**Software Complexity**

Based on Hypergraph

Technique Report

*Version 0.1*

**Wuhan University**

INDEX

[0 Preparation works 3](#_Toc353182743)

[0.1 Previous Work 3](#_Toc353182744)

[0.2 Our Research Route 3](#_Toc353182745)

[1 Modeling 5](#_Toc353182746)

[1.1 Definitions and Terminologies 5](#_Toc353182747)

[1.1.1 Software Complexity 5](#_Toc353182748)

[1.1.2 Coupling 5](#_Toc353182749)

[1.1.3 Cohesion 5](#_Toc353182750)

[1.1.4 Hypergraph 5](#_Toc353182751)

[1.2 Topology structure in hypergraphs 5](#_Toc353182752)

[1.2.1 Software Architecture 5](#_Toc353182753)

[1.2.2 5](#_Toc353182754)

[1.3 Metrics 5](#_Toc353182755)

[1.3.1 C++ Parser 6](#_Toc353182756)

[1.3.2 Coupling Metrics 6](#_Toc353182757)

[1.3.3 Cohesion Metrics 6](#_Toc353182758)

[1.3.4 Modeling 6](#_Toc353182759)

[2 Algorithm 7](#_Toc353182760)

[3 Coding 8](#_Toc353182761)

[4 Experiments 8](#_Toc353182762)

[4.1 Test Case 8](#_Toc353182763)

[4.1.1 Tiny Test Case 8](#_Toc353182764)

[4.1.2 Small Test Case 9](#_Toc353182765)

[5 Memorial 9](#_Toc353182766)

[6 Reference 9](#_Toc353182767)

|  |  |  |
| --- | --- | --- |
| Change Log | Update Content | Date |
| 1 | 4.1.2 | 2013/4/8 |
| 2 | Modeling | 2013/4/9 |
| 3 | Modeling | 2013/4/11 |
| 4 | Modeling | 2013/4/13 |
| 5 | Modeling | 2013/4/15 |
| 6 | Modeling | 2013/4/16 |
| 7 | Modeling | 2013/4/18 |
| 8 | Modeling | 2013/4/20 |
| 9 | Modeling | 2013/4/25 |
| 10 | Modeling | 2013/4/27 |
| 11 | Modeling | 2013/4/29 |
| 12 | Memorial/Modeling | 2013/5/3 |
| 13 | Modeling | 2013/5/6- 2013/5/7 |
| 14 | Modeling | 2013/5/9 |
| 15 | Modeling | 2013/5/20 |
| 16 | 0.2 research road | 2013/6/3 |
| 17 | 0.2 | 2013/6/4-7-13-14-15 |
| 18 | 0.2/ Memorial | 2013/6/18 |
| 19 | 0.2 | 2013/6/19 |
| 20 | 0.2 | 2013/6/29 |
| 21 | 0.2 | 2013/7/4 |
| 22 | 0.2 | 2013/7/7 |
| 23 | Memorial | 2013/7/9 |
| 24 | Memorial | 2013/7/13 |
| 25 | Memorial | 2013/7/16-2013/7/19 |
| 26 | Memorial | 2013/7/29 |
| 27 | Memorial | 2013/7/31 |
| 28 | Memorial | 2013/8/7 |

# 0 Preparation works

2013/7/24

Recent task list:

1. 找代码中解析方面的漏洞，并与黄和罗讨论。--魏小凤
2. 修改原有的代码，使解析更完善。--黄威
3. 预处理部分，重新思考架构问题，超图划分等。--罗永恩
4. 将预处理部分不以文件输出直接存于内存，给原有代码读取；编译器版本问题。--罗永恩
5. 内聚耦合度的计算，先至少计算出耦合度 （hMetis二分、树形结构的划分等） --魏小凤
6. 一周内至少新增两个testcase，大小不限制，并填写Excel表格，记录时间项。--罗永恩，魏小凤
7. 在代码中添加能输出5中需要的Excel内容，以txt形式输出，每行后面添加\t，便于直接复制到Excel表格中。 --黄威
8. 三个人经常讨论解析方面的问题，还有关于权重的问题。
9. 上述所有事情完成后，三个人一起接手张珊珊原来那块代码。

## Previous Work

Briand等人提出的内聚与耦合的度量模型

基于图模型的度量方法

李兵教授等人的基于复杂网络的软件复杂性度量

经典的软件复杂性度量方法：

根据软件的生命周期Halstead复杂性度量和McCabe圈复杂度度量都属于可以应用在软件测试阶段的度量技术。Halstead从统计学和心理学的角度研究软件复杂性把程序看成由可执行的代码行词汇(操作符和操作数)组成的符号序列。**Halstead**在其度量理论中采用一些基本的度量值来确定软件开发中的一些定量规律。这些度量值通常在程序产生之后得出或者设计完成之后算出。Halstead的重要结论之一是程序实际的Halstead长度值N可以由代码行词汇n算出。**McCabe**于1976年指出应该用程序流图的圈数(Cycloramic number)来测量程序的复杂性并基于程序控制论和图论提出了经典的**McCabe圈复杂性度量理论**。McCabe控制流图是一种简化的程序流程图，如果把流程图中的每个基本框抽象为一个点，略去每个框的具体信息，就产生一个由结点和弧或称为分支组成的图，称为控制流图。控制流图是有向图，可用G = <V, E>表示，其中V表示结点集合，代表程序流程图中的基本框，E表示有向边，代表程序流程图中的控制方向。

## Our Research Route

1. C++ parser: 使用C++ Parser分析得到语义语法等结果，筛选出对我们有用的语义或语法作为节点和边。
2. Coupling && Cohesion: Coupling 和cohesion的度量。
3. Graph model — Hypergraph model
4. 简单研究路线：重点在于超图建模的方案。
5. 确定一个简单的testcase，其中有包含类及类的调用等，可以进行手工parser，形成文件超图

A tiny test case: 包含2个类，5个函数，6个变量。顶点如下：

类A: v1, v2,v3, m1, m2

类B: v4, v5, v6, m3, m4, m5

以及类a, b

分别对应编号0-12.

