

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 1400

**BISM2**

**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](#_Toc413125201) صفحه

**فصل اول: معرفی**

[۱-۱ مقدمه](#_Toc504300857) 3

[۱-1-۱ نرم افزار محاسبات علمی](#_Toc504300857) 3

[2-۱-1 معماری نرم افزار](#_Toc504300857) 4

[3-۱-1 رابطه وابستگی](#_Toc504300857) 5

[4-۱-1 معیارهای نرم افزار](#_Toc504300857) 5

[5-۱-1 معیار اندازه نرم افزار](#_Toc504300857) 6

[6-۱-1 معیارهای علمی نرم افزار](#_Toc504300857) 7

[7-۱-1 معیارهای کنترل جریان](#_Toc504300857) 8

[2-1 مشارکت ما](#_Toc504300858) 9

[3 – 1 پیچیدگی در نرم افزار](#_Toc504300859) 9

[4 – 1 سازمان پایان نامه](#_Toc504300859) 10

[فصـل دوم:](#_Toc504300864) استخراج و مدل سازی وابستگی

[1 – 2 طراحی ساختار ماتریس](#_Toc504300867) 12

[۲ – ۲ استخراج وابستگی](#_Toc504300868) 13

[3 – 2 روابط ، ماتریس ها و نمودارها](#_Toc504300869) 16

[1 – ۳ – 2 روابط](#_Toc504300870) 17

[۲ – ۳ – ۲ نمودارها](#_Toc504300871) 17

[3 – ۳ – 2 ماتریس ها](#_Toc504300872) 19

[فصـل سـوم](#_Toc504300878): تجزیه و تحلیل وابستگی

[1- 3 ارزش های ویژه و بردارهای ویژه](#_Toc504300881) 30

[2- 3 مرکز و اقتدار](#_Toc504300882) 33

[۳- ۳ جستجوی موضوعی ناشی از ابرمتن (HTS)](#_Toc504300887) 33

[فصـل چهارم](#_Toc504300889): روش و نتایج

[1 – 4روش شناسی](#_Toc504300892) 39

[1-1-4 استخراج وابستگی ها](#_Toc504300893) 39

[2 – ۱ – 4 ساخت DSM ها](#_Toc504300894) 41

[3 – ۱ – 4 مرکز محاسبات و رتبه بندی قدرت](#_Toc504300895) 43

[۴ – ۱ – ۴ شباهت های محاسبه کسینوس](#_Toc504300896) 46

[5 – ۱ – 4 الگوریتم](#_Toc504300897) 48

[2 – 4 تنظیمات](#_Toc504300904) 50

[1 – ۲ – 4 سیستم های هدف](#_Toc504300905) 50

[2 – ۲ – 4 انتخاب آستانه](#_Toc504300906) 51

[3– 4 نتایج](#_Toc504300908) 51

[1 – 3 – 4 بحث](#_Toc504300905) 52

[فصـل پنجم](#_Toc504300909): [جمع‌بندی](#_Toc504300910)

[مقدمه](#_Toc504300911) 60

[مراجع](#_Toc504300915) 63

[پیوست](#_Toc504300915) 66

**فهرست اشکال**

[عنوان](#_Toc413125201) صفحه

|  |  |
| --- | --- |
| تصویر 1 - 2 : نمونه ای از DSM: الف) نمودار وابستگی، ب) ماتریس وابستگی  تصویر ۲ - ۲ : رابطه تماس ها  تصویر 3-2 : وابستگی تابع  تصویر 4-2 : شرح عملکرد  تصویر 5-2 : یک مثال شبه برای وابستگی به تابع  تصویر 6-2 :نمونه ای از نمودار : الف)گراف غیرمستقیم، ب) نمودار جهت دار | 13  15  15  15  16  18 |
| تصویر 7 - 2 : نمونه ای نمودار باارزش  تصویر 8 -۲ : نمونه ای از DSM  تصویر 1-3 : یک مثال: الف) نمودار، ب)ماتریس مجاور معادل یا DSM  تصویر 1-4 : ماتریس پراکنده CS parse  تصویر 2-4 : ماتریس پراکنده ADOL-C  تصویر 3-4 :نمودار وابستگی CS parse  تصویر 4-4 :نمودار وابستگی ADOL-C  تصویر 5-4 :DSM از CS parse  تصویر 6-4 :نمونه ای از Hub، Authority  تصویر 7-4 :DSM از CS parse با پارتیشن های ارائه شده  تصویر 8-4 :توابع CS parse ،Tiers of Hub از DSM انتخاب شده اند  تصویر 9-4 :توابع CS parse ،Tiers of Authority از DSM انتخاب شده اند  تصویر 10-4 :توابع CS parse ،Tiers of Hub از AAT انتخاب شده اند  تصویر 11-4 :توابع CS parse ،Tiers of Authority از ATA انتخاب شده اند  تصویر 12-4 :توابع ADOL-C ،Tiers of Hub از AAT انتخاب شده اند  تصویر 13-4 :توابع ADOL-C ،Tiers of Authority از ATA انتخاب شده اند  تصویر 14-4 : ADOL-C ، توابع Hub از AAT، آستانه=0.4 انتخاب شده اند  تصویر 15-4 : ADOL-C ، توابع Authorityاز ATA، آستانه=0.4 انتخاب شده اند | 18  20  31  40  40  42  43  44  45  52  53  54  55  56  57  58  59  59 |

**فهرست جداول**

[عنوان](#_Toc413125201) صفحه

|  |  |
| --- | --- |
| جدول 1-3 : Authority Ranking Hub and نمرات مربوط به بردار ویژه غالب است  جدول 1-4: Authority Hub and رتبه پنج عملکرد اول پروژه CS Parse  جدول 2-4 : Authority Hub and رتبه پنج عملکرد اول پروژه ADOL-C  جدول 3-4 شباهت های کوزین با پنج عملکرد اول ( با توجه به مرکز و رتبه اقتدار) پروژه CS Parse  جدول 4-4 شباهت های کوزین با پنج عملکرد اول ( با توجه به مرکز و رتبه اقتدار) پروژه ADOL-C | 36  46  46  48  48 |

**فهرست علائم اختصاری**

BTF: Block Triangular Form شکل مثلثی مسدود

DMM: Domain Mapping Matrix ماتریس نقشه برداری دامنه

DSM: Design Structure Matrix ماتریس ساختار طراحی

HITS: Hypertext Topic Search جستجوی موضوعی ناشی از ابرمتن

MDM: Multi-Domain Matrix ماتریس چند دامنه

SVD: Singular Value Decomposition تجزیه مقدار منفرد

## 

## فصـل 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 [7]. 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 [7]. از آنجا که DSM را می توان با ماتریسی در 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 (BTF)" AP 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 [14 ، 15 ، 1 ، 13]. 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 [9].

پروفسور حسین و گروهش ویژگی های معماری مجموعه کوچکی از نرم افزارهای علمی نمایشی را مطالعه کرده اند [15]. 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 [4 ، 19].

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 [16].

ابزارهای تجزیه و تحلیل مانند "درک" [21] 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 [13] ، هدف ما در این کار گروه بندی یا دسته بندی عناصر طراحی به "ردیف" بر اساس "اهمیت" آنها است.

## 1-1 مقدمه

## ۱-1-۱ 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 [11]. 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 [15]. 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.

## 2-1-1 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 [20]. 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 [2].

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:

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

2. 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.

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

## **3-۱ - 1 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 [5]. 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.

## **4-۱ - 1 معیار های نرم افزار**

معيارهاي نرم فزار از يک ديدگاه به دو دسته ي معيارهاي فرآيند و معيارهاي محصول تقسيم ميشـوند. معيارهـاي فرآينـد بـه عنـوان معيارهاي مديريت شناخته شده اند و براي اندازه گيري مورد استفاده قرار ميگيرند و از خواص فرايند براي به دست آوردن انـدازه نرم افزار استفاده ميشود. معيارهاي فرآيند شامل معيارهاي هزينه ، معيارهاي تلاش، معيارهاي پيشـرفت و معيارهـاي اسـتفاده مجـدد است . معيارهاي فرآيند در پيش بيني اندازه سيستم نهايي و تعيين يک پروژه با توجه به برنامه در حال اجرا کمک ميکند.

معيارهاي محصول نيز به عنوان معيارهاي با کيفيت شناخته شده اند و براي اندازه گيري خواص يک نرم افـزار اسـتفاده مـي شـوند.

معيارهاي محصول شامل محصول غير قابل اطمينان، معيارهاي عملکرد، معيارهاي قابل استفاده، معيارهاي هزينه ، معيارهـاي انـدازه، معيارهاي پيچيدگي و معيارهاي سبک است . معيارهاي محصول در بهبود کيفيت بخش هاي مختلف سيستم و مقايسـه ميـان سيسـتم هاي موجود کمک ميکند.

به طور معمول استفاده از معيارهاي نرم افزار با اهداف زير صورت ميگيرد:

* مطالعه تطبيقي روش هاي مختلف طراحي سيستم هاي نرم افزاري.
* براي تجزيه و تحليل ، مقايسه و مطالعه زبان هاي برنامه نويسي مختلف با توجه به ويژگي هاي آنها.
* مقايسه و ارزيابي توانايي ها و بهره وري از افراد درگير در توسعه نرم افزار.
* تهيه مشخصات از نرم افزار با کيفيت .
* تأييد انطباق نرم افزار مورد نياز سيستم ها و مشخصات.
* ساخت استنتاج در مورد تلاش براي طراحي و توسعه سيستم هاي نرم افزاري.
* گرفتن يک ايده در مورد پيچيدگي کد.
* گرفتن تصميمات مربوط به تقسيم بيشتر ماژول پيچيده که بايد انجام شود يا نه .
* ارائه راهنمايي به مدير منابع براي بهره برداري مناسب خود.
* مقايسه و مبادلات بين طراحي و توسعه نرم افزار و هزينه هاي تعمير و نگهداري.
* ارائه بازخورد به مديران نرم افزار در مورد پيشرفت و کيفيت در مراحل مختلف چرخه عمر توسعه نرم افزار.
* تخصيص منابع تست ، براي تست کد.

از طرف ديگر معيارهاي نرم افزار محدوديت هايي هم دارند. از آن جمله ميتوان به موارد زير اشاره نمود:

* استفاده از متريک هاي نرم افزاري هميشه آسان نيست و در برخي موارد دشوار و پر هزينه است .
* تأييد و توجيه معيارهاي نرم افزار بر اساس داده هاي تاريخي.تجربي که اعتبار آن به سختي قابل بررسي است .
* براي مديريت محصولات نرم افزاري مفيد هستند ولي براي ارزيابي عملکرد کارکنان فني کارا نيستند.
* تعريف و استخراج معيارهاي نرم افزار دقيق نيست و به طور کلي فرض بر اين است که معيارهاي نرم افزار اسـتاندارد نيسـتند و بـه ابزارهاي موجود و محيط کار بستگي دارد.
* اکثر مدل هاي پيش بيني در برآورد متغيرهاي خاصي که اغلب شناخته شده اند تکيه مي کنند.
* بسياري ازمدل هاي توسعه نرم افزارها احتمالي وتجربي هستند.

