DSMDE: A data exchange

format for design structure models. Sustainability in Modern Project Management: Proceedings of the 18th International DSM Conference, pages 111–121, 2016.

[2] Len Bass, Paul Clements, and Rick Kazman. Software Architecture in Practice, 2e.

Addison Wesley, 2003.

[3] Michele Benzi, Ernesto Estrada, and Christine Klymko. Ranking hubs and authorities

using matrix functions. Linear Algebra and its Applications, 438(5):2447–2474, 2013.

[4] Dan Braha and Yaneer Bar-Yam. The statistical mechanics of complex product development:

Empirical and analytical results. Management Science, 53(7):1127–1145, 2007.

[5] Jonathan J Crofts and Desmond J Higham. A weighted communicability measure applied to complex brain networks. Journal of the Royal Society, Interface, 6(33):411–414, 2009.

[6] Timothy A. Davis. Direct methods for sparse linear systems (fundamentals of algorithms

2). SIAM, 2006.

[7] Steven D Eppinger and Tyson R Browning. Design structure matrix methods and

applications. MIT press, 2012.

[8] John Forrest, Ted Ralphs, Stefan Vigerske, Lou Hafer, Bjarni Kristjansson, jpfasano,

Edwin Straver, Miles Lubin, Gambini Santos, rlougee, and Matthew Saltzman. coinor/cbc: Version 2.9.9, July 2018.

[9] Linton C Freeman. conceptual clarification.” social networks. “Centrality in social networks, 1(3):215–239, 1978.

[10] Gene H Golub and Charles F Van Loan. Matrix computations. JHU Press, 3, 2012.

[11] Marco A. Gonzalez. A new change propagation metric to assess software evolvability. PhD thesis, University of British Columbia, 2013.

[12] Andreas Griewank, David Juedes, and Jean Utke. ADOL-C: a package for the automatic differentiation of algorithms written in c/c++. ACM Transactions on Mathematical Software, 1996.

[13] S Hossain, SF Khan, and R Quashem. On ranking components in scientific software. In DSM 2015: Modeling and managing complex systems-Proceedings of the 17th International DSM Conference Fort Worth (Texas, USA), 4-6 November 2015, pages 245–254, 2015.

[14] Shahadat Hossain et al. Efficiently computing with design structure matrices. In DSM 2010: Proceedings of the 12th International DSM Conference, Cambridge, UK, 22.-23.07. 2010, pages 345–358, 2010.

[15] Shahadat Hossain and Ahmed Tahsin Zulkarnine. Design structure of scientific software–a case study. In DSM 2011: Proceedings of the 13th International DSM Conference, pages 129–141, 2011.

[16] D. Kelly and R. Sanders. Assessing the quality of scientific software. in Proc of the First International Workshop on Software Engineering for Computational Science and Engineering, 2008.

[17] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. Journal of the ACM (JACM), 46(5):604–632, 1999.

[18] Anany V Levitin. Introduction to design & analysis of algorithms: For anna university, 2e. Pearson Education India, 2009.

[19] Alan MacCormack, John Rusnak, and Carliss Y Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. Management Science, 52(7):1015–1030, 2006.

[20] M. W. Maier, D. Emery, and R. Hilliard. Software architecture: introducing ieee standard 1471. Computer, 34(4):107–109, 2001.

[21] Scientific Toolworks Inc. Scitools: Understand. https://scitools.com/.

[22] Alexandru Telea, Hessel Hoogendorp, Ozan Ersoy, and Dennie Reniers. Extraction and visualization of call dependencies for large C/C++ code bases: A comparative study. In Proceedings of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2009, Edmonton, Alberta, Canada, September 25, 2009, pages 81–88. IEEE Computer Society, 2009.