其调用关系及超边的生成：类间调用关系暂时没考虑

1. 函数间调用

2. 变量间关系

3. 变量作为函数的参数

【testcase见代码，手工分析的超图文件见txt】

1. 产生超图的方案

已知：一段代码，包括类、变量及函数、函数的调用等。

顶点：变量、函数。(考虑一个类的情况，没有类-类间关系)

边：变量-变量( 变量出现在同一个函数内或是实现同一个功能，归为一条边)

函数-函数( 如一个函数调用另一个函数，则这两个函数所代表的顶点归为一条边 )

1. 超图建模代码思路：

将已有的手工parser结果，作为建立超图的测试数据。(testcase// invokerelationship.txt)。该文件的格式如下：

某个函数或变量名：

调用了哪些变量或函数

以”!”结尾

顶点部分：遍历这个输入的测试文件，找到单个的变量或函数(或类)，生成一个顶点，包括顶点编号及顶点内容(变量、函数及类名)，将生成的顶点以链表形式存储，用作后续建立边所需。将其输出到Vertex.txt中。

边的建立：遍历顶点列表，以顶点为依据，每一个顶点对应建立一条超边；同时，分析输入文档的调用情况，某个顶点若被其他顶点调用过，便将这个顶点加到以其他顶点为边的超边中，找到一个符合条件的便添加至新生成的边中，直至遍历完整个文件，将生成的边连成链表，输出到Edges.txt中

**注：**

目前已有的程序：社团检测ColluderDetection (多线程)，对检测社团的数据分析程序DataAnalyze。

已有的数据：hypergraph-cond-mat.txt。

对上述已有的程序及数据已经跑通，原来的单线程程序都没有备份。

**2013/6/3**

下一步：

1. 搞定多线程部分的内存泄露，完成多线程部分的文档说明。【罗永恩】
2. 构造规模稍大的testcase，慢慢修改程序（不要作太大改动）
3. 想一个计算体系结构的度量方法。即根据内聚和耦合度这两个标准构造一个目标函数，作为度量超图模块度的度量标准，对已有的系统评估其模块划分的合理性。
4. 可以根据超图划分的切片数来分析，(切片数—cuts：表示的是对于已有的超图，对超边进行切分时，被切超边的权重之和。即如果有两条权重分别为2和3的超边在划分时被切割，则cuts数为5.在超图划分中，cuts数越小越好，越小说明该超图中超边的凝聚度大，联系比较紧密。)
5. 内聚和耦合两个参数共同组成目标函数。比如，coupling/cohesion + 超图等因素；1/(coupling + cohesion)

2013/6/4

新构建testcase后出现的新问题，关于类间调用问题，判断标准的选取。原本的做法是根据类的对象，找由对象调用的函数属于哪些类。而当有构造函数出现，或是类指针等，问题变复杂，不能简单的使用原来的那个判断方法 （类名后面的字符串视为对象，显然有问题！！！）

2013/6/7

重新修改了寻找类的方法，将得到的类体也保存起来，类似于函数的处理方法。这样便于对象的查找。

2013/6/13 by loyoen

多线程部分：

除主线程外，另外开了三个线程，其中一个专门用于处理IO，另外两个用于处理超图建模部分。

为了实现方便，新建了一个CMgrJob的类，用于专门管理所有testcase，在IO线 程中循环去CMgrJob里取出需要IO的testcase进行IO处理，如果是input，结束 后，则将该testcase添加到建模队列；在建模线程中，循环取出CMgrJob中的 testcase，用于建模处理，处理完毕后，将该testcase添加到IO队列里，进行ouput 操作。

多线程的添加以及CMgrJob类的介绍：

第一， 将所有的testcase封装在CMgrJob里，CMgrJob维护两个testcase队列，一个用作 IO操作，一个用作建模。

第二， 新建三个线程，一个用于专门处理IO，另外两个用来处理建模。IO线程循环从 CMgrJob队列里取出testcase，调用testcase内部的读写过程函数，如果是input操作， 则在读完后，将该testcase添加到建模队列；建模线程循环从CMgrJob建模队列里 取出testcase，调用testcase内部的建模函数，建模完毕后，添加到IO队列进行output 操作。

2013/6/14

遇到的问题：在处理类的时候，需要明确类的结尾处，一个类的结尾若是没有其他字符的定义，则需要有空格，否则在读到字符};时会视为解析工作结束，那该类就没有被考虑到。

2013/6/15

根据社团检测中寻找超图最佳划分的方法，即模块度函数的定义。可以同理推导内聚和耦合度的函数。

例如，判断某一规模社团的划分结果，不仅仅要求它与外部联系少 ( 即面相对象程序中的低耦合 )，还要求内部连接紧密 ( 即面向对象程序中的高内聚 )。所以定义目标函数时考虑的情况跟社团划分类似。