## **5-۱ - 1 معيارهاي اندازه نرم افزار**

* تعداد خط کد

تعداد خط کد يکي از اولين و ساده ترين متريک ها براي محاسبه اندازه برنامه هاي کامپيوتري است . به طـور کلـي از ايـن معيـار براي محاسبه و مقايسه بهره وري از برنامه نويسان استفاده مي شود. هر خط از متن برنامه بدون در نظر گرفتن توضـيحات و يـا خـط خالي يک خط کد در نظر گرفته ميشود.

* تعداد نشانه

در اين معيارها، يک برنامه کامپيوتري به عنوان يک مجموعه اي از نشانه ها در نظـر گرفتـه شـده اسـت کـه ممکـن اسـت براسـاس اپراتور يا عملوندها طبقه بندي شده باشد. همه معيارهاي نرم افزار را مي توان درقالب نمادهاي اساسي تعريف کـرد. ايـن نمادهـا بـه عنوان نشانه ناميده مي شود.

مقياس هاي اساسي عبارتند از:

n١= تعداد دفعات مشاهده اپراتورهاي منحصر به فرد

n٢= تعداد دفعات مشاهده شده از عملوند هاي منحصر به فرد.

N١= تعداد دفعات مشاهده کل وقوع اپراتورها.

N٢= تعداد مجموع تکرار عملوند.

* تعداد عملکرد

اندازه بزرگ محصول نرم افزاري را مي توان از راه بهتري و از طريق يک واحد بزرگتر به نام ماژول تخمين زد. ماژول را مي تـوان به عنوان بخشي از کد که ممکن است به طور مستقل وارد شده تعريف کرد.

به عنوان مثال، اجازه دهيد يک محصول نرم افزاري به N ماژول احتياج داشته باشد. به طور کلي توافق شده که انـدازه مـاژول بايـد حدود ٥٠-٦٠ خط از کد باشد. بنابراين تخمين اندازه اين محصول نرم افزاري حدود ٦٠×N خط از کد ميباشد.

## **6-۱ - 1 معيارهاي علمی نرم افزار**

مدل Halstead نيز به عنوان نظريه علم نرم افزار شناخته شده است . بر اساس اين فرضيه ساخت و ساز و برنامه شامل يـک فراينـد ذهني دستکاري اپراتورهاي منحصر به فرد (n١)و عملوند منحصر به فـرد(n٢) ميباشـد. ايـن بـدان معنـي اسـت کـه يـک برنامـه از اپراتورهاي N١ و عملوندهاي N٢ با انتخاب اپراتورهاي منحصر به فرد از n١ و عملوند هاي منحصر به فرد n٢ ساخته شده اند. بـا استفاده از اين مدل، Halstead از تعدادي از معادلات مربوط به برنامه نويسي مانند سطح برنامه ، تلاش زبان پياده سازي و غيـره به دست آمده است . ويژگي مهم و جالب اين مدل آن است که برنامه را مي توان از ويژگي هاي مختلف مانند اندازه، تلاش و غيره تجزيه و تحليل کرد.

واژگان برنامه اينگونه تعريف شده اند :

و طول واقعي برنامه ها به عنوان: يکي از اين فرايض اين نظريه اين است که طول يک برنامه به خوبي ساختار يک تابع فقط n١و n٢ است .

اين رابطه به عنوان معادله پيش بيني طول شناخته شده و تعريف شده است مانند:

طول پيشنهادي برآورد برخي ديگر محققان: برآورد طول برنامه جنسن

اين توسط جنسن و وايرون براي زمان واقعي برنامه هاي کاربردي نوشته شده توسط پاسکال بکار گرفته و معتبر شـده اسـت وحتـي نتايجي دقيق تر از برآورد هالستيد کسب کرد.

برآورد طول برنامه جيف [Nz] :

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

برآورد طول برنامه بيل مش [Nb] :

= تعدادي ازاپراتورهاي منحصربه فرد که شامل عملگرهاي اساسي، کلمات وتوابع .روشها.

= تعداد عملوندهاي منحصر به فرد.

حجم برنامه (V)

برنامه نويسي لغت منجر به اقدامات يکي ديگر از اندازه ها که ممکن است به عنوان تعريف شود.

حجم پتانسيل (\*V)

ممکن است به عنوان تعريف شود که در آن n١ به معني حداقل تعداد اپراتورها و n٢ حداقل تعداد عملوند ها است .

## **7-۱ - 1 معيارهاي کنترل جریان**

* معيار McCabe’s Cyclomatic

McCabe برنامه هاي کامپيوتري را به عنوان يک مجموعه ي قوي متصل به گراف جهت دار تفسير مي کند. گره ها نشان دهنده بخش هايي از کد منبع هستند که هيچ شعبه و کمان نماينده که ممکن است در حين کنترل جريان اجراي برنامه انتقال يابند، ندارند.

مفهوم گراف برنامه براي اندازه گيري استفاده مي شود و آن را براي اندازه گيري و کنترل تعدادي از مسيرها از طريـق يـک برنامـه استفاده ميکنند. پيچيدگي يک برنامه کامپيوتري را مي توان با پيچيدگي توپولوژيک يک گراف ارتباط داد.

McCabe تعداد cyclomatic را پيشنهاد داد، (V) G نظريه گراف به عنوان يک شاخص پيچيدگي نرم افزار ارائـه شـده اسـت.

تعداد cyclomatic برابر با تعداد مسيرهاي خطي مستقل از طريق يک برنامه در نمودار آن است. بـراي کنتـرل برنامـه ، گـرافG، تعدادcyclomatic،(V) G اينگونه در نظر گرفته شده اند.

## 2-1 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.

1. تجزیه و تحلیل ساختار وابستگی نرم افزار با ثبت ارتباط تماس گیرنده بین عملکردها با استفاده از ابزار "درک" [21].

2. رتبه بندی عناصر طراحی (توابع) با استفاده از مفهوم "اهمیت" آن عناصر طراحی [13].

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

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

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

## 3-1 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.

## 4-1 سازمان پایان نامه

در مجموع 5 فصل در این پایان نامه وجود دارد. فصل 1 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 2 ، 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 3 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 4 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 5 ارائه می دهیم.

## 

## فصـل دوم

**استخراج و مدل سازی وابستگی**

## 1 – 2 طراحی ساختار ماتریس

طراحی ساختار ماتریس (DSM) یک نمایش ساده فشرده و بصری از یک سیستم یا پروژه در قالب یک ماتریس مربع است [7]. 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:

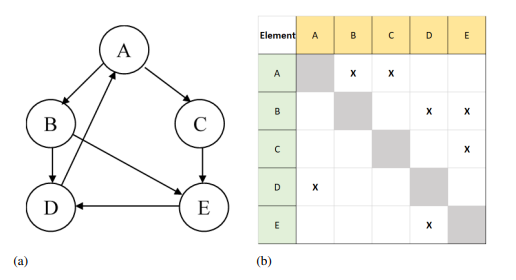
• DSM یک قالب نمایشی فشرده برای سیستم های پیچیده بزرگ ارائه می دهد.

• 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 2.1 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 2.1 الف نشان داده شده است).



شکل 2.1: 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 2.1 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 [22]. 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 [21] 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 [21]:

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

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

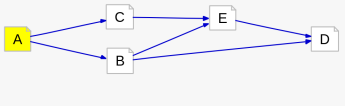
• خروجی صفحه گسترده روابط وابستگی

• 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 2.5 نشان داده شده است در نظر بگیریم. اکنون می توانیم این مثال را برای نشان دادن وابستگی های مختلف نشان دهیم:

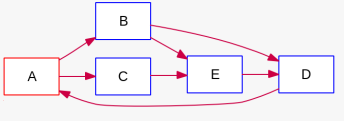
• شامل وابستگی: نمودار Include Dependency فایل هایی را نشان می دهد که برای پیاده سازی مورد نیاز است. برای مثال یک فایل "xyz.c" به فایل دیگری وابسته است "xyz.h" بدین معناست که "xyz.c" فایل "xyz.h" را در بر گرفته است. این وابستگی را می توان با استفاده از Understand پیدا کرد.

• وابستگی به تماس: با استفاده از درک شکل 2.2 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 2.3 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 2.3 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 2.2: رابطه تماس ها

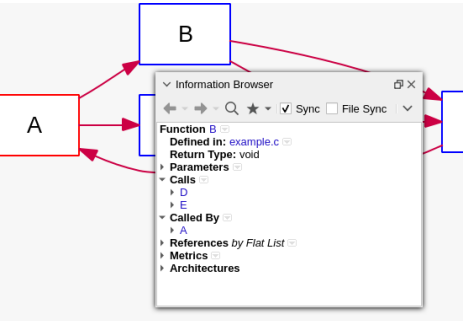
• وابستگی اولیه: وابستگی init بر مقداردهی اولیه یک شی تمرکز می کند.



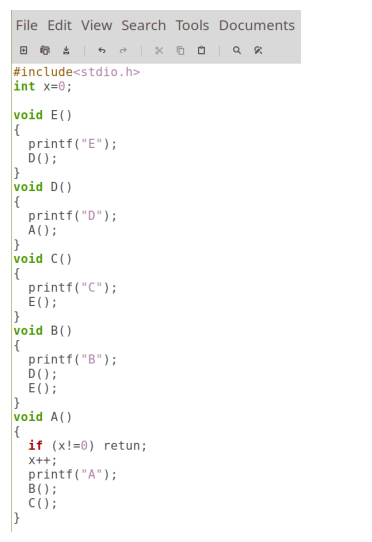
شکل 2.3: وابستگی تابع

• 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 2.4 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 2.4: شرح عملکرد



شکل 2.5: یک مثال شبه برای وابستگی به تابع

## 3 – 2 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.

## 1 – 3-2 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 ، {(1،2) ، (2،4) ، (3،6) ، ...} است.

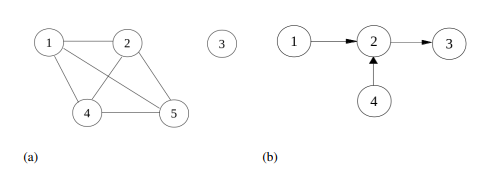
مثال نشان داده شده در شکل 2.5 را می توان به صورت R = { (A,B)، C)، (A، D)، (B، E)، (B، E)، (C، A)، (D ، (D، E)}.

## ۲ – 3-2 نمودارها

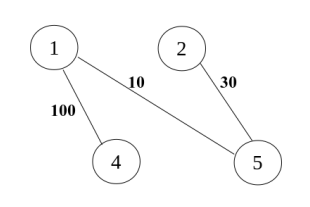
اجازه دهید G (V,E) ، یک گراف باشد که V مجموعه محدود رأس ها و E مجموعه ای از لبه ها است که رابطه زوجی بین رئوس را در V نشان می دهد. چند دسته از نمودارها وجود دارد و ما آنها را با توجه به نیاز خود برای تجزیه و تحلیل استفاده می کنیم.