پیوست اول

لیست توابع CSparse

Node Number	Function Name	Node Number	Function Name	Node Number	Function Name
1	cs_add	23	cs_lsolve	45	cs_symperm
2	cs_amd	24	cs_ltsolve	46	cs_tdfs
3	cs_chol	25	cs_lu	47	cs_transpose
4	cs_cholsol	26	cs_lusol	48	cs_updown
5	cs_compress	27	cs_malloc	49	cs_usolve
6	cs_counts	28	cs_maxtrans	50	cs_utsolve
7	cs_cumsum	29	cs_multiply	51	cs_calloc
8	cs_dfs	30	cs_norm	52	cs_done
9	cs_dmperm	31	cs_permute	53	cs_spalloc
10	cs_droptol	32	cs_pinv	54	cs_sprealloc
11	cs_dropzeros	33	cs_post	55	cs_spfree
12	cs_dupl	34	cs_print	56	cs_ndone
13	cs_entry	35	cs_pvec	57	cs_free
14	cs_ereach	36	cs_qr	58	cs_idone
15	cs_etree	37	cs_qrsol	59	cs_dalloc
16	cs_fkeep	38	cs_randperm	60	cs_ddone
17	cs_gaxpy	39	cs_reach	61	cs_dfree
18	cs_happly	40	cs_scatter	62	cs_nfree
19	cs_house	41	cs_scc	63	cs_sfree
20	cs_ipvec	42	cs_schol	64	cs_realloc
21	cs_leaf	43	cs_spsolve		
22	cs_load	44	cs_sqr		

شکل A.1: فهرست توابع CSparse

ضمیمه B

لیست توابع ADOL-C

No	Name	No	Name	No	Name	No	Name	No	Name
1	function	31	tensor_eval	61	myalloc2	91	operator >>	121	log10
2	gradient	32	filewrite_start	62	myalloc3	92	operator++	122	sinh
3	vec_jac	33	filewrite	63	myfree3	93	operator--	123	tanh
4	jacobian	34	filewrite_ampi	64	myfree1	94	operator +=	124	ceil
5	large_jacobian	35	filewrite_end	65	myfree2	95	operator -=	125	floor
6	jac_vec	36	tape_doc	66	myfreeI2	96	operator *=	126	asinh
7	hess_vec	37	lie_scalarcv	67	myallocI2	97	operator /=	127	acosh
8	hess_mat	38	lie_scalarc	68	myalloc1_uint	98	operator !=	128	atanh
9	hessian	39	lie_gradientcv	69	myalloc1_ulong	99	operator ==	129	erf
10	hessian2	40	lie_gradientc	70	myalloc2_ulong	100	operator <=	130	fabs
11	lagra_hess_vec	41	lie_covector	71	myfree1_uint	101	operator >=	131	fmin
12	forodec	42	lie_bracket	72	myfree1_ulong	102	operator >	132	fmax
13	forodec_	43	lie_scalar	73	myfree2_ulong	103	operator <	133	ldexp
14	accodec_	44	lie_gradient	74	condassign	104	operator +	134	frexp
15	abs_normal	45	jac_pat	75	condeqassign	105	operator -	135	myquad
16	directional_active_gradient	46	absnormal_jac_pat	76	initInternal	106	operator *	136	operator >>
17	abs_normal_	47	generate_seed_jac	77	adouble	107	operator /	137	adubref
18	freecoefflist	48	hess_pat	78	~adub	108	recipr	138	adub
19	tensorpoint	49	generate_seed_hess	79	adubp_from_adub	109	exp	139	blocker
20	tensorsetup	50	deepcopy_HP	80	getValue	110	log	140	nondecreasing
21	freetensorpoint	51	sparse_jac	81	operator double const&	111	sqrt	141	operator[]
22	freetensor	52	sparse_hess	82	operator double&&	112	cbrt	142	lookupindex
23	summand	53	set_HP	83	operator double	113	sin	143	adolc_vec_copy
24	coeff	54	get_HP	84	setValue	114	cos	144	adolc_vec_dot
25	tensor_address	55	bit_vector_propagation	85	operator =	115	tan	145	adolc_vec_axpy
26	LUFactorization	56	freeSparseHessInfos	86	declareIndependen	116	asin	146	deallocate
27	GauszSolve	57	ADOLC_get_sparse_Jacobian	87	operator <<=	117	acos	147	unpackDeallocate
28	jac_solv	58	forward	88	operator >>=	118	atan	148	ADOLC_TLM_Init
29	inverse_Taylor_prop	59	reverse	89	declareDependent	119	atan2	149	ADOLC_TLM_AMPI_Send
30	inverse_tensor_eva	60	myalloc1	90	operator <<	120	pow	150	ADOLC_TLM_AMPI_Recv