2013/6/18

新增testcase的处理，代码修改完成。

新增处理有：

1. 多个类的情况，对类重新进行了解析。
2. 类指针的情况。
3. 类间关系的解析也重新处理。使用了map

备注：还有一个需要解决的问题，由typedef定义的类指针的处理。

根据类间关系推断耦合度，内聚度如何体现？（类内关系？）

2013/6/19

根据论文[1]中所述的内聚和耦合度的度量标准：

耦合度（couple）：

Data Abstraction Coupling (DAC): 类成员变量，方法数

Message Passing Coupling (MPC): 静态方法调用情况

Class-Method Interaction (CM): 方法形参，返回类型

Global Coupling (GC): 全局变量及其在表达式和参数中的使用情况

Inheritance coupling (IC): 继承情况及父类数

由上述各个部分组成，即在分析OO代码耦合度时，需要考虑上述五种情况。

内聚度（cohesion）：

|CI(c)| = |DD(c)| + |DM(c)| + |MM(c)| 各部分之和

|CI(c)| 是某一个类内部数据-数据交互、数据-方法交互、方法-方法交互数的总和

|DD(c)| 某类c中变量的相互关系，发生交互的变量数

|DM(c)| 某类c中成员函数与变量间交互的个数

|MM(c)| 某类c中成员函数与函数间产生调用等联系的方法个数

这里我们暂时只考虑了类间关系，即根据形成的超图计算类间耦合度。类与类之间联系越紧密，耦合度越大。这里在图中所体现的就是某几个点（由类组成）被划分到一起的概率是否大。（因为关系越紧密的点-类，越会被划分在一起。由于这里的紧密程度体现的是类间关系，所以对应耦合度）。类间关系无法体现内聚度，需要对类内成员的关联度进行分析。

我所理解的耦合度可以如下计算：

耦合度CO：设超图G = ( V, E )，V = { v1, v2, … , vn }, E = { e1, e2, … , em }。

【这里点集V对应每一个类， E对应类间关系。由于E的由来是基于每个类，找调用过它的类与其形成一条边，则每条边中存在的调用数为 #e -1 】

所以，耦合度CO =).其中 表示每一条边的基数，即每条边中包含的点的个数。#V表示点集合的基数，即所有点的个数。

计算调用关系时还要考虑权重。

2013/6/29

新增处理：

1. struct也算类
2. 排除以下情况：
3. typedef定义的类指针形成类
4. 类声明
5. 条件判断语句中可能包含带有类名的字符串

所以光找到class或是struct关键字是不可行的。还要符合类体或结构体的定义

2013/7/4

新增内容

1. 由typedef定义的类指针引起的类间调用关系
2. 以建模代码为例，提取代码骨干作为测试数据，已测试通过

存在的问题：

1. 读取文本内容出现问题：

在读取文本内容时，有部分内容无故读不进，还在调试中。

1. 在非类定义的文件中定义的类指针以及由其产生的函数调用关系未考虑。

【在查找类对象及类指针时，主要针对含有类定义的文件，所以如果在.cpp中出现了如在CEdge.cpp 的某个函数中出现LPVERTEX pVertex; ……. pVertex->output….等类似这样的语句，无法体现其调用性。 】

2013/7/7

readString函数有些问题：在处理注释问题上，以 // 注释起来的内容，所做的处理是将其所在的一整行都滤掉不予考虑，而会有下述情况出现：

1. 紧随某行代码或是某些变量的后边。如【int i; // 某某索引】或者【} // end for while】等
2. 输入某个路径时，会使用 ..//..//等

问题2：处理段注释时，是略过/\*….\*/之间的所有字符串，而当这种情况出现时：有可能我需要查找/\* \*/对，那么在代码中会出现，在这之间的代码都会不恰当的被忽略。

问题3：当一个函数调用中同时出现 . 和 -> 标识时，会混淆提取的对象和被调用函数。

问题4：如前面所述，关于局部变量产生的调用关系没有体现。

# Modeling

2013/4/9

对于4.1.2节中提到的问题：A被B调用，B被C调用，那么A也间接被C调用了，这里A和C是有关联还是无关联？这对建模生成边的时候有所影响，分割的结果是用作度量复杂性，是不是分割后被分在一起的顶点或边之间是关系比较密切的，然后提示这所代表的一段代码不能轻易修改？

分析：如果类A中有类B和类C，作为类A的数据成员，那么A和B、C也许有弱关联，也可能没有什么关系，最终要看函数调用情况，因为有可能在类A中只是声明了而没有实际使用过。在论文中可以举例说明，如某个球队类，它的类成员有球员、教练、洗衣机、清洁员等，球员与清洁工之间关系并不大。( 更好的例子？ )

类成员的属性？private? protected? public? friend?

超图模型建立后用分割算法进行分割，目标是看模块的划分是否合理，每个划分出的模块内部的关联强度，来判断架构是否合理。

数据结构的修改：添加一个数据结构relation，用于定义关系密切的类。

2013/4/11

上述问题已解决。

添加了一个数据结构relation，用于表示找到有关联的类之后，新形成的对象，将这个对象加入到CSimplex链表上。这样的做法保证了不复用原始的class类对象，从而不会出现所有有弱关联的类对象和有关联的类对象在一条大边上的情况。

新添加了两个mini test case，均测试通过

mini\_testcase\_1:

三个类A,B,C, 类A中同时出现类B,C的对象，而此时B,C不一定有关联( 需要继续分析函数调用的关系 )，B,C中未出现其他类的对象。仅对类而言，形成边是下述三条：

A, B->A, C->A

mini-testcase\_2:

四个类A,B,C,D, 类A中使用了类B的对象，类B中使用了C和D，类C和D均未使用其他类对象。对类而言，可形成下述边：

A->B, B->C->D, C, D

在程序中，给定的测试路径是mini-testcase\_2。

2013/4/13

parser函数的目标：只是为了进一步分析类间关系还是同时给出函数间调用情况？

函数解析部分：

思路1：遍历.h文件，找出其中所有的函数声明 (格式： 返回类型 函数名 ( 参数列表); )

然后到对应的.cpp文件中找它的函数定义，对函数主体进行逐行解析，查找其中使用其他类对象的部分 ( 以及调用其他成员函数的部分) 。

思路2：直接在.cpp文件中按照函数整体的定义来查找函数，后续同思路1

在4-13版的findMethod函数中查找函数的方法用的思路1.