• 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 2.6 (الف) نمونه ای از یک گراف غیر مستقیم و شکل 2.6 (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 2.7 نمونه ای از نمودار ارزشمند است که در آن هر لبه مقداری روی خود دارد. از طرف دیگر شکل 2.6 را می توان به عنوان نمونه ای از دو نمودار بدون ارزش در نظر گرفت.



شکل 2.6: نمونه ای از نمودار: الف) گراف غیرمستقیم ، ب) نمودار جهت دار



شکل 2.7: 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 2 را در لبه بین A و B قرار می دهیم.

اکنون ما نیاز به برخی از تعاریف اولیه داریم.

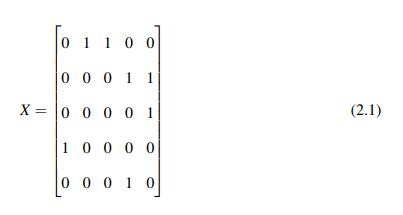
درجه یک راس تعداد راس هایی است که در مجاورت آن راس قرار دارند. با این حال ، تعداد لبه هایی است که در آن راس قرار می گیرند. به عنوان مثال در شکل 2.6 (a) ، Vertex 4 دارای درجه 3 است. یک راس صفر درجه ایزوله نامیده می شود Vertex 3 در شکل 2.6 (a) یک راس جدا شده است. رأسی با درجه 1 آویز نامیده می شود. رأس 1 در شکل 2.6 (b) چنین رأسی است.

در یک دیگراف (شکل 2.6 (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 2.6 (b) ، Vertex 2 دارای درجه 2 درجه بالاتر و درجه 1 است.

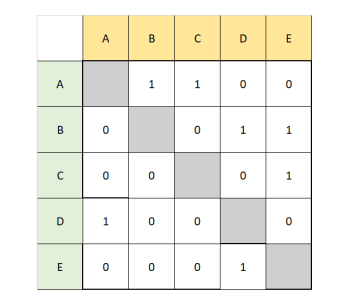
اگر مسیری از هر راس به هر راس دیگر وجود داشته باشد ، نمودار متصل می شود. حداکثر زیرگراف متصل را جزء می نامند. نمودار نشان داده شده در شکل 2.6 (الف) دارای دو جزء است: {1،2،4،5} و {3}. حداکثر زیرگراف یک زیرگراف است که برخی از ویژگی های مشخص شده (مانند اتصال) را برآورده می کند و بدون نقض ویژگی نمی توان به آن راس اضافه کرد.

## 3 – 3-2ماتریس 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.



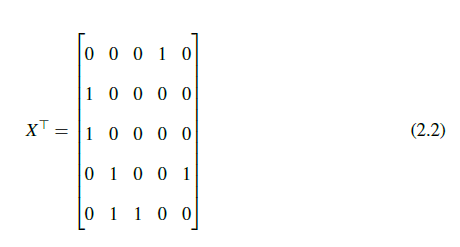
در شکل 2.8 ما DSM را با توجه به مثال نشان داده شده در شکل 2.5 نشان می دهیم. در اینجا ردیف ها مربوط به توابع تماس فرستنده و ستون ها مربوط به توابع تماس گیرنده است.



شکل 2.8: نمونه ای از DSM

نمایش ماتریس معادل در رابطه 2.1 ارائه شده است. ورودی Xi j = 1 به این معنی است که تابع در Rowi تابع را در Columnj فراخوانی می کند.

اگر می خواهیم رابطه در جهت دیگری باشد ، می توانیم ماتریس X را به جابجا کرده و XT را بدست آوریم (به رابطه 2.2 مراجعه کنید). وقتی X متقارن است XT = X داریم.



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.

## 

## فصـل سـوم

**تجزیه و تحلیل وابستگی**

زمانی که درمورد پيچيدگی نرم افزار بحث می کنيم باید توجه داشته باشيم که پيچيدگی مفهوم ثابتی نيست و باید در شرایط مختلف مورد برسی قرار گيرد.از این رو می گویيم پيچيدیگی نرم افزار اصطلاح غير استانداری است زیرا از محدوده مشخصی برخوردار نيست، امکان دارد نرم افزار بزرگی با خطوط زیادی از کد و پيمانه های مختلف و مرتبط را پيچيده و همينطور نرم افزار کوچکی با الگوریتمی سخت را نيز پيچيده ارزیابی نمایيم .برای همين است که نمی توان پيچيدگی را از یک بعد مورد برسی قرار داد. موسسه IEEE تعریفی از پيچيدگی ارائه داده است که به شرح زیر می باشد: "ميزان سختی در درک یک سيستم و یا مؤلفه که طراحی و پياده سازی شده است." از این رو باید عنوان کرد که پيچيدگی نرم افزار به فاکتورهایی از اندازه گيری تاثير گذار در هزینه ، نگهداری و توسعه اطلاق می شود.البته افراردی چون "زئوس" این نظریه را زیر سوال برده و پيچيدگی را از دید روانشناسی نرم افزار مورد برسی قرار داده و دسته بندی نموده اند Zeus پيچيدگی روانشناسی نرم افزار را به سه بخش زیر تقسيم نموده:

1-پيچيدگی در مساله

2-پيچيدگی در طراحی سيستم

3-پيچيدگی رویه ای

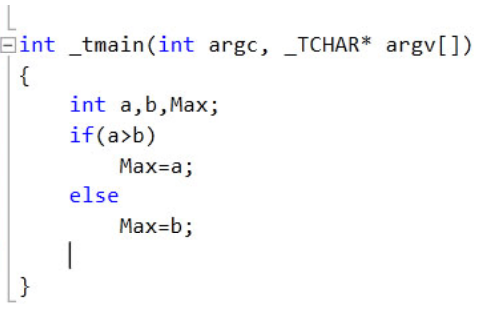
مثلا" درمورد پيچيدگی در مساله باید عنوان داشت که این امر به پيچيدگی کلی الگوریتم مساله بر ميگرد.یعنی ممکن است یک سيستم پيچيدگی ذاتی کم یا بيشتری داشته باشد که در ميزان پيچيدگی کلی آن تاثيرگذار خواهد بود.همينطور می توان به پيچيدگی موجود در طراحی سيستم اشاره داشت که به دو قسمت: ساختاری و داده ای تقسيم می شود. پيچيدگی ساختاری به مفهوم کلی اتصال اشاره دارد و اتصال بين پيمانه های درونی در زبان برنامه نویسی را مورد برسی قرار می دهد. به عنوان مثال ؛ در یک زبان برنامه نویسی (فرض کنيد C ++ (ارتباطاتی بين پيمانه های درونی زبان + کلاسها و کتابخانه ها وجود دارد که به خودی خود پيچيدگی هایی را به دنبال خواهد داشت. پيچيدگی داده ای به مفهوم دیگری با عنوان "چسبندگی" اشاره دارد و چسبندگی های درون ساختاری را مورد برسی قرار می دهد. پيچيدگی ساختاری و داده ای ارتباط مبتنی بر IN-FAN و OUT-FAN در پيمانه ها و متغييرهای خروجی/ورودی را مورد برسی قرار مي دهند. آنگونه که اشاره شد پيچيدگی های ساختاری و داده ای به روابط بين پيمانه های درون سيستمی نرم افزار اشاره دارند حال آنکه در زبان های پيش گام و با به کارگيری OOP یا برنامه نویسی شیء گرا از ميزان این پيچيدگی ها تا حد زیادی کاسته شده است.

باید اشاره داشت که اغلب پيچيدگی های درونی مانند آنچه گفته شد درصد پایين و ضریب اندکی از پيچيدگی نرم افزار را شامل می شوند و در شکل گيری بحران نرم افزار آنچنان تاثير گذار نيستند.

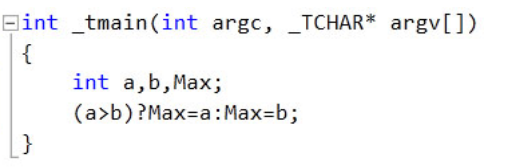
پيچيدگی رویه ای: این پيچيدگی به ساختار کلی برنامه ، خطوط کد ، حلقه ها و شرط ها و طریقه به کار گيری آنها اشاره می کند. به صورت کلی پيچيدگی بخش غيرقابل اجتناب در توليد نرم افزار است یعنی به هر نحو ميزانی از پيچيدگی در نرم افزار وجود دارد اما مشکل بحران نرم افزار زمانی رخ می دهد که این پيچيدگی ها از یک حد بيشتر شوند. وظيفه توسعه دهندگان نرم افزار این است که این پيچيدگی ها را تا حد ممکن کاهش دهد و این یعنی ارائه راهکار و رویه ای مناسب تا یک الگوریتم هر چه ساده تر و مفيد تر پياده سازی شود .

**نمونه كد**

در ادامه به مثال زیر توجه کنيد که در آن یک روش رویه ای در کنترل پيچيدگی اعمال شده است.برنامه زیر دو عدد را مقایسه کرده و عدد بزرگتر را درون متغييری با نام Max ميریزد .



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



در مثال فوق همان برنامه اول منتهی با رویه ای مناسبتر برای جلوگيری از پيچيدگی 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 پيچيدگی امکان پذیر و راهبردی است. فراموش نکنيدم که هرچه مدل ما حاوی اطلاعات بيشتری باشد، طبيعتا مدل ما پيچيده تر خواهد بود.

**ماهيت پيچيدگي نرم افزار**

همانطور که در بحث های ابتدایی و الزام آور بودن اصول اندازه گيری نرم افزار (measurement) مطرح شد، شناخت ماهيت نرم افزار باعث می شود از عمليات کورکورانه و به قولی آزمون و خطا جلوگيری شود. عدم رعایت این اصل(شناخت) یکی از مهمترین عوامل پيچيدگی می باشد.زمانی که برنامه ساز در مورد برنامه پيش رو شناخت کافی نداشته باشد و بر ماهيت کلی آن اشراف نداشته باشد قطعا نرم افزار توليدی او نيز کارایی بالایی نخواهد داشت و امکان بروز اختلالات غيرمرقبه در حين کار با نرم افزار بالا می رود به گونه ای که عدم رضایت مشتریان را به دنبال خواهد داشت.

**اجزاي كليدي در پيچيدگي**

پيچيدگی برچسبی است که ما به وجود متغييرهای وابسته موجود در یک سيستم نسبت می دهيم،هرچه متغييرها بيشتر باشد وابستگی بين آنها بيشتر می شود. اتصالات بين متغييرها ویژگی های زیادی برای آن واحد می طلبد که منجر به بروز پيچيدگی بيشتر خواهد شد. پيچيدگی چند جزء کليدی دارد که به شرح زیر می باشد:

1-مقياس :مقدار اجزاء سيستم نرم افزاری

2- تنوع:گستره اجزاء متنوع تشکيل دهنده سيستم

3- اتصال : ارتباط بين مؤلفه ها را فراهم می کند

**مقياس** : مقياس به تنهایی مساله نيست و در صورت زیاد بودن مقياس با استفاده از داده های آماری می توان آن را تعيين و کنترل نمود.