شکل B.1: فهرست توابع ADOL-C

No	Name	No	Name	No	Name	No	Name	No	Name
151	ADOLC_TLM_AMPI_Isend	161	ADTOOL_AMPI_pop_AMPI_Request	211	Buffer	241	call_ext_fct_commonPrior	271	cos_forward
152	ADOLC_TLM_AMPI_Irecv	162	ADTOOL_AMPI_push_request	212	SubBuffer	242	call_ext_fct_commonPost	272	fov_forward
153	ADOLC_TLM_AMPI_Wait	163	ADTOOL_AMPI_push_comm	213	zeroAll	243	get_ext_diff_fct	273	fov_forward
154	ADOLC_TLM_AMPI_Barrier	164	ADTOOL_AMPI_packDType	214	append	244	edfoo_wrapper_function	274	fos_fi_reverse
155	ADOLC_TLM_AMPI_Gather	165	ADTOOL_AMPI_unpackDType	215	getElement	245	edfoo_wrapper_fov_forward	275	fos_reverse
156	ADOLC_TLM_AMPI_Scatter	166	ADTOOL_AMPI_getAdjointCount	216	reg_timestep_fct	246	edfoo_wrapper_zos_forward	276	fov_reverse
157	ADOLC_TLM_AMPI_Allgather	167	ADTOOL_AMPI_setAdjointCountAndTempBuf	217	checkpointing	247	edfoo_wrapper_fos_forward	277	callHandleReverse
158	ADOLC_TLM_AMPI_Gatherv	168	ADTOOL_AMPI_allocateTempBuf	218	get_cp_fct	248	edfoo_wrapper_fos_reverse	278	callHandleForward
159	ADOLC_TLM_AMPI_Scatterv	169	ADTOOL_AMPI_releaseAdjointTempBuf	219	init_edf	249	edfoo_wrapper_fov_reverse	279	callHandlePrimal
160	ADOLC_TLM_AMPI_Allgatherv	170	ADTOOL_AMPI_allocateTempActiveBuf	220	cp_zos_forward	250	edfoo_tarf_wrapper_function	280	initTape
161	ADOLC_TLM_AMPI_Reduce	171	ADTOOL_AMPI_copyActiveBuf	221	revolve_for	251	edfoo_tarf_wrapper_fov_forward	281	freeTape
162	ADOLC_TLM_AMPI_Allreduce	172	ADTOOL_AMPI_setupTypes	222	cp_fos_reverse	252	edfoo_tarf_wrapper_zos_forward	282	mediAddHandle
163	ADOLC_TLM_AMPI_Bcast	173	ADTOOL_AMPI_cleanupTypes	223	cp_fov_reverse	253	edfoo_tarf_wrapper_fos_forward	283	mediInitTape
164	AMPI_Init_NT	174	MPI_Datatype ADTOOL_AMPI_FW_rawType	224	cp_clearStack	254	edfoo_tarf_wrapper_fos_reverse	284	mediInitStatic
165	ADTOOL_AMPI_pushBcastInfo	175	MPI_Datatype ADTOOL_AMPI_BW_rawType	225	cp_takeshot	255	edfoo_tarf_wrapper_fov_reverse	285	addHandle
166	ADTOOL_AMPI_popBcastInfo	176	AMPI_Send	226	cp_restore	256	iteration	286	pdouble
167	ADTOOL_AMPI_pushDoubleArray	177	AMPI_Recv	227	cp_taping	257	fp_zos_forward	287	mkparam
168	ADTOOL_AMPI_popDoubleArray	178	AMPI_Isend	228	cp_release	258	fp_fos_forward	288	mkparam_idx
169	ADTOOL_AMPI_pushReduceInfo	179	AMPI_Irecv	229	revolveError	259	fp_fos_reverse	289	adjust
170	ADTOOL_AMPI_popReduceCountAndType	180	AMPI_Wait	230	edf_zero	260	fp_iteration	290	revolve
171	ADTOOL_AMPI_popReduceInfo	181	AMPI_Barrier	231	reg_ext_fct	261	zos_forward_partx	291	rpl_malloc
172	ADTOOL_AMPI_pushSRinfo	182	AMPI_Gather	232	update_ext_fct_memory	262	fos_forward_partx	292	rpl_calloc
173	ADTOOL_AMPI_popSRinfo	183	AMPI_Scatter	233	call_ext_fct	263	fos_forward_partx	293	rpl_realloc
174	ADTOOL_AMPI_pushGSinfo	184	AMPI_Allgather	234	get_ext_diff_fct_v2	264	fov_forward_partx	294	GlobalTapeVarsCL
175	ADTOOL_AMPI_popGScommSizeForRootOrNull	185	AMPI_Gatherv	235	edfoo_v2_wrapper_function	265	fov_forward_partx	295	free_loc
176	ADTOOL_AMPI_popGSinfo	186	AMPI_Scatterv	236	edfoo_v2_wrapper_zos_forward	266	fos_reverse	296	next_loc
177	ADTOOL_AMPI_pushGSVinfo	187	AMPI_Allgatherv	237	edfoo_v2_wrapper_fos_forward	267	fov_reverse	297	ensure_block
178	ADTOOL_AMPI_popGSVinfo	188	AMPI_Bcast	238	edfoo_v2_wrapper_fov_forward	268	fov_fi_reverse	298	grow
179	ADTOOL_AMPI_push_CallCodeReserve	189	AMPI_Reduce	239	edfoo_v2_wrapper_fos_reverse	269	init_reverse_safe	299	initTapeInfos_keep
180	ADTOOL_AMPI_push_AMPI_Request	190	AMPI_Allreduce	240	edfoo_v2_wrapper_fov_reverse	270	fos_forward	300	initNewTape