进行了第一步，查找到.h 文件中的函数签名。

2013/4/15

在.h中找到某个类 (如classA ) 中对应的某几个函数 ( 如method\_1和method\_2 )，以map结构存储；再到.cpp文件中，找key-value 形式的 ( 即函数 )，定位到该函数定义处，对其函数主题进行解析，若在其他类的成员函数定义中出现classA的对象如cA部分，则该类和classA有关联。

需要记录的内容：在遍历.h时，需要记录类及其引用的对象，如classB中出现classA的对象cA，则记录为<classB, cA>，到classB的函数定义中找cA，若出现，则记录该函数及该类，如<classB, method\_3>

2013/4/16

继续函数解析：

1. 首先需要定义一个数据结构用于表示某个类中的成员函数调用了其他哪些类中的成员函数
2. 对每个类，遍历其.h和.cpp 文件，查找其中的成员函数。并记录该成员函数属于哪个类
3. 对当前类中每一个成员函数的主体进行分析，若出现调用其他类的成员函数，则将被调用的成员函数及其所属类存于1中所定义的数据结构中，并将调用者链接在其后。
4. 经过步骤3后，会形成调用表 ( 该表以什么形式存于哪里，map？ )，我们通过分析这个表，找出某一个函数曾被哪些类的哪个函数调用过，并将这些函数都关联起来( 是否重新定义关联关系？ )。
5. 建模标准：所有类的函数都是边，这些函数间的调用关系是点

图的对偶：类的函数作为点，函数间的调用即代表了类的关联，所以类之间的关联关系作为边

【成员函数间的调用，代表的是类间的关系】

注意：查找函数时，.h和.cpp文件中均要遍历，以函数定义的格式查找，有可能在.h中就直接定义了简单的函数。

问：是否考虑构造函数？

2013/4/18

继续函数解析：

1. 定义了数据结构stInvoke，想用来表示类的成员函数间存在的调用关系。每一个成员函数作为一个stInvoke对象，当发现该成员函数中调用了其他成员函数，则将被调用的也作为stInvoke对象加到当前stInvoke链上。后续反过来查找那些被调用的函数在哪些stInvoke对象中出现过。
2. 遍历了所有打开的文件，查找其中的函数定义部分，记录了其函数名，以及类中的对象名。包括了构造函数及主函数main。

用的testcase为名为small\_testcase文件夹中的数据

1. 下一步查找哪些函数调用了哪些函数。

2013/4/20

函数解析：

1. 首先对各个.h和,cpp文档遍历查找函数。都保存于Doc文档中
2. 每个函数作为一个Simplex对象，便于后续存调用关系
3. 解析函数调用的思路是直接以某个类的对象作为目标，在各个函数中查找该对象是否被某个函数调用，若调用，则以被调用函数为参数新建一个stInvoke对象，将该对象加到每个Simplex函数对象中。
4. 测试数据为small-testcase，函数调用的输出结果在文档methodInvoke.txt中，用 -> 表示了有调用关系的函数。这里的调用只表示了不同类间成员函数的调用，同一类中函数的相互调用没有管。
5. 还未处理新产生的内存泄露问题，后续处理
6. 未处理函数调用对类间调用的决定作用 （ 所以解析函数调用那块的返回值要重新考虑 ）

2013/4/25

在遍历文件解析函数时，读文件均用的CString自带的ReadString函数，得到的都为CString类型，在解析过程中，由于后续需要用到函数体的内容，所以按行解析的时候需要存储函数体的每行内容，直接将CString类型的字符串相加会不断申请新的内存，使得内存不够用。这里需要自己手动管理内存，所以我们用Windows内存管理函数GlobalAlloc和GlobalReAlloc函数进行分配及管理内存，存储我们所需的函数体内容。

使用CString自带的Append函数也可以实现。

2013/4/27

对函数解析部分所做的细节改动：

1. 在过滤注释这一块，重新定义了函数模块，只对有效的字符串进行后续解析

还考虑到了在函数的参数中出现注释的情况，这种情况，我们滤去注释，返回不含注释的字符串。

1. 将查找控制语句这块单独作为函数列出，便于扩展。

解析函数调用整体思路修改：

对于前面的解析函数调用情况，效率很低，因为我是以每个对象为查找基准，在每一个函数中都遍历是否有该对象调用该类中成员函数的情况。即对每一个类的对象进行分析，它被哪个类的函数所调用。这样，每个成员函数会被遍历多次，很多遍历都是无用功，(稀疏问题）。因此，我们反过来思考，针对每一个函数，解析函数体，分析这个函数调用了其他哪些类的成员函数，被调用的函数就加到以该函数为头节点的链表中，这样就得到每个函数的调用链表。再通过查找这个调用表，获得某个函数被其他哪些函数调用的结果，这些处在同一个调用关系链上的函数所对应的类在一条超边上。

区别： 某个函数调用了其他哪些类的哪些函数。

某个函数被其他哪些类的哪些函数调用过。

这样引出**对偶超图**的概念。

2013/4/29

已找出所有函数的调用关系，以及根据函数调用关系找到的对应类的调用关系

这种调用关系为：某个类调用了其他哪一个类，根据该调用关系，可以知道某个类被其他哪些类调用了，而形成边。

到目前为止，parser部分还有需要待完善的地方，一个文件定义中多个类的情况需要考虑。调用和被调用关系这块，只得到调用关系，而建模生成边是根据被调用关系得到的。这里的转换可以在parser完成，也可以在建模时实现。

界面操作流程：

O – 打开文件

C – 得到类及其对象，得到函数

M – 函数解析

R – 类间关系

2013/5/3

接下来要做的事：

1. 修改函数调用的存储结构，以便于将得到的函数调用关系。即由调用关系得到被调用关系。
2. 根据调用关系形成超图
3. 添加测试开关
4. 记录每次提到的在testcase未考虑到的异常，即特殊情况
5. 批处理测试，多个测试文件一起自动连续的跑完。（可将所有测试文件路径存放于一个文本文件中，读取该文本文件获得）

2013/5/4

解决事件1.