**تنوع** : این جزء انواع اجزائی را که باید تحليل شوند افزایش می دهد. هرچه تنوع بيشتر شود تلاش بيشتری برای درک سيستم لازم داریم.

**اتصال** : با افزایش مقياس قاعدتا تعداد اتصالات و تعامل بين اجزاء بيشتر خواهد شد و این امر نيز به نحوی موجب پيچيدگی می شود.

**بررسي معيارهاي پيچيدگي**

پيچيدگی به صورت کم و بيش وجود دارد و وجود خواهد داشت به صورت کلی پيچيدگی دارای رابطه مستقيم با گستردگی پروژه می باشد. پر واضح است که هر مقدار که تعداد خطوط کد پروژه بيشتر باشد قاعدتا" از مدل ها ، کلاس های زیادی برخوردار بوده و پيچيدگی ذاتی آن بيشتر است.حالا کافيست مطالب قبلی را به یاد بياوریم "پيچيدگی به صورت کامل قابل از بين بردن نيست ، از این رو باید آن را کنترل نمود."

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

• با پيچيدگی سخت افزاری در وسایل فيزیکی و مکانيکی تفاوت دارد و از یک مفهوم منطقی در نرم افزار نشآت می گيرد

•یک خاصيت ذاتی در نرم افزارهای بزرگ می باشد

•پيچيدگی حوزه مساله

1. نيازمندی های گوناگون و متضاد
2. ارتباط بين کاربر و نرم افزار
3. تغيير نيازها

• پيچيدگی در فرایند توليد

• عدم انعطاف پذیری سيستم و استاندارد نبودن اجزاء آن

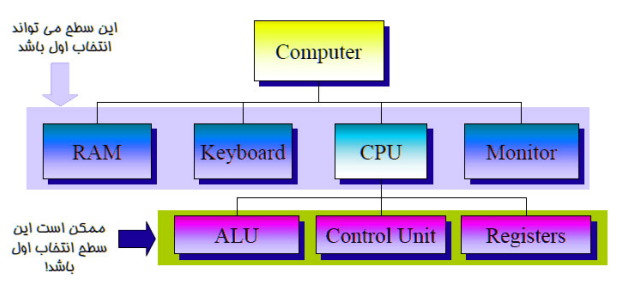
• مشکل در توصيف رفتارهای سيستم های گسسته

**پيچيدگي عارضي**

پيچيدگی ذاتی عموما مشخصه هر مساله نرم افزاری و الگوریتم های مربوط به آن می باشد، یعنی اگر مساله ای ساده را نيز در نظر بگيریم شامل راه حلی می باشد ، همين راه حل هر چند ساده یک پيچيدگی ذاتی برای نرم افزار محسوب می شود. در مقابل پيچيدگی ذاتی ، پيچيدگی عارضی قرار گرفته است که از انتخاب صورت گرفته در ساخت سيستم 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 پيچيدگی عارضی این است که ابتدا مساله را درست تعریف کنيم سپس به دنبال راه حل(چگونگی) انجام مساله برویم. تمام مواردی که در سرفصل های قبلی پيچيدگی مطرح شد به نوع عارضی اشاره دارند و با به کارگيری راهکارهای ذکر شده تا حدود زیادی می توان در مقابل بروز اینگونه پيچيدگی ها مصون بود. البته برخی راهکارها در مقابله با بروز پيچيدگی عارضی وجود دارد که در ادامه به آن اشاره خواهيم داشت.

**پيچيدگي سلسله مراتبي (Hierarchy )**

در اغلب سيستم ها ، پيچيدگی به صورت سلسله مراتبی ظاهر می شود. در اینگونه سيستم ها ارتباط بين اجزای درونی سيستم قویتر از ارتباط بين خود زیر سيستم هاست و این یکی از ویژگی های پيچيدگی در ساختار سلسله مراتبی می باشد. شکل زیر نمایانگر یک سيستم پيچيده سلسله مراتبی می باشد.



سيستم های سلسله مراتبی معمولا از تعداد کمی از زیر سيستم های مشخص و متفاوت تشکيل می شوند که این زیر سيستم ها به صورت های گوناگون و ترتيب های مختلف ظاهر می شوند.

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

شيء گرايي و نقش آن در كنترل پيچيدگي نرم افزار یکی دیگر از مواردی که موجب کاسته شدن از پيچيدگی می شود به کار گيری روش های شیء گرایی(Oriented Object) می باشد. شیء گرایی دارای 4 اصل کلی می باشد که عبارتند از:

1-تجرید

2-محصور سازی

3-واحد بندی

4-سلسله مراتب

**تجريد(Abstraction )**

به صورت کلی به فرآیند تشخيص خصيصه های مهم یک پدیده گویند، در تجرید باید از پرداختن به ویژگی های موقت و غير مهم جلوگيری کرد و خصيصه ها و ویژگی های مهم یک پدیده را مدنظر قرار دهيم. مثلا یک دانشجو را در نظر بگيرید(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 نکنيم امکان استفاده از ویژگی های موقت و غيرضروری مثل :شماره تلفن همراه ، وضعيت جسمانی و... را فراهم کرده ایم که این امر به خودی خود موجب بروز پيچيدگی بيشتر خواهد شد. به زبان ساده اصل تجرید مؤلفه های غيرضروری که از اهميت کمتری برخوردار هستند را حذف و به ویژگی های مؤثر در طراحی صحيح الگوریتم و رسيدن به روشی مناسب برای حل مساله می پردازد.

**محصورسازي (Encapsulation )**

محصورسازی موجب می شود تا پدیده از اثرات مخرب ميحط خارجی به دور بماند و همينطور طریقه دسترسی سایر اشياء به آن پدیده را نيز کنترل نمایيم. محصورسازی با کنترل راه های دسترسی به یک پدیده (شیء) باعث جلوگيری از تاثيرات منفی و احتمالی بر روی پدیده می شود و همينطور از گسترش خطاهای رخ داده در درون پدیده به سایر اشياء جلوگيری می کند. از جمله سایر نقش های محصورسازی در کنترل پيچيدگی این است که به دليل ثبات در واسط ارتباطی (Interface ) می توان تغييرات زیادی بر روی آن اعمال کرد و همينطور 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 تغييرات آماده باشد باز و به ماژولی که امکان تغيير در آن وجود نداشته باشد ، ماژول بسته می گویيم. در عمليات Encapsulation هر دو ماژول باز و بسته کاربردی و لازم می باشند.

**واحد بندي(Modularity)**

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

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

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

3 – استقلال یا Independent

4- Well defined interfaces یا واسط های خوش تعریف

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

1 - ماژول بندی نباید با تعداد خيلی زیاد و یا خيلی کم باشد

2- معياری 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 مراتب A-IS

ساختار شیء :سلسله مراتب OF-PART

در کل باید گفت فرمول شکل اصلی یک سيستم پيچيده به این صورت می باشد:

شکل اصلی سيستم پيچيده =خواص پنجگانه + سلسله مراتب A-IS + سلسله مراتب OF-PART

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

سلسله مراتب OF-PART : بيانگر قسمتی از چيزی 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 کليت می باشد. پياده سازی در قالب سلسله مراتب ما را در دریافت بهتر مساله و درک بهتر آن کمک کرده لذا موجب کاستن پيچيدگی می شود.

می توان ویژگی های استفاده از شیء گرایی را در کنترل پيچيدگی چنين ارزیابی کرد:

1. منظم کردن فرآیند توليد نرم افزار
2. بالا رفتن درک مساله در سيستم های نرم افزاری
3. ارائه مدلی قدرتمندتر برای کنترل هرچه بيشتر پيچيدگی نرم افزاری
4. کاهش هزینه های توليد (زمانی – مالی )
5. استفاده مجدد و انعطاف پذیری بيشتر همينطور افزودن امکان پشتيبانی 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.

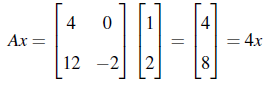
## 1- 3 ارزش های ویژه و بردارهای ویژه

اگر A یک ماتریس n n باشد ، یک بردار غیر صفر x در Rn نامیده می شود اگر Ax یک مقیاس مقیاس x باشد که



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

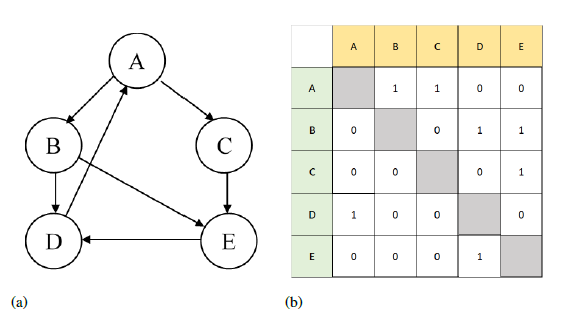
به عنوان مثال ، بردار x = بردار ویژه A = مربوط به ارزش ویژه λ = 4 می باشد



برای یافتن مقادیر ویژه یک ماتریس n n ، معادله 3.1 را به صورت Ax = λIx یا به شرح زیر بازنویسی می کنیم:

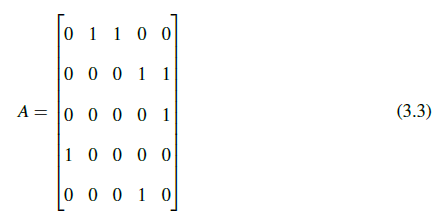


بنابراین معادله 3.2 دارای راه حل غیر صفر است اگر و فقط اگر det (λI-A) = 0. در اینجا می توانیم با یک مثال این اصطلاحات ریاضی را مورد بحث قرار دهیم. برای سادگی ، مثال ذکر شده در فصل قبل را به خاطر می آوریم.



