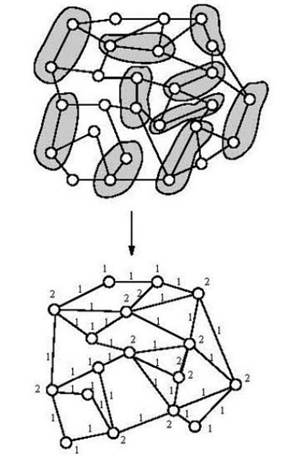
**METIS接口文档**

李仲萱

**1.  METIS图分割简介**

METIS是由Karypis Lab开发的一个具有强大功能的图切分软件包。准确来说，METIS是一个串行图切分的软件包，Karypis Lab还提供了并行版的图切分软件包parMETIS和支持超图和电路划分的hMETIS。METIS的算法设计主要基于多层次递归二分切分法、多层次K路切分法以及多约束划分机制。用户使用METIS软件包时，可以根据需要选择相应的切分方式。

METIS的工作原理：以k路多层次划分为例。

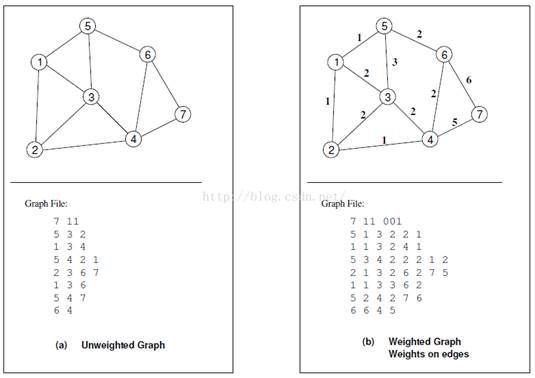


整个划分过程分成coarsening, initial partitioning 和 uncoarsening (refinement) 三个部分。coarsening将图的大小逐渐缩小，G1->G2->G3-G4。在G4阶段执行K路划分，然后在uncoarsening阶段将图中的原始节点映射到G4划分的cluster中。

METIS的安装：用户需要到Karypis Lab的官网下载最新的METIS安装包，目前最新版本为METIS 5.1，然后解压软件包。需要保证系统上已经安装了C编译器和GNU make和C Make 2.8，根据操作系统的位数修改metis.h文件中的IDXTYPEWIDTH常量的值为32或者64，然后在metis底层文件下执行make指令，既可以完成安装。

METIS的使用：以gpmetis为例，其使用方法为 gpmetis [options] graphfile nparts。gpmetis是编译生成的可执行文件，[options]是执行gpmetis可选项，graphfile是将要划分的文件名，nparts是用户指定的划分的cluster的个数。用户通过配置options的-ptype参数可以指定gpmetis切分方式，-ptype = rb时，使用的是多层次递归二分切割算法，-ptype = kway时，使用的是多层次k路划分算法（默认值）。-ctype指定了coarsen操作时的策略，-ctype = rm时，随机匹配，-ctype = shem时使用SHEM（Sorted heavy-edge matching）方法进行匹配（默认值）。

METIS输入和输出文件的格式：输入文件格式如下图所示。



左图是无权重图，第一行是顶点个数和边的条数。除第一行之外，第 i 行表示 i-1 节点连接的顶点编号。比如第2行表示的是顶点1与5,3,2顶点之间有边直接相连。右图是有权图，第一行表示顶点个数和边的条数，以及format格式为带权重图。第 i 行表示 i-1 节点连接的顶点编号，紧跟边的权重值。

**2.  METIS编译**

因为在个人电脑上MAKE编译不成功，我使用了GitHub上的一个分支版本（见附件），然后用VS2019编译。只要遵循常规的编译流程就行。打包好的工程也在附件里，可以直接点开使用。

编译选项都已配好，记得选择DLL生成。

**3.  自制JNA接口**

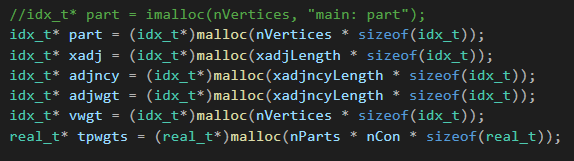
自制的 METIS JNA接口 注意内部数据类型的转换，使用了malloc创建可变长度的数组，然后cast成METIS需要的int64\_t。所有和JAVA接口的交换代码都在JnaInterface.cpp中，如果后续需要封装其他METS方法，继续在该文件内添加即可。相应的JNA版本包同样见附件。