解决方法：在遍历每个函数，得到该函数调用了哪些函数时，将那个被调用的函数作为一个调用关系的指针头，将调用该函数的函数指针串在被调用的后面，解决了哪些函数被其他哪些函数调用的问题。

存在的问题：对于一个函数被多个函数调用的问题还未解决，下一步解决。

2013/5/6-2013/5/7

之前提到的多个函数调用问题已解决

超图建模的思路：

1. 每个类作为一个顶点 （在MainFrm中进行操作，新增顶点生成按钮，该按钮实现的功能是将类链表中的每个类对应生成顶点，做法是遍历m\_pHeadCl链表内容，以每一个LPCLASS对象建立一个CVertex对象，将得到的顶点链接起来）

注意，从生成调用关系开始就没有处理内存泄露

1. 每个类间关系形成一条边 （ 新增了一个数据结构用于表示有关系的顶点，原理同寻找类间关系）
2. hMetis分割的超图格式：

边数 顶点数 类型

（类型：不写为空，均无权重；1 超边权重顶点无权重；10 顶点由权重超边无权重；11 均有权重）

边的权重 边的顶点

顶点权重列表

2013/5/9

新增TestCase类用于实现批量自动测试，后续会增加多线程测试功能，实现多个线程同时跑测试代码，生成超图

TestCase类主要封装了两个功：

1. 完成代码解析，即parser的过程。该过程原来在Doc和MainFrm里面实现的，现在直接在该类中使用Doc和Mainfrm的对象调用这些已实现的函数，如果要考虑耦合性的话，可以将这些函数全部在TestCase类中实现。
2. 完成超图建模过程，及modeling。该过程原来全部在Mainfrm里实现，同上，直接调用

在new TestCase类之前，需要做的事是将所有testcase路径放在一个文本文件中，通过按行读取该文本文件获得每个测试文件的路径及名字，并到该测试路径下去遍历每一个文件进行后续处理。

多线程部分交给罗永恩。

目前还存在的问题：

内存泄露未处理，导致随着testcase的增加堆栈内存会变得不够用。( 这个问题亟待解决 )

输出中间结果时，最后一个testcase的输出内容会覆盖之前的结果，即最后在output输出中看到的只是最后一个testcase的内容。但是全部内容可以通过testcase对象链表获得

2013/5/13

目前的做法：

1. 每个testcase中的每个.cpp和.h文件分别在MFC界面的Doc文档中处理的，在该文档中实现了每个.h和.cpp文件中的类及函数查找。并对每个Doc新建一个对象，将该对象存在了vector中便于后续parser使用。
2. 将原本在mainfrm中实现的parser和modeling过程，均封装在了TestCase类中；同时封装了读取测试文件以及输出分析结果。
3. 主函数在App中，包括从配置文件列表( 该文件包含的是所有testcase的文件路径 )获得所有testcase，并为每个testcase新建一个对象，用该对象对每个testcase进行操作。
4. 有个问题，这样写，不利于内存泄露的处理。

新的思路：

完全不用MFC界面，将上述第一条叙述中提到的，在doc中进行的处理，可以重新封装成另外一个类，比如TestFileDeal。在TestCase中建立TestfileDeal的对象链

主函数使用main函数实现。

其他不变

2013/5/20

所作改动：

1. 将原本使用MFC界面所带的Doc实现的功能，重新封装在了新类中，如2013/5/13新的思路中所述。该新类为CTestFile，在该类中实现的功能同原本Doc。
2. 原本使用的CString进行字符串解析以及使用MFC所封装的读文件对话框以及遍历查找文件等等所有跟MFC/CString相关的内容全用C++及string所替换。
3. 将所有的内存泄露解决掉了
4. 迁移到Linux下 ，实现双重版本。( BY 罗永恩 )

【在修改后的工程中有几处，如读取文件及输出文件部分，用到了windows提供的API，Linux下不能使用，需要重新在Linux小实现，并保存原始版本】

## Definitions and Terminologies

### 1.1.1 Software Complexity

### 1.1.2 Coupling

### 1.1.3 Cohesion

### 1.1.4 Hypergraph

## Topology structure in hypergraphs

### Software Architecture

抽象数据结构的定义思想：

CSimplex类，作为软件架构中最基本的拓扑要素，是一个单纯型类。它是对变量、函数及类的共性与关联的一个抽象，即一个变量或函数或类，对应一个CSimplex对象。同时，在CSimplex中还定义了这些抽象“对象”间的关系。

此时，每个CSimplex对象作为超图建模中的顶点，根据CSimplex对象间关系形成边。

### 1.2.2

## Metrics

Coupling && Cohesion的典型度量方法是Briand等人提出的面向对象设计中内聚与耦合，基于各种抽象概念，如类继承、类类型、函数调用、类与属性间的引用等，分析内聚与耦合度时，将代码按其语法进行解析，得到相应的代表概念性的关键词，累计这些概念关键词的出现情况，据此分析代码间的联系。

Coupling && Cohesion的另一种度量方法，根据论文Measuring Coupling and Cohesion: An information-Theory Approach的启示，不仅仅使用简单的累加方法，而是根据各个节点与边之间的信息，求其信息熵。将节点与边组成的简单图看做是一个子系统，子系统中含有子模块，对每一个子系统分析其intermodule coupling 及intramodule coupling， cohesion就在intramodule的基础上求得的。具体求法。。。

面向对象的软件度量方法，MOOD方法，主要聚焦于：

类的内部复杂性 (如类的加权方法数、类内聚缺乏度、方法隐藏因子、属性隐藏因子)

类之间继承的复杂性 (继承树深度、继承方法数比例)

类之间的耦合复杂性 (对象类之间的耦合数、所有可能关联类对的数目)

### C++ Parser

使用工具ANTLR解析得到grammar files，扫描grammar files，对含有相应的度量关键词进行累计，直到文件最后一个字符停止。根据论文”A Coupling and Cohesion Metrics Suite for Object-Oriented Software”我们可以得知，源代码解析得到的grammar files中，可以选出哪些度量参数是对我们的研究有用的。