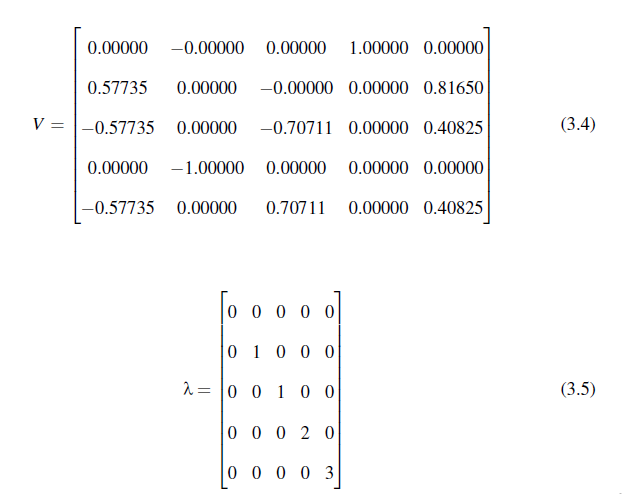
شکل 3.1: یک مثال: الف) نمودار ، ب) ماتریس مجاور معادل یا DSM

از مثال نشان داده شده در شکل 3.1 ما DSM را به عنوان ماتریس A به عنوان معادله زیر دریافت می کنیم:

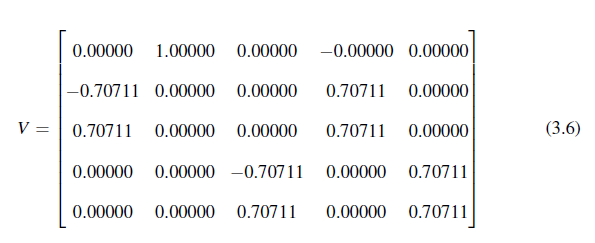


در بخش 3.3 ما در مورد الگوریتم HITS از [17] بحث خواهیم کرد. برای توصیف آن الگوریتم ما به دو ماتریس ویژه B1 = AAT و B2 = ATA نیاز داریم که AT ترانسپورس ماتریس A است (ماتریس A یک ماتریس بولی است). در الگوریتم HITS ما بزرگترین ارزش ویژه و بردار ویژه مربوط به ماتریس های B1 و B2 را محاسبه می کنیم. این ماتریس های B1 و B2 ماتریس متقارن هستند. ماتریس های متقارن دارای مقادیر ویژه (واقعی) هستند.

فرمان=eig(B1) [V,λ] 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 B1\* V = V\* λ .



به طور مشابه ما ماتریس بردارهای اختصاصی V را که با مقادیر ویژه λ ماتریس B2 مرتبط است ، محاسبه می کنیم.



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.

## 2- 3 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 3.1 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 [3].

## 3-3 جستجوی موضوعی ناشی از ابرمتن (HITS)

جستجوی موضوعی ناشی از ابرمتن (HITS) یک الگوریتم است که توسط کلاینبرگ [17] 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 (xi) و وزن هاب (yi). در ابتدا به هر xi و yi یک مقدار غیرجدی دلخواه داده می شود. سپس وزن ها با استفاده از رابطه 3.7 و معادله 3.8 برای k = 1,2,3,… به روز می شوند.



* **وزن قدرت به روز رسانی**: در اینجا ما از رابطه 3.7 استفاده می کنیم. kth در گره تکرار i یک وزن جدید Xi(k) تعیین می شود که برابر است با مجموع yj(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 3.8 در اینجا استفاده می شود. وزن hub جدید yi(k) مجموع Xi(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.

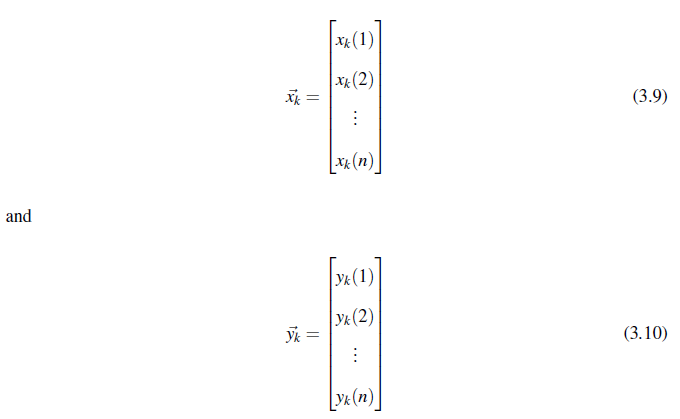
توجه داشته باشید که وزن های هاب از وزن های قدرت فعلی محاسبه می شوند که در آن وزن های مجاز از وزنهای هاب قبلی محاسبه شده است.

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 [18].

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



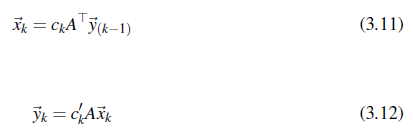
اکنون ، در تکرار k برای n گره ، می توانیم مقدار hub و authority را از نظر بردارها نشان دهیم. اگر xk نشان دهنده بردار مقادیر قدرت و yk نشان دهنده بردار مقادیر مرکز در تکرار k باشد ، برای n گره داریم



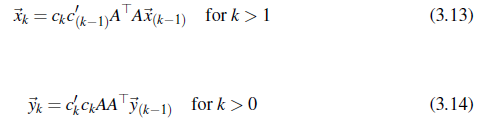
اگر k = 0 ، سپس با استفاده از معادلات 3.9 و 3.10 می توانیم x0 و y0 را مقداردهی کنیم ، به عنوان

x0 (1) = x0 (2) = …= x0 (n) = (1 = ) و y0 (1) = y0 (2) = …= y0 (n) = (1 = ).

بگذارید A یک ماتریس مجاورت نمودار هدایت شده G. باشد سپس با استفاده از معادله 3.11 و 3.12 می توانیم الگوریتم گفته شده در بالا را نشان دهیم.



ck و c’k از رابطه 3.11 و معادله 3.12 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 1 باشد. با در نظر گرفتن این معادلات ، اکنون می توانیم روش HITS را با استفاده از معادلات زیر نشان دهیم [17]:

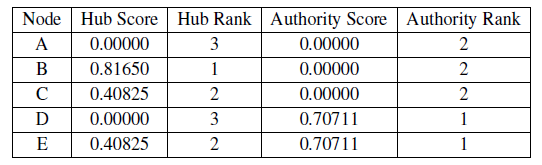


بنابراین می توان گفت که HITS یک روش قدرت تکراری برای محاسبه بردار ویژه غالب برای AAT و AT A است [10]. بردار ویژه غالب ، ستون ماتریس V (بردار ویژه مربوطه) است که با ارزش ویژه مطابقت دارد.

نمرات هاب و نمرات اقتدار توسط ورودی های بردار ویژه غالب AAT و AT A به ترتیب تعیین می شوند [3].

مجدداً می توان شکل 3.1 را به عنوان مثال برای توضیح الگوریتم HITS در نظر گرفت که در آن ماتریس مجاورت آن نمودار در رابطه 3.3 آمده است.

جدول 3.1: Hub and Authority Ranking (نمرات مربوط به بردار ویژه غالب است)



بردارهای ویژه AAT و AT A مربوط به بزرگترین ارزش ویژه λmax = 3 (به رابطه 3.5 مراجعه کنید) رتبه بندی هاب ها و مقامات (با استفاده از الگوریتم HITS) را که در جدول 3.1 نشان داده شده است ، به دست می آورند. در معادله 3.5 مشاهده می کنیم که ارزش ویژه غالب در ستون 5th است و از این رو ستون پنجم V در معادلات 3.4 و 3.6 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 {3؛ 1؛ 2؛ 3؛ 2} و رتبه بندی مقامات {2؛ 2؛ 2؛ 1؛ 1} است.

## 

## فصـل 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 محصولT B = A\* A 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" [21] و "شباهت" بین عناصر طراحی (توابع) با استفاده از یک متریک شباهت مناسب (شباهت کسینوس) کشف می کنیم. سپس با مفهوم "اهمیت" عناصر طراحی [13] 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.

## 1 – 4 روش شناسی

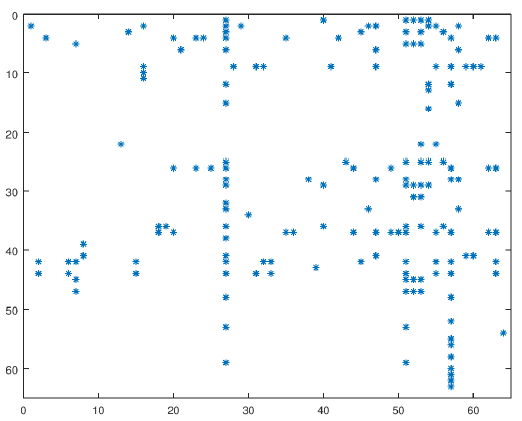
این بخش را با توصیف روش استخراج وابستگی و سپس ساخت DSM شروع می کنیم. با استفاده از ابزار تجزیه و تحلیل "درک" [21] ما می توانیم ساختار وابستگی نرم افزار را مشاهده و استخراج کنیم. ما برای تجزیه و تحلیل ساختار وابستگی برنامه ها ، ارتباط بین تماس گیرنده بین عملکردها را استخراج می کنیم. روش یافتن "اهمیت" عناصر طراحی [13] 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 [8] (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 4.2.1 به طور مختصر شرح داده می شود. سپس نتایج این نرم افزار که از آزمایشات خود بدست آورده ایم در بخش 4.3 مورد بحث قرار می گیرد.

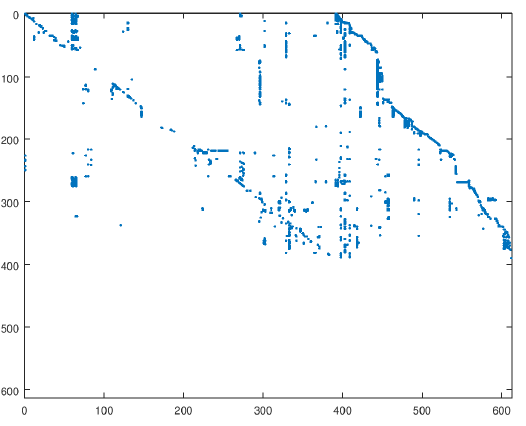
## 1-۱-4 استخراج وابستگی ها

در بخش 2.2 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 [21] 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 4.1 و شکل 4.2 ماتریس های CSparse و ADOL-C را به ترتیب با استفاده از Octave ترسیم می کند.



شکل 4.1: ماتریس پراکنده CSparse

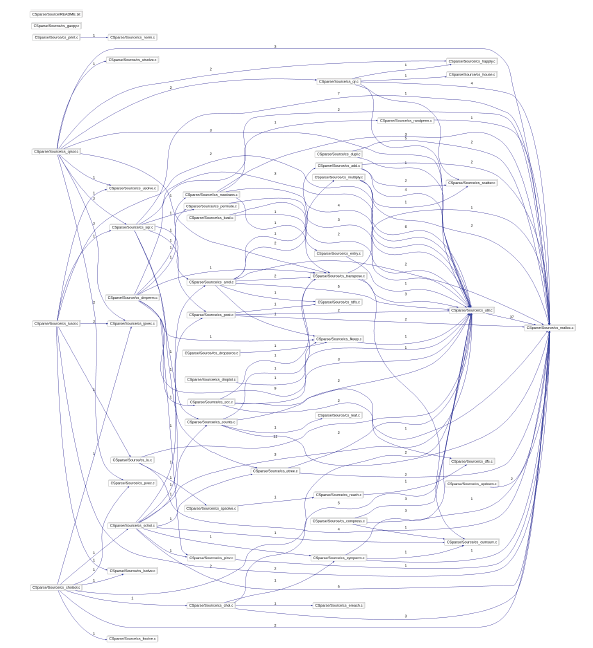


شکل 4.2: ماتریس پراکنده ADOL-C

برای استخراج وابستگی ها ابتدا بسته کامل (آخرین نسخه) نرم افزار سیستم (CSparse و ADOL-C) را بارگیری می کنیم. سپس بسته را در "Understand" SciTool وارد می کنیم. سپس می توان نمودارهای تماس را تجسم کرد (به شکل 4.3 و شکل 4.4 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.