需要注意的几点：

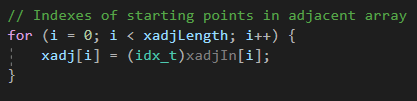
METIS自己构建了数组的数据格式idx\_t，本质上就是int32和in64，对应不同系统的版本，默认是64位。如果需要更改，就在metis.h中修改IDXTYPEWIDTH=32/64即可。

对于与JAVA交互接口函数，Char都使用指针传递，int和int[]数组都使用原生数组传递。

传递完毕后，在C语言中使用malloc构建一块内存空间，再用该地址的指针调用真正的METIS源码方法。例子如下：

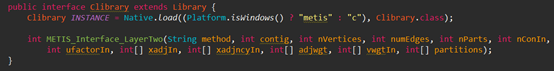


构建完内存空间后，手动给内存空间赋值，把从Java传递进来的原生数组赋值到内存空间。

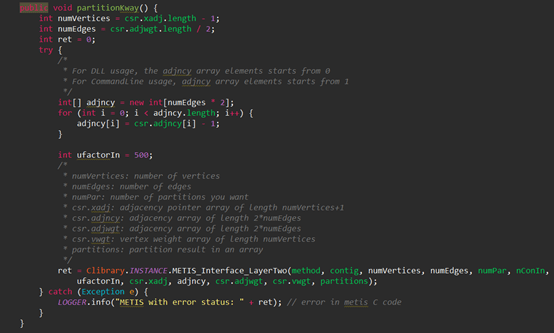


**4.  Java侧代码**

Java侧首先要导入JNA加包，并使用下图的Native方法加载我们自编译的METIS.DLL。

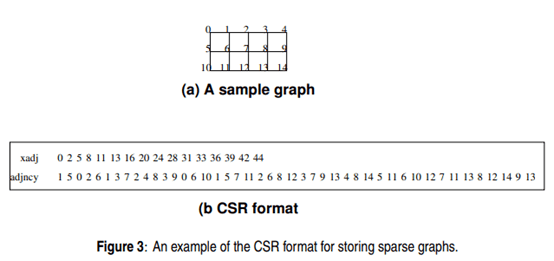


加载完后，我们就可以直接使用DLL中写好的接口函数了。



**5.  Compressed Storage Format (CSR)储存邻接矩阵**

METIS使用CSR矩阵传递图结构信息。这样做的好处是，使用四个简单的int数组，就可以表示整个图结构了。第一个xadj是指针数组，代码下图两个数组中，第几位到第几位属于哪一个点。第二个adjncy是邻接数组，记录了每个点的邻接点，并且通过xadj可以进行截断判断。第三个是adjwgt是边的权重。第四个vwgt是点的权重。



图结构的邻接矩阵通常大且稀疏，使用CSR格式储存可节省内存。

For each vertex i, its adjacency list is stored in consecutive locations in the array adjncy, and the array xadj is used to point to where it begins and where it ends. Figure 3(b) illustrates the CSR format for the 15-vertex graph shown in Figure 3(a).

adjncy储存了每一个vertex的邻居点信息，xadj则储存了每一个vertex在xadj中对应信息的起始和终止位置。

例如：vertex 0 邻接1和5，在xadj中，起始位置为0，终止位置为2-1=1。

vertex 1 邻接0，2，和6，在xadj中，起始位置为2，终止位置为5-1=4 。

**6.  附录**

全部接口函数源码如下：

#include<stdio.h>

#include<stdlib.h>//要使用malloc是要包含此头文件

#include <memory.h>

#include "metislib.h"

int METIS\_Interface(char \*method, int contig, int nVerticesIn, int numEdgesIn, int nPartsIn, int nConIn, int ufactorIn,

int xadjIn[], int xadjncyIn[], int adjwgtIn[], int vwgtIn[], double tpwgtsIn[], int partAns[])

{

    idx\_t nVertices = (idx\_t)nVerticesIn;

    idx\_t xadjLength = nVertices + 1;

    idx\_t xadjncyLength = (idx\_t)numEdgesIn \* 2;

    idx\_t objval;

    idx\_t nParts = (idx\_t)nPartsIn;

    idx\_t nCon = (idx\_t)nConIn; // nWeights

    idx\_t i;