解析工作进展：

解析目标：由度量需要的关键词而定 (1.3.2和1.3.3节中可筛选)。

### Coupling Metrics

Data Abstraction Coupling (DAC): 类成员变量，方法数

Message Passing Coupling (MPC): 静态方法调用情况

Class-Method Interaction (CM): 方法形参，返回类型

Global Coupling (GC): 全局变量及其在表达式和参数中的使用情况

Inheritance coupling (IC): 继承情况及父类数

### Cohesion Metrics

|CI(c)| = |DD(c)| + |DM(c)| + |MM(c)|

|CI(c)| 是某一个类内部数据-数据交互、数据-方法交互、方法-方法交互数的总和

|DD(c)| 某类c中变量的相互关系，发生交互的变量数

|DM(c)| 某类c中成员函数与变量间交互的个数

|MM(c)| 某类c中成员函数与函数间产生调用等联系的方法个数

### Modeling

1. 对给定的代码进行分析，解析得到我们所需的特定“节点”。Parser的过程及是否得到了所需的结果？

2. 节点的选取：变量，函数，类，形参。

3. 边：参数、变量、函数等调用情况。

模型建立后，定一种metric分别度量其coupling和cohesion

最简单的一种模型，只考虑变量、函数两种节点以及各个模块间的调用情况。每个变量和函数分别作为节点，一个函数和它的参数变量作为一个模块，模块间可能存在相互调用的情况，形成一条边。

后期可以考虑代码的重用性(如最简单的重用，使用类的对象，将类的对象放到一个新的类里面)、全局变量、继承、变量与变量间的联系等。

我的想法：通过解析，得到各变量及个数和函数个数，这样能得到节点情况。关于边的计算，解析能否得到函数调用情况？如果能，则采用这种方式，将具有调用关系的函数及变量归为一条边内，调用一次则累计其调用次数作为超边权重，用作后续分析其耦合度等。

面向对象C++源码数据形式的超图建模：

源码数据的超图形式：

根据给定测试源码，我们有一个变量集V，一个函数集M，一个类集C以及变量函数类等各自的调用情况，我们可以通过这四者构造变量超图、函数超图以及类超图。有如下定义：

— 变量超图：

顶点 — 每个变量就是一个顶点。(类成员变量，局部变量，参变量)

超边 — 在某一段代码内，存在计算上的或是有相互影响关系的 ( 比如它们为实现同一个功能或者出现在同一个函数内 ) 变量集构成一条边， 不管它们出现在类成员中还是函数中(包括局部变量及参变量)。

权重 — 暂时不考虑变量的权重对程序整体复杂度的影响，为了简化，顶点和超边的权重都默认为1.

— 函数超图：

顶点 — 每个函数就是一个顶点。

超边 — 在某一段代码内，含有相同参数变量或存在调用关系的函数集构成一条边。

权重 — 此处权重表示函数调用其他函数的程度，若调用的函数越多，权重越大，表示对整个代码影响越大。为了简化处理，首先全部默认为1。

 — 类超图：

顶点 — 每个类作为一个顶点。

超边 — 类与类间存在某种关系( 这种关系可以理解为继承、引用对象等 ) 构成一条边。

权重 — 暂时默认为1

# Algorithm

Parser思路框架整理：

定义一个类 (名字为CSimplex)，该类至少包含三个类指针，分别指向变量(simplexAttribute)，函数 (simplexMethod)，关联关系 (simplexRelations)。遍历每一个.cpp或.c文件，每得到一个变量，或者函数，分别加入到变量或函数的指针链中，根据这两个链得到第三个关联关系链，形成三个链表。其间会需要其他辅助信息，如每个链表的下一个链表next，要区分开，以及总的指针类的next指针。（将得到的链表在一个map中维护？）

第一步，遍历.h文件，得到所有的类和函数，存起来。

第二步，寻找类间关系，将得到的关系作为CSimplex的元素

主要是要做一个parser，解析C\C++语言，不需要关心语言的特征或是语法什么的，只关注类与类之间、函数与函数之间的关联(如调用关系)。我们的分析目标可能是linux内核，也可能是其他大型的工程代码，最终目的是为了分析它们的架构复杂性。parser这部分的主要目的是得到这段代码包含了哪些变量函数等以及它们之间的关联关系，用于我们后来建模做准备.

parser的目的，是为建立超图作准备工作的。我需要有顶点，根据顶点间的关系生成边。

所以此处，我们对应到parser的目标中来，从整体来考虑，我们的顶点来源于类，即一个类作为一个顶点，

边来源于类之间的关联关系( 这个关联关系通过分析类中函数或变量的调用关系来实现，也包括类与类之间的继承关系 )

如果类之间有关系，则我们把有关联的类归为一条边，这是整体的大目标。

所以，在parser的时候，需要得到工程中有哪些类，记录下来，这些类之间有没有关系，将有关系的类串起来，（如果能记录下它们通过哪些函数关联的更好，但这不是最重要的）。

需求理解：

1. 分析C++文件

2. 不完全解析C++文件，可以只关注类名，关于类之间的调用关系，我们需要遍历类中的方法，在其方法实现中寻找是否会使用其他的类。如果找到，那么则表示存在这么一个关系，记录下来。

3. 不需要关心方法中的分支、以及循环语句，只要使用了其他的类，那么就记录

4. 可以以类为单位进行分析

# Coding

# Experiments

## Test Case

### Tiny Test Case

定义了两个类A,B，其中，在类B中使用了类A的对象，于是，我们有A->B这样一个类间关系。

操作流程：

o toolbar：打开文件，到testcase路径下

c toolbar：寻找class并找到class的关系

将得到的类及其关系输出在output文件中。

输出的结果已经相当于一个小型的超图原型。（classes.txt中的结果作为vertices， simplex.txt中的结果作为edges）

后续进行建模时，这些中间输出结果可以不需要 (使用调试开关)。

### Mini Test Case

Small\_1： 定义三个类A, B, C，实现这样的类间关系 A->B, B->C.