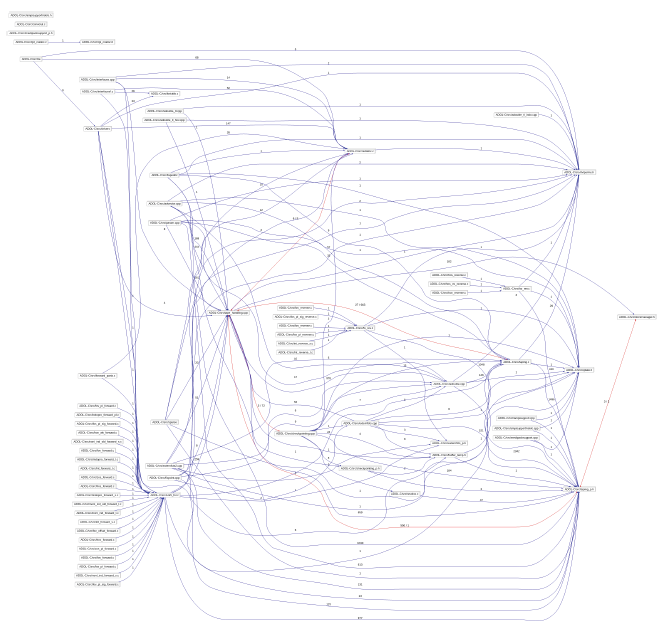
## 2 - ۱ - 4 ساخت DSM ها

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 64 عملکرد و پروژه ADOL-C را با 612 عملکرد بررسی کرده ایم. بنابراین CSparse دارای DSM با ماتریس 64X64 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 4.3: نمودار وابستگی CSparse

شکل 4.5 DSM را برای CSparse نشان می دهد. ترتیب توابع (در سطرها یا ستون ها) در این ماتریس باینری 64X64 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 612X612) داریم.

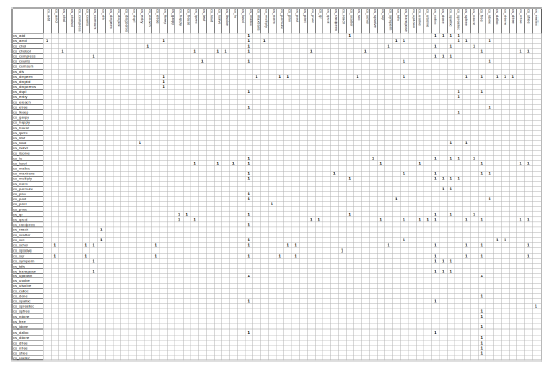


شکل 4.4: نمودار وابستگی ADOL-C

## 3 - ۱ - 4 مرکز محاسبات و رتبه بندی قدرت

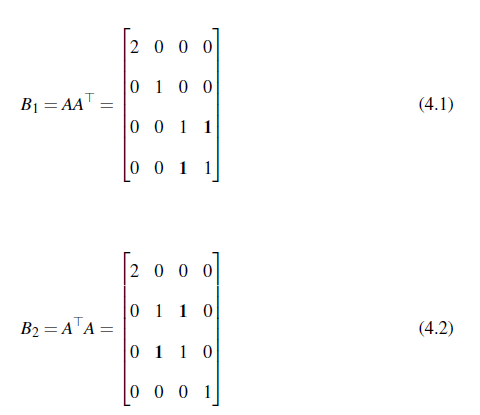
در فصل 3 با استفاده از یک مثال کوچک ، ما برخی از روش های تجزیه و تحلیل وابستگی بین توابع را مورد بحث قرار داده ایم. برای مثال محاسبه ارزش ویژه و بردار ویژه از ماتریس مشخص شده ، هاب ها و مقامات را مشخص می کند و از روش HITS برای رتبه بندی توابع استفاده می کند. در این بخش ما اهمیت و روش محاسبه مرکز و رتبه بندی قدرت را برای رویکرد خود شرح می دهیم.

برای یک ماتریس مشخص ، الگوریتم HITS دو ماتریس B1 = AAT و B2 = AT A را محاسبه می کند. ماتریس های B1 و B2 به ترتیب نقش مهمی در محاسبه رتبه بندی مرکز و رتبه بندی قدرت دارند.

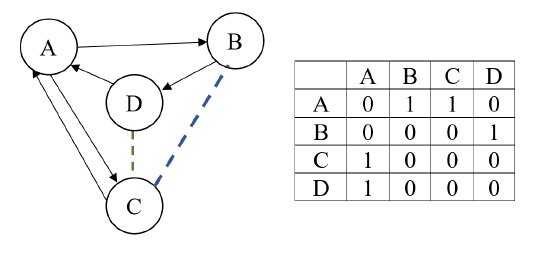


شکل 4.5: DSM از CSparse

شکل 4.6 نمونه ای از نمودار فراخوانی را نشان می دهد که برای آن دریافت می کنیم ،



بنابراین مقدار مورب ماتریس (B1) diag , B1 = [2؛ 1؛ 1؛ 1] نشان دهنده خارج از توابع و مقدار مورب دیاگرام ماتریس ,B2 (B2) = [2؛ 1؛ 1؛ 1] نشان دهنده عدم تابع توابع است به علاوه بر این در معادله 4.1 و 4.2 برخی از مقادیر خارج از مورب را مشاهده می کنیم (به عنوان متن پررنگ گزارش شده است). معادله 4.1 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 4.6). از طرف دیگر معادله 4.2 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 4.6) نامیده می شوند.



شکل 4.6: نمونه ای از Hub و Authority

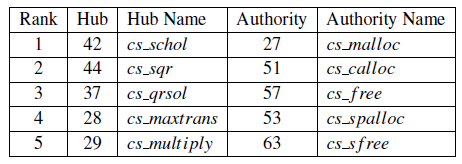
بنابراین ماتریس های محاسبه B1 و B2 رابطه وابستگی معنایی می دهد که در آن DSM فقط رابطه وابستگی نحوی بین توابع را ارائه می دهد.

فرض کنید ما DSM خود را از کد منبع داده شده که A است ساخته ایم. حالا ماتریس B1 = AAT و B2 = AT A را محاسبه کنید ، جایی که AT ترانسپورس ماتریس A است.

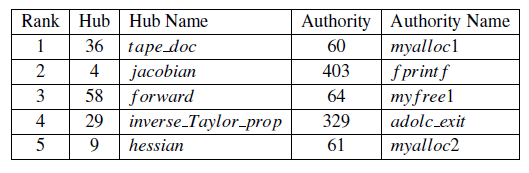
سپس با استفاده از Octave می توان ماتریس مورب λ از مقادیر ویژه و ماتریس V را پیدا کرد که ستونهای آنها بردارهای اختصاصی مربوطه هستند به طوری که Bi \*V = V\* λ جایی که Bi B1 و B2 است. ما به دنبال ارزش ویژه غالب λ هستیم که بیشترین مقدار λ را در خود دارد. ستون مربوط به V بردار ویژه غالب است که به عنوان مقدار مرکز (برای B1) و مقدار قدرت (برای B2) در نظر گرفته می شود [13]. ما توابع را به ترتیب نزولی با توجه به مقادیر آنها مرتب می کنیم (بخش 3.1 را ببینید) و از این رو رتبه قدرت و رتبه بندی توابع را بدست می آوریم.

در جداول 4.1 و 4.2 ما 5 مقام برتر و هاب را با توجه به مقادیر آنها برای هر دو نرم افزار CSparse و ADOL-C گزارش کرده ایم.

جدول 4.1: Hub and Authority رتبه پنج عملکرد اول پروژه CSparse



جدول 4.2: Hub and Authority رتبه پنج عملکرد اول پروژه ADOL-C



## 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 4.3 cos (Ө) (شباهت کسینوس) بین بردارهای a و b را نشان می دهد.



جایی که ||a||و ||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 a و 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 a و b انتخاب می کنیم.

در رویکرد ما DSM و ماتریس B1 = AAT (hub) و B2 = AT A (اقتدار) را برای انتخاب بردارها برای بررسی شباهت در نظر می گیریم. اهمیت ماتریس های B1 و B2 در بخش 4.1.3 مورد بحث قرار گرفته است ، جایی که ما نشان داده ایم که ماتریس های B1 و B2 اطلاعات مهم تری نسبت به DSM دارند. بنابراین ما رویکرد خود را برای هر دو نوع ماتریس آزمایش کردیم.

از رابطه 4.3 مشاهده می کنیم که برای محاسبه محصول نقطه بین دو بردار () مجموع حاصل ضرب ورودی های مربوطه دو ردیف یا ستون انتخاب شده را محاسبه می کند.

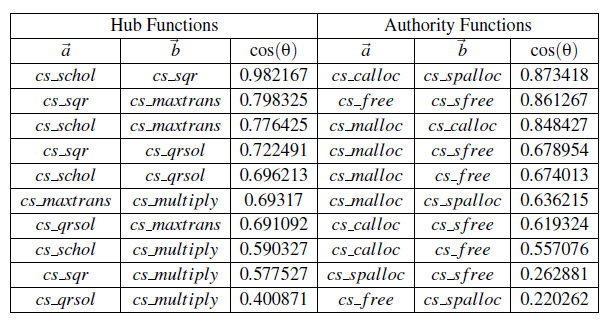
ما می دانیم cos (0ᵒ) = 1 و cos (90ᵒ) = 0. بنابراین اگر دو بردار متعامد باشند (مشابه هم نباشند) مقدار cos (Ө) 0 و اگر دو بردار موازی (مشابه) باشند ، مقدار cos (Ө) 1 خواهد بود. اما اگر مقدار cos (Ө) بین 0 و 1 باشد ، می توانیم آستانه ای برای تشخیص شباهت تعیین کنیم. ما الگوریتم خود را برای مقدار آستانه مختلف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

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.

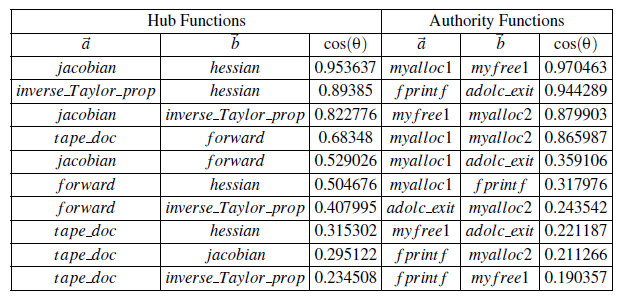
اکنون شباهت های کسینوس بین پنج تابع اول (با توجه به رتبه آنها به جدول 4.1 و جدول 4.2 مراجعه کنید) را با توجه به هاب ها و مقامات (برای پروژه های CSparse و ADOL-C) محاسبه می کنیم.