شکل B.2: فهرست توابع ADOL-C

No	Name	No	Name	No	Name	No	Name	No	Name
301	openTape	331	setStoreManagerType	361	set_param_vec	391	spread1	421	get_val_block
302	getTapeInfos	332	reallocStore	362	read_tape_stats	392	pack1	422	TAPE_AMPI_read_int
303	releaseTape	333	fail	363	skip_tracefile_cleanup	393	MINDEC	423	TAPE_AMPI_read_MPI_Datatype
304	set_nested_ctx	334	printError	364	init_for_sweep	394	pack2	424	TAPE_AMPI_read_MPI_Request
305	currently_nested	335	clearTapeBaseNames	365	init_rev_sweep	395	fov_offset_forward	425	accodeout
306	cachedTraceTags	336	createFileName	366	end_sweep	396	fos_forward	426	acccov
307	setTapeInfoJacSparse	337	duplicatestr	367	put_op_reserve	397	hov_wk_forward	427	accadj
308	setTapeInfoHessSparse	338	readConfigFile	368	get_op_block_f	398	free	428	accbrac
309	init_lib	339	take_stock	369	put_loc_block	399	myfree	429	indopro_forward_tight
310	clearCurrentTape	340	keep_stock	370	put_vals_writeBlock	400	spread3	430	indopro_forward_safe
311	cleanUp	341	taylor_begin	371	put_val_block	401	accodec	431	indopro_forward_absnormal
312	removeTape	342	taylor_close	372	get_val_block_f	402	pack3	432	nonl_ind_old_forward_tight
313	trace_on	343	taylor_back	373	get_val_space	403	fprintf	433	nonl_ind_old_forward_safe
314	trace_off	344	write_taylor	374	discard_params_r	404	zos_pl_forward	434	nonl_ind_forward_tight
315	checkInitialStoreSize	345	write_taylors	375	reset_val_r	405	fos_pl_reverse	435	nonl_ind_forward_safe
316	Keeper	346	write_scaylors	376	get_op_f	406	memset	436	calloc
317	initADOLC	347	put_tay_block	377	get_op_r	407	fov_pl_forward	437	int_reverse_tight
318	beginParallel	348	get_taylors	378	get_locint_f	408	fos_pl_sig_reverse	438	int_reverse_safe
319	endParallel	349	get_taylors_p	379	get_locint_r	409	malloc	439	int_forward_tight
320	TapeInfos	350	get_tay_block_r	380	get_val_f	410	dbinomi	440	int_forward_safe
321	copy	351	initTapeBuffers	381	get_num_switches	411	binomi	441	freeSparseJacInfos
322	PersistantTapeInfos	352	start_trace	382	copy_index_domain	412	convert	442	populate_dpp
323	StoreManagerLocintBlock	353	save_params	383	merge_2_index_domains	413	multma2vec1	443	populate_dppp
324	ensureContiguousLocations	354	stop_trace	384	combine_2_index_domains	414	multma2vec2	444	loc
325	setStoreManagerControl	355	close_tape	385	merge_3_index_domains	415	strncpy	445	upd_resloc_check
326	consolidateBlocks	356	freeTapeResources	386	free_tree	416	checkPageBreak	446	ADOLC_PUT_LOCINT
327	enableMinMaxUsingAbs	357	tapestats	387	traverse_crs	417	fflush	447	put_op
328	disableMinMaxUsingAbs	358	printTapeStats	388	traverse_unary	418	fclose	448	upd_resloc
329	adolc_exit	359	get_num_param	389	extend_nonlinearity_domain_binary_ste	419	get_op_block	449	upd_resloc_inc_prod
330	free_all_taping_params	360	read_params	390	extend_nonlinearity_domain_unary	420	get_loc_block_f	450	ADOLC_PUT_VAL