操作流程同4.1.1，得到的结果也输出在output文件中。

问题：A被B调用，B被C调用，那么A也间接被C调用了，这里A和C是有关联还是无关联？这对建模生成边的时候有所影响，分割的结果是用作度量复杂性，是不是分割后被分在一起的顶点或边之间是关系比较密切的，然后提示这所代表的一段代码不能轻易修改？

Small\_2： 定义三个类A, B, C，实现这样的类间关系 A->B, B->C, C->A

这种类间循环调用情况无法实现，由于B中include了A需要先编译A，而A中又include了，需要链接C，而C又include了B….陷入死循环。所以这种测试情况没有给出。

# Memorial

1. Parser的过程中，遇到库函数，当做global的“变量”，谁包含它，就引一条链，权重置为0，表示其关联强度为0.（有可能会成为大边？）
2. 准备工作：解析makefile文件，得到其中所包含的.cpp, .c或.h文件
3. 建模时，对偶超图 ( 函数和函数间调用关系形成对偶关系 )。对偶超图的作用，使得超图分割更容易。
4. 考虑类间关系时，未考虑类的继承
5. 对于解析函数调用关系这块，还未考虑到静态函数(static)的情况
6. 查找类的过程还不够完善，很多情况未考虑，比如使用对象指针。只处理了以类名作为依据所生成的对象进行的调用等
7. 结构体算不算类？结构体也算类。[已考虑]
8. 当一个类被调用的次数很多的时候，要加权重，当一个类占的空间很大，也要加权重。
9. 由typedef定义的类指针的情况。也涉及到类间关系 [已考虑]
10. 某个类自己调用自己的情况，如形成链表时，调用下一指针。
11. 邻居关系，一邻居二邻居等，边界算子
12. 模块度 （local及global）、梯度算子
13. 面向服务的软件架构问题。

2013/7/9

目前已实现的功能小结：

解析部分

1. 查找类
2. 查找函数
3. 将函数对应到其所在的类中
4. 查找类的对象及类指针变量 (包括由typedef定义的类指针)
5. 查找函数间调用关系，通过分析每个函数调用了哪些其他类的函数，得到某个类的某个函数被其他类的函数调用。输出methodInvoke.txt
6. 通过类间函数调用关系得到类间调用关系。输出simplex.txt

建模部分

1. 根据所得的类，生成顶点。输出 vertex.txt
2. 根据顶点间关系，即类间调用关系 (若某个类A被类B调用，则形成A->B这样一条边)。输出edges.txt
3. 根据上述所得，可形成超图，由点和边的序号所表示，所有权重暂时默认为1. 输出Hypergraph.hgr

上述已基本实现，后期所修改的都是解析部分的未考虑到以及出现的错误。

如由局部变量引起的调用

2013/7/13

To do:

1. 加大testcase规模。
2. Modularity的定义，需要和老师一起讨论。（目标函数，期望值，最优化技术）
3. Coupling和cohesion的计算。

注意分割的终止条件，形成分割树。

最大目标：分析大型软件系统架构的有关描述参量。近期可以在架构上出一些参数相关的文章，如coupling和cohesion以及modularity. 超图适合解决弱关联的问题。

对于模块度：

不仅要定义模块度，还有与其匹配的如局部模块度，对应梯度算子。

如global modularity和local modularity

此外，还要考虑邻居关系，一邻居，二邻居等，这些都可以归为边界算子。

将来要考虑的工作，面向服务的软件系统架构。

2013/7/16-2013/7/18

由新的testcase产生的新问题：

1. typedef定义的类指针处理不够完善，只解析了一种定义，可能有多种对类指针的定义。
2. 对于一行没写完换到下一行的做法在解析时没有考虑到，暂时还没发现有什么影响，对我们分析有用的代码尽量不使用换行写，如果遇到怎么办。
3. 带继承的类。在查找类的过程中，只针对了单个类的考虑，对于这个类在定义时出现父类，没有做处理。
4. 查找类的对象时，需要排除将类作为返回类型的函数的情况或者类作为函数参数类型的情况。
5. 还是解析中 { 的问题，根据不同编程者的习惯，{ 的出现位置，分为第一行末尾或是第二行起始，我在处理的时候都作为第二行起始来分析的。
6. 在分析类间调用时，主要是分析类间成员函数间的调用关系，这时，成员函数的返回类型或是参数类型也会对类间关系有所影响，这点在解析时也没有考虑到。
7. 在搜寻CMU站点上的开源代码时，大部分都是基于Linux，只给出include文件以及Makefile等，没有源文件。有源文件的部分大都是用C写的，没有类这些概念出现。
8. 标准模版类的使用，会导致查找类结果有偏差，如template <class T>，会错找出类T>
9. 在变量中可能会出现class或struct等关键字，也可能会误作为类。 （这个在parser的代码中写，查找时，确保class 和struct是单独的关键字）
10. 类名为空格的情况，如语句 typedef struct ;
11. } } 这种花括号在同一行的情况以及花括号对在同一行的情况，对找类或函数都有影响，对后续的类或函数都有影响。
12. 没有换行符 \ 直接换行的情况 【不一定是有小括号对的中断换行，还有其他的的情况，如一个函数声明太长】
13. 【非预处理的问题】
14. 查找函数调用的时候会出现的问题：

在return语句中出现函数调用会将return与调用者一起作为调用对象。

有多个->或是 . 的时候，无法确定是谁调用谁。以及在函数调用语句处有多个()对出现，无法确定函数名。

解决的方法：在找到出现函数调用的语句处，确定是哪个对象调用了哪个函数名，而不能直接将->或是 . 左边的所有字符都当做调用者的名字。以及不能把 -> 和 ( 之间的字符串作为被调用的函数名。