جدول 4.3 و جدول 4.4 شباهت های کسینوس بین توابع را نشان می دهد که در آن a و b توابع پروژه ها را نشان می دهند (پیوست A و B را ببینید). در اینجا مقدار cos (Ө) را بین 0 و 1 بدست می آوریم. بنابراین با تعیین یک آستانه می توان گفت که آیا دو تابع مشابه هستند یا خیر.

جدول 4.3: شباهت های کوزین با پنج عملکرد اول (با توجه به مرکز و رتبه اقتدار) پروژه CSparse



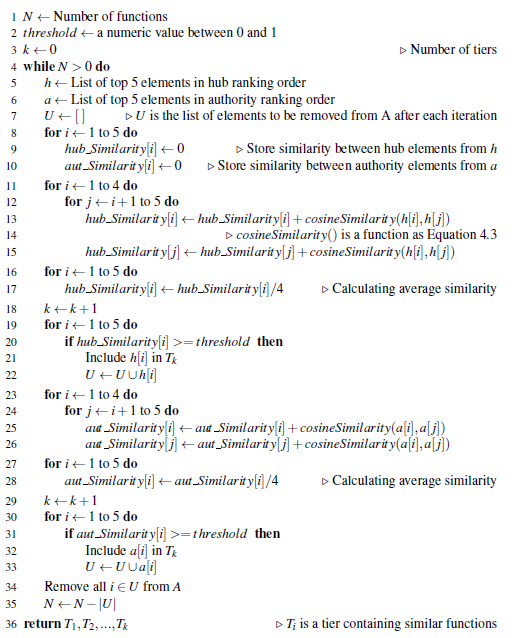
جدول 4.4: شباهت های کوزین با پنج عملکرد اول (با توجه به مرکز و رتبه اقتدار) پروژه ADOL-C



## 5 – ۱ – 4 الگوریتم

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

الگوریتم 1: گروه بندی توابع مشابه (DSM A)



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 0 تا 1 است. در بخش 4.1.3 ما نحوه محاسبه توزیع مرکز و قدرت توابع را مورد بحث قرار دادیم. با پیروی از روش مشابه در مرحله 5 ، یک لیست (h) از 5 عملکرد برتر را با توجه به رتبه بندی هاب دریافت می کنیم و در مرحله 6 یک لیست (a) از 5 عملکرد برتر را با توجه به رتبه بندی بدست می آوریم.

اکنون نحوه محاسبه سطوح هاب را توضیح می دهیم. در مراحل 11 تا 15 شباهت کسینوس بین این 5 توابع هاب را که در بخش 4.1.4 توضیح داده شده است محاسبه می کنیم. سپس میانگین شباهت هر یک از توابع هاب را در مرحله 17 محاسبه می کنیم. در مرحله 20 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 Tk 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 23 تا 33 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.

## 2 – 4 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.

## 1 - ۲ - 4 سیستم های هدف

برای آزمایش دو نرم افزار پیاده سازی شده در C/C ++ ,CSparse 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 [6].

از طرف دیگر ADOL-C یک سیستم نرم افزاری برای محاسبه مشتقات ریاضی (ضرایب گرادیان ، ژاکوبیان ، هسیان ، تیلور) یک تابع ریاضی است [12].

**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 [6]. این پروژه بسته ماتریس پراکنده قابل بارگیری است که الگوریتم ها و قضایای ارائه شده [6] 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 [6] ایده بگیرید.

توابع توسط نویسنده نرم افزار به شرح زیر طبقه بندی می شوند: ابزار اولیه ، اصلی ، ثانویه ، کاربرد ثانویه ، ابزار سوم و سوم [6].

**ADOL-C**

نرم افزار ADOL-C در /C++ C پیاده سازی شده است [12]. 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.

## 2 - ۲ - 4 انتخاب آستانه

در الگوریتم 1 ما در مورد یک آستانه (مرحله 2 در الگوریتم 1) صحبت کرده ایم که به تصمیم گیری در مورد اینکه آیا یک تابع باید در ردیف قرار گیرد یا خیر (مراحل 20 و 31) کمک می کند. مقدار آستانه باید بین 0 تا 1 انتخاب شود زیرا می دانیم که مقدار cos (Ө) از 0 تا 1 متغیر است. در این تز ارزش شباهت کسینوس (cos (Ө)) 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 (Ө) آنها نزدیک به 1 باشد دو تابع بیشتر شبیه هم هستند (توجه: اگر دو بردار مشابه یا موازی باشند به این معنی است که زاویه آنها 0ᵒ است بنابراین cos (0ᵒ) = 1). از طرف دیگر ، اگر مقدار cos (Ө) بین دو بردار نزدیک به 0 باشد ، آنها بیشتر شباهت ندارند.

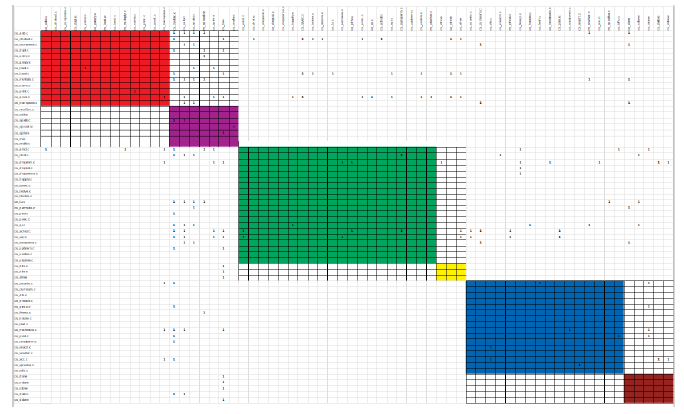
## 3 – 4 نتایج

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) [8] به دست آمده است. آزمایشات با استفاده از رایانه ای با پردازنده Intel Xeon 3.4 گیگاهرتز 8 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 4.8 ، 4.9 ، 4.10 و 4.11 گزارش شده است. در اینجا شکل های 4.8 و 4.9 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 4.10 و 4.11 نتایج را نشان می دهند که در آن AAT و ATA (به ترتیب برای هاب و اقتدار) برای انتخاب توابع برای بررسی شباهت آنها در نظر گرفته شده است.

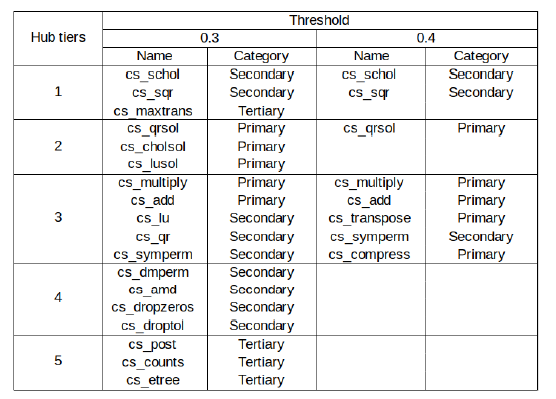
به طور مشابه نتایج آزمایش برای نرم افزار انتخابی ADOL-C در شکل های 4.12 ، 4.13 ، 4.14 و 4.15 گزارش شده است. در اینجا در همه موارد ما ماتریس AAT و ATA (به ترتیب برای hub و autority) را برای انتخاب توابع برای بررسی شباهت آنها در نظر گرفتیم. ما نتایج را به صورت جداگانه برای آستانه 0.4 = در شکل های 4.14 و 4.15 گزارش کردیم. اکنون نتایج مربوط به نرم افزار ADOL-C را مورد بحث قرار می دهیم. شکل های 4.12 و 4.14 نشان می دهد که برای مقادیر مختلف آستانه ، تعداد سطوح متفاوتی برای توابع هاب دریافت می کنیم. باز هم شکل های 4.13 و 4.15 نشان می دهد که برای مقادیر مختلف آستانه ، تعداد سطوح متفاوتی برای توابع اقتدار بدست می آوریم.

## 1 - 3 - 4 بحث

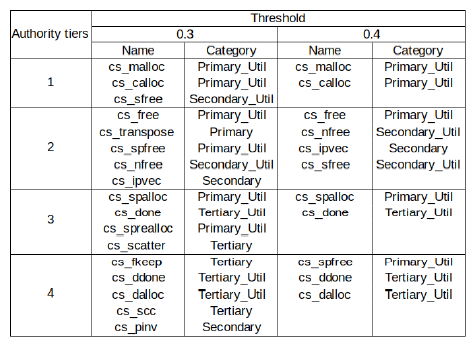


شکل 4.7: DSM از CSparse با پارتیشن های ارائه شده [6]. رنگهای اولیه ، اصلی ، ثانویه ، ثانویه ، ثانویه ، ثالث و ثالث با رنگهای قرمز ، بنفش ، سبز ، زرد ، آبی و قهوه ای مشخص می شوند.

شکل 4.7 DSM است که پارتیشن ارائه شده در [6] 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 [17] 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.

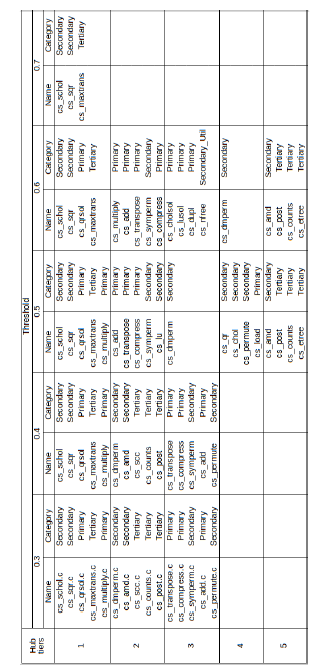


شکل 4.8: توابع CSparse ، Tiers of Hub از DSM انتخاب شده اند

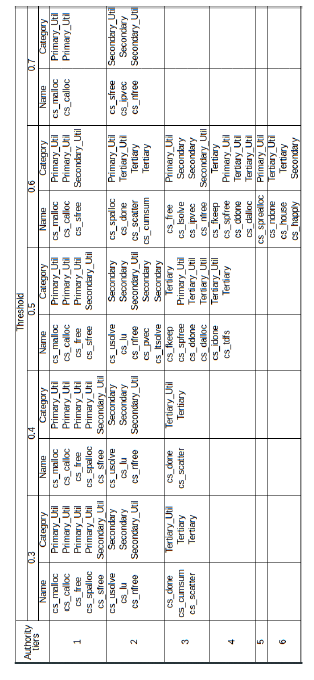


شکل 4.9: توابع CSparse ، Tiers of Authority از DSM انتخاب شده اند

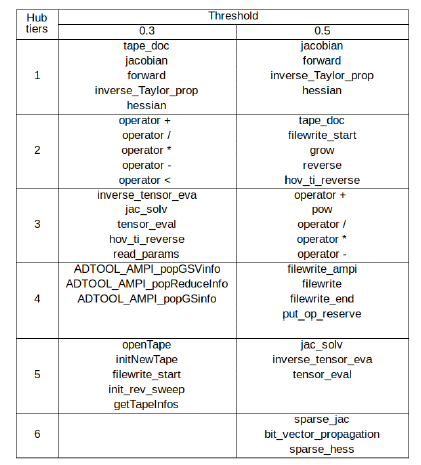
شکل های 4.8 و 4.10 نشان می دهد که برای مقادیر مختلف آستانه ، تعداد سطوح متفاوتی برای توابع هاب دریافت می کنیم. این ردیف ها همچنین با مقوله ای که در CSparse ذکر شده مطابقت دارد. به عنوان مثال در شکل 4.8 Hub Tier 5 شامل سه عملکرد است: cs post cs count و cs etree که در [6] به عنوان توابع درجه سوم طبقه بندی شده اند (شکل 4.7). به طور مشابه ، نمودارهای 4.9 و 4.11 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 4.9 مرجع درجه 1 شامل سه عملکرد است: cs malloc cs calloc و cs sfree که در [6] به عنوان توابع مفید طبقه بندی شده اند (شکل 4.7). بنابراین این روش تقریب خوبی از پارتیشن نویسندگان ایجاد می کند.