شکل B.3: فهرست توابع ADOL-C

No	Name	No	Name	No	Name	No	Name	No	Name
451	value	484	TAPE_AMPI_push_double	517	FW_AMPI_Gatherv	550	BW_AMPI_Gather	583	emplace_front
452	ADOLC_init_sparse_pattern	485	TAPE_AMPI_pop_double	518	FW_AMPI_Scatterv	551	BW_AMPI_Scatter	584	clear
453	delete_pattern	486	TAPE_AMPI_push_MPI_Op	519	FW_AMPI_Allgatherv	552	BW_AMPI_Allgather	585	pop_front
454	push_back	487	TAPE_AMPI_pop_MPI_Comm	520	FW_AMPI_Bcast	553	BW_AMPI_Gatherv	586	next
455	get_pattern_size	488	TAPE_AMPI_pop_MPI_Op	521	FWB_AMPI_Reduce	554	BW_AMPI_Scatterv	587	cend
456	get_pattern	489	ADTOOL_AMPI_pop_CallCode	522	FW_AMPI_Allreduce	555	BW_AMPI_Allgatherv	588	cbegin
457	begin	490	assert	523	function_double	556	BW_AMPI_Bcast	589	front
458	end	491	TAPE_AMPI_push_MPI_Request	524	ADOLC_WRITE_SCAYLOR	557	BWB_AMPI_Reduce	590	gcTriggerRatio
459	trunc	492	TAPE_AMPI_pop_MPI_ReqDest	525	dummy	558	BW_AMPI_Allreduce	591	before_begin
460	size	493	MPI_Request ADTOOL_AMPI_pop_request	526	zp_fos_forward	559	spread2	592	erase_after
461	dubref	494	MPI_Comm ADTOOL_AMPI_pop_comm	527	zp_fov_forward	560	getTapeVector	593	sort
462	MPI_Op_create	495	ADTOOL_AMPI_isActiveType	528	zp_hos_forward	561	funcReverse	594	FatalError
463	TAPE_AMPI_read_MPI_Comm	496	memcpy	529	zp_hov_forward;	562	funcForward	595	StoreManagerLocint
464	allocatePack	497	isDerivedType	530	zp_hos_reverse	563	funcPrimal	596	strlen
465	TLM_AMPI_Isend	498	derivedTypeIdx	531	zp_hov_reverse	564	HandleVector	597	sprintf
466	TLM_AMPI_Irecv	499	ADTOOL_AMPI_setAdjointCount	532	evolve_for	565	clearHandles	598	strchr
467	TLM_AMPI_Wait	500	MPI_Abort	533	evolve	566	AdolcHeapStatic	599	strtoul
468	TLM_AMPI_Barrier	501	ADTOOL_AMPI_isActiveTyp	534	ADOLC_GET_TAYLOR	567	max	600	strcmp
469	TLM_AMPI_Gather	502	getDTypeData	535	empty	568	maxrange	601	stat
470	TLM_AMPI_Scatter	503	MPI_Type_contiguous	536	top	569	numforw	602	put_vals_notWriteBlock
471	TLM_AMPI_Allgather	504	MPI_Type_commit	537	pop	570	realloc	603	fseek
472	TLM_AMPI_Gatherv	505	MPI_Type_contiguous	538	saveNonAdoubles	571	initTapeInfos	604	fread
473	TLM_AMPI_Scatterv	506	MPI_Type_commit	539	restoreNonAdoubles	572	rewind	605	fopen
474	TLM_AMPI_Allgatherv	507	MPI_Type_free	540	populate_dppp_nodata	573	push	606	fwrite
475	TLM_AMPI_Allreduce	508	FW_AMPI_Send	541	function_lAm	574	resize	607	get_tay_block
476	TLM_AMPI_Bcast	509	FW_AMPI_Recv	542	get_ext_diff_fct	575	init	608	markNewTape
477	MPI_Init	510	FW_AMPI_Isend	543	back	576	pop_back	609	put_op_block
478	TAPE_AMPI_push_int	511	FW_AMPI_Irecv	544	BW_AMPI_Send	577	remove	610	MIN_ADOLC
479	TAPE_AMPI_push_MPI_Datatype	512	FW_AMPI_Wait	545	BW_AMPI_Recv	578	erase	611	get_val_block_r
480	TAPE_AMPI_push_MPI_Comm	513	FW_AMPI_Barrier	546	BW_AMPI_Isend	579	flush	612	extend_nonlinearity_domain_binary_step
481	APE_AMPI_pop_MPI_Comm	514	FW_AMPI_Gather	547	BW_AMPI_Irecv	580	touch		
482	TAPE_AMPI_pop_int	515	FW_AMPI_Scatter	548	BW_AMPI_Wait	581	omp_get_thread_num		
483	TAPE_AMPI_pop_MPI_Datatype	516	FW_AMPI_Allgather	549	BW_AMPI_Barrier	582	omp_get_num_threads		