1. 类间调用关系的因素：

友元类的出现算不算类间有调用关系？

1. 继续问题 宏函数定义怎么算？

例如：

#define DECLARE\_OPEN\_HANDLE(From, To)

OPEN\_HANDLE\_LIST(DECLARE\_OPEN\_HANDLE)

#undef DECLARE\_OPEN\_HANDLE

当做函数对我们的解析调用关系目前无影响。

1. 构造函数后续冒号处进行变量初始化时，用的格式为 变量名 ( 值 ) 这种形式会被当成函数处理

**2013/7/19**

版本1所实现的功能小结：

1. 预处理。待完善

将对我们后续解析有影响的以及没有用处的注释等代码清空。

1） 注释（行注释，段注释等）

2） 模版类的定义 （对找类有影响）

3） 双引号及单引号的清除 （对解析有很大影响）

4） 花括号的位置问题 （顶行及换行的不同风格，无法统一，通过预处理统一放在下一行）

5） 换行问题，一行没写完换到下一行的情况对解析语句有影响，将换行的语句写回到同一行。

1. 读操作
2. 解析testcase 待完善
3. 查找类
4. 查找函数
5. 查找类的对象
6. 函数映射到相应类中
7. 查找函数间调用关系
8. 根据5）得到类间调用关系
9. 超图建模
10. 顶点
11. 超边

5. 写操作

建模得到的超图孤立顶点太多，是由于在解析时只考虑了类成员变量所引起的调用，而没有考虑函数内部局部变量引起的调用，以及其他，在coupling相关的那个文档里有写引起耦合的各种因素。

进一步寻找调用关系是下一步所要做的。

**2013/7/29**

1. Cohesion的粒度问题，分层？(cohesion体现在类内部成员函数之间的调用)
2. 考虑调用关系时，继承问题。包括两个父类的问题。( 由子类追踪到父类 )

父类和子类的内含程度，例如MFC中，所有类都直接或间接继承自CObject类

1. 参数传递、函数返回类型等
2. 对于函数指针这块暂时完全不考虑。
3. 对于OO论文中提到的所有内聚耦合度的计算在我们代码中要有所体现。

**2013/7/31**

所讨论到的问题：

1. 对于coupling和cohesion的公式定义的表述部分，使用“程序集的子集”概念来表述；以及关于cohesion相加的问题，可以以cohesion的性质给出。
2. 【在分割的过程中，不仅考虑到coupling和cohesion，还要考虑balance平衡因子，这里balance所指的是分割的块间顶点数的差别。例如一个超图被划分为两块时，这两块分别含有100和10个顶点，而这两块的coupling和cohesion已经达到很优，若只根据这种情况，则会继续划分10个顶点的块而不是100个顶点，这样会使得划分结果出现有的块很大有的块很小的情况。(这样的系统在设计时是不太合理的。) 考虑到balance，就会选择分割100顶点的块。】删。balance后续再考虑

在hMetis分割过程中，按照树形结构去分，以规模数作为终止条件 (如分到某个块顶点少于20就停止)。 分割时，每一次分割，将coupling和cohesion的公式代入，计算每个块的cohesion以及每两块间的coupling，到分割终止时，选取一个最优的分法。这样做的目的是比较直观的看到coupling和cohesion的作用，同时得到结果后，可以便于进一步思考modularity的定义。

1. 分割过程中，在上一层被切分过的边，其权重赋为0，到下一层切分时不算在内。【论文中不要提到，实现时可考虑】
2. 每一次分割，保存父超图的结构，而不重新生成超图，效率问题，暂时不考虑。
3. 在解析调用关系时，对于函数重载的情况，已经考虑过。
4. 针对解析时间效率这一块，用哈希结构，跑三个以上testcase，寻找时间增长情况。
5. 在解析过程中，能应对错误情况的发生，比如同一个类中出现同一个函数名的情况，在代码中能忽略其中一个，而不是出现代码崩溃的现象。

2013/8/2

1. Merge问题。对原始超图不断进行二分，分割到某一规模停止，将分成的许多小块进行合并，使得最后分成的两部分其coupling和cohesion达到最佳效果。在这个过程中，每一次分割后我如何剪枝，以便后面进行回溯。这个决策过程后续再考虑。(如何保留种子问题)
2. 哈希问题。
3. 丰富testcase。在testcase添加一些可能出现问题的代码，并给出注释等。包括常见情况或是特殊情况等，用于诊断代码的健壮性、架构的准确性以及算法。

2013/8/7

讨论调用关系如何处理以及权重等问题列表：

1. 再一次确定某个类被其他类调用时，该类所在顶点与其他类所表示的顶点组成一条超边。
2. 在解析类间关系时，保存了每一个类调用其他类的次数。比如有A,B,C,D四个类，类A分别被B,C,D三个类分别调用了2,3,1次，那么这条超边的权重如何确定？是取最大值还是平均值。还有一个想法是，如果B,C,D调用了A的次数差别比较大，那么选出调用次数大于某个阈值的点重新作为一条超边，原始超边还是不变。
3. 对于顶点的权重，我们考虑了某个类的成员变量。如果类A中出现的成员变量类型为其他类类型，则类A所代表的顶点的权重即为这些类的个数之和。如类A中出现类B,C,D三个类类型的成员变量，则顶点A的权重为3.

# Reference

[1] Sukainah Husein, Alan Oxley: A Coupling and Cohesion Metrics Suite for Object-Oriented Software. IEEE 2009 International Conference on Computer Technology and Development, pp.421-425, 2009

[2] Edward B. Allen, Taghi M.Khoshgoftaar: Measuring Coupling and Cohesion: An information-theory approach.

[3] L. C. Briand, J. W. Daly, and J. Wust, A unified framework for coupling measurement in object oriented systems, IEEE Transactions on Software Engineering, vol. 25, pp. 91–121, 1999.