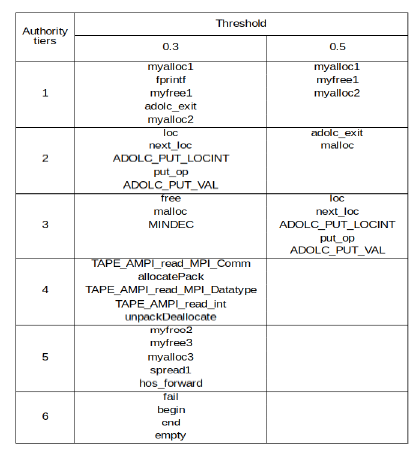
شکل 4.10: توابع CSparse ، Tiers of Hub از AAT انتخاب شده اند



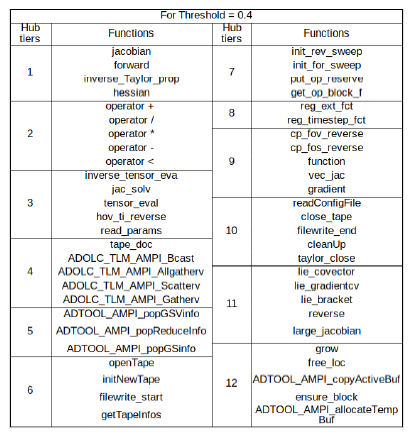
شکل 4.11: توابع CSparse ، Tiers of Authority از AT A انتخاب شده اند



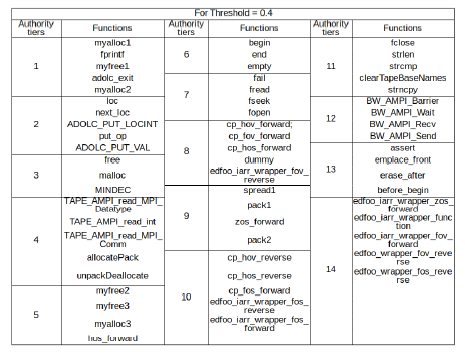
شکل 4.12: توابع ADOL-C ، Tiers of Hub از AAT انتخاب شده اند



شکل 4.13: ADOL-C ، توابع Tiers of Authority از AT A انتخاب شده اند



شکل 4.14: ADOL-C ، توابع Hub از AAT ، آستانه 0.4= انتخاب شده اند



شکل 4.15: ADOL-C ، توابع Authority از AT A ، آستانه 0.4= انتخاب شده اند

## فصـل 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 [13] ، 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 = AAT و B2 = AT 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.

يکي از دلايل اصلي که برنامه هاي توسعه نرم افزار در نهايت با شکست مواجه ميشوند، تخمين ضعيف اندازه نرم افزار است . اندازه عامل مهمي در تعيين هزينه ، زمان و تلاش است . تخمين حجم ، يک فعاليت پيچيده است ، که نتايج آن بايد به طور مداوم در سراسر چرخه حيات نرم افزار به روز شده باشد. اقدامات تشخيص حجم نرم افزار شامل تعداد خط کد منبع ، تعداد نقاط تابع و تعداد نقاط ويژه مي باشد. پيچيدگي، تابعي از اندازه است ، که تا حد زيادي تحت تاثير خطاهاي طراحي و نقص پنهان هستند که منجر به مشکلات کيفيت و افزايش هزينه ها خواهد شد. پيچيدگي را بايد به طور مداوم اندازه گيري، رديابي و کنترل کرد. معيارهاي اندازه گيري نرم افزار جنبه هاي مختلفي از پيشرفت و پيچيدگي نرم افزار را دارد در نتيجه نقش مهمي در تجزيه و تحليل و بهبود کيفيت نرم افزارخواهد داشت . معيارهاي نرم افزار اغلب به معيارهاي محصولات و معيارهاي فرآيند طبقه بندي 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.

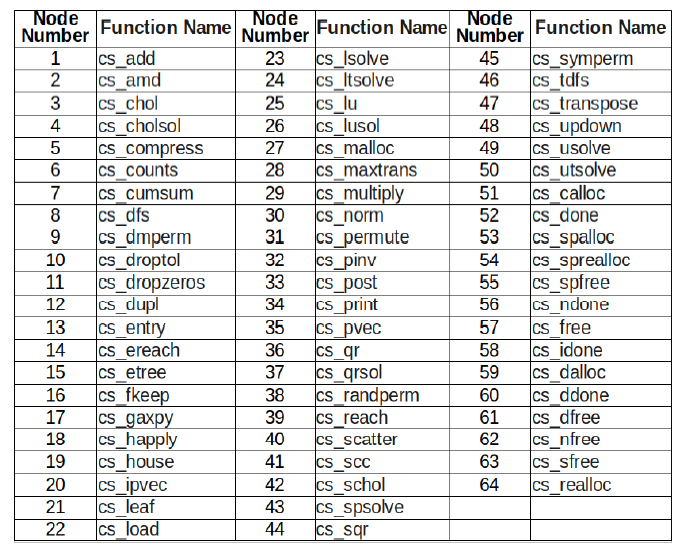
[21] Scientific Toolworks Inc. Scitools: Understand. https://scitools.com/.

[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**

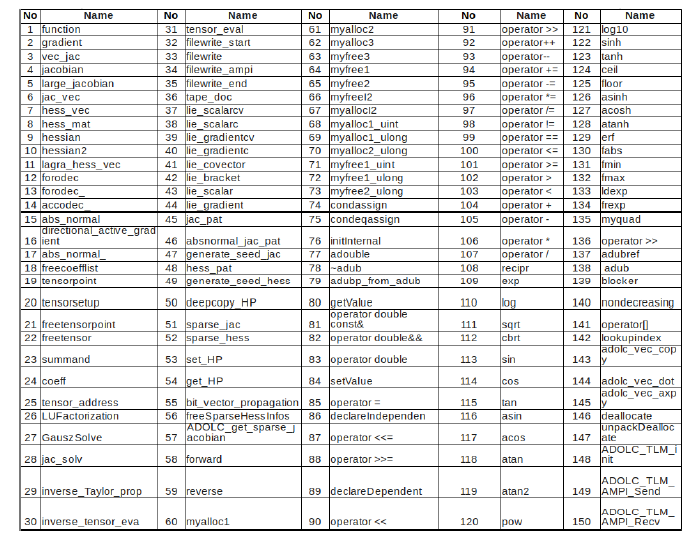
**لیست توابع CSparse**



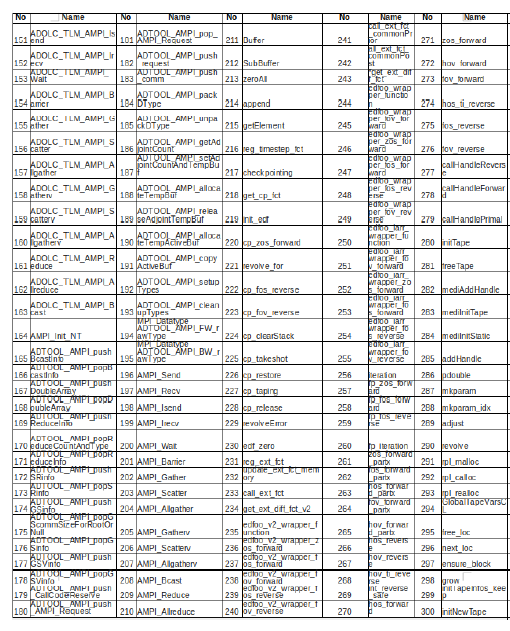
شکل A.1: فهرست توابع CSparse

**ضمیمه B**

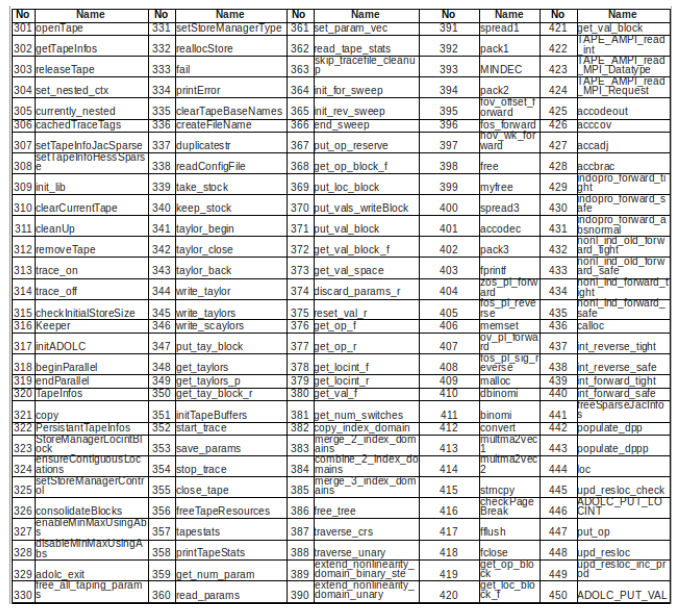
**لیست توابع ADOL-C**



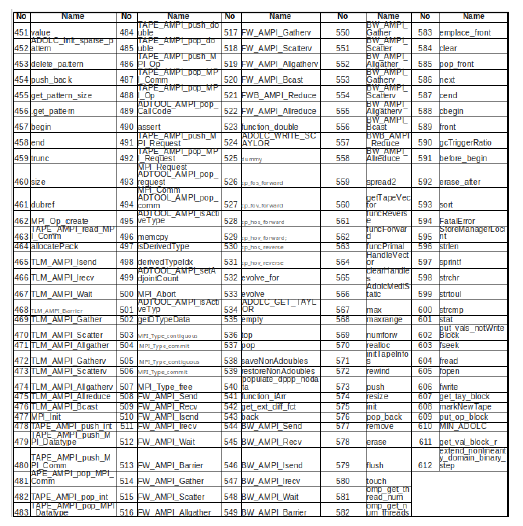
شکل B.1: فهرست توابع ADOL-C



شکل B.2: فهرست توابع ADOL-C



شکل B.3: فهرست توابع ADOL-C



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