شکل B.4: فهرست توابع ADOL-C

واژه‌نامه فارسی به انگلیسی

ارزیابی الگو	Pattern Evaluation
استخراج	Extraction
انتخاب داده‌ها	Data Selection
پیش‌بینی	Prediction
تبدیل داده‌ها	Data Transformation
تشخیص	Diagnosis
سازمان	Organization
خوشه‌بندی	Clustering
دقت	Accuracy
روابط	Relationships
ماتریس	matrix
محاسبات	Calculations
معماری	Architecture
وابستگی	Dependence

واژه‌نامه انگلیسی به فارسی

Accuracy	دقت
Architecture	معماری
Calculations	محاسبات
Classification	طبقه‌بندی
Data Itegratin	یکپارچه‌سازی داده‌ها
Data Mining Techniques	تکنیک‌های داده‌کاوی
Data Selection	انتخاب داده‌ها
Data Transformation	تبدیل داده‌ها
Dependence	وابستگی
Diagnosis	تشخیص
Error	خطا
Extraction	استخراج
Knowledge Presentation	ارائه دانش
Matrix	ماتریس
Organization	سازمان
Pattern Evaluation	ارزیابی الگو
Prediction	پیش‌بینی
Relationships	روابط

Abstract

One of the expected benefits of a modular design is flexibility. By the word “flexibility” we mean possibility of drastic changes to a module without changing or without knowing other modules. Based on the evolutionary data available on version control systems, it should be possible to analyze the quality of a modular software architecture and decide whether it is worth to restructure its design. In this thesis we investigate this issue using a novel approach based on a general theory of modularity that uses design structure matrices (DSM) for reasoning about quality attributes. Using our approach, we can categorize the functions in different tiers. This finding suggests that the analysis of different tiers of functions of a software system might serve as guidance to developers in the challenging task of redesigning a software by detecting and retrieving components that could be reused in other software projects.

Keywords

Scientific computing software, software complexity, relationship and dependency, design structure matrix.

Payam Noor University

Faculty of Engineering

Seminar Report

Department of Computer Engineering and Information Technology

MANAGING COMPLEXITY IN SCIENTIFIC SOFTWARE

Bahareh Mirzakhani

Supervisor:

Dr. Razavi Ebrahimi

September 2021