    //idx\_t\* part = imalloc(nVertices, "main: part");

    idx\_t\* part = (idx\_t\*)malloc(nVertices \* sizeof(idx\_t));

    idx\_t\* xadj = (idx\_t\*)malloc(xadjLength \* sizeof(idx\_t));

    idx\_t\* adjncy = (idx\_t\*)malloc(xadjncyLength \* sizeof(idx\_t));

    idx\_t\* adjwgt = (idx\_t\*)malloc(xadjncyLength \* sizeof(idx\_t));

    idx\_t\* vwgt = (idx\_t\*)malloc(nVertices \* sizeof(idx\_t));

    real\_t\* tpwgts = (real\_t\*)malloc(nParts \* nCon \* sizeof(real\_t));

    //fill(adjncy, adjncy + xadjncyLength, 0);

    // Indexes of starting points in adjacent array

    for (i = 0; i < xadjLength; i++) {

        xadj[i] = (idx\_t)xadjIn[i];

    }

    // Adjacent vertices in consecutive index order

    for (i = 0; i < xadjncyLength; i++) {

        adjncy[i] = (idx\_t)xadjncyIn[i];

    }

    // Weights of vertices

    for (i = 0; i < nVertices; i++) {

        vwgt[i] = (idx\_t)vwgtIn[i];

    }

    for (i = 0; i < xadjncyLength; i++) {

        adjwgt[i] = (idx\_t)adjwgtIn[i];

    }

    // Target partition weight

    for (i = 0; i < nParts \* nCon; i++) {

        tpwgts[i] = (real\_t)tpwgtsIn[i];

    }

    // Initialise partition

    for (i = 0; i < nVertices; i++) {

        part[i] = 0;

    }

    idx\_t options[METIS\_NOPTIONS];

    METIS\_SetDefaultOptions(options);

    // options[METIS\_OPTION\_CCORDER] = 1; // Sepecifies if the connected components of the graph should be identified and ordered separately

    options[METIS\_OPTION\_UFACTOR] = (idx\_t)ufactorIn;

    options[METIS\_OPTION\_CONTIG] = (idx\_t)contig; // enforce contiuous partition

    int ret;

    switch (method[0])

    {

    case 'k':

        ret = METIS\_PartGraphKway(&nVertices, &nCon, xadj, adjncy,

            vwgt, NULL, adjwgt, &nParts, tpwgts, NULL, &options, &objval, part);

        break;

    case 'r':

        ret = METIS\_PartGraphRecursive(&nVertices, &nCon, xadj, adjncy,

            vwgt, NULL, adjwgt, &nParts, tpwgts, NULL, &options, &objval, part);

        break;

    default:

        ret = METIS\_PartGraphKway(&nVertices, &nCon, xadj, adjncy,

            vwgt, NULL, adjwgt, &nParts, tpwgts, NULL, &options, &objval, part);

    }

    //partition results to return, convert back to int[]

    for (i = 0; i < nVertices; i++) {

        partAns[i] = (int)part[i];

    }

    free(xadj);

    free(adjncy);

    free(adjwgt);

    free(part);

    free(vwgt);

    free(tpwgts);

    return ret;

}

int METIS\_Interface\_LayerTwo(char\* method, int contig, int nVerticesIn, int numEdgesIn, int nPartsIn, int nConIn, int ufactorIn,

    int xadjIn[], int\* xadjncyIn, int adjwgtIn[], int vwgtIn[], int partAns[]) {

    idx\_t nVertices = (idx\_t)nVerticesIn;

    idx\_t xadjLength = nVertices + 1;

    idx\_t xadjncyLength = (idx\_t)numEdgesIn \* 2;

    idx\_t objval;

    idx\_t nParts = (idx\_t)nPartsIn;

    idx\_t nCon = (idx\_t)nConIn; // nWeights

    idx\_t i;

    //idx\_t\* part = imalloc(nVertices, "main: part");

    idx\_t\* part = (idx\_t\*)malloc(nVertices \* sizeof(idx\_t));

    idx\_t\* xadj = (idx\_t\*)malloc(xadjLength \* sizeof(idx\_t));

    idx\_t\* adjncy = (idx\_t\*)malloc(xadjncyLength \* sizeof(idx\_t));

    idx\_t\* adjwgt = (idx\_t\*)malloc(xadjncyLength \* sizeof(idx\_t));

    idx\_t\* vwgt = (idx\_t\*)malloc(nVertices \* sizeof(idx\_t));

    //fill(adjncy, adjncy + xadjncyLength, 0);

    // Indexes of starting points in adjacent array

    for (i = 0; i < xadjLength; i++) {

        xadj[i] = (idx\_t)xadjIn[i];

    }

    // Adjacent vertices in consecutive index order

    for (i = 0; i < xadjncyLength; i++) {

        adjncy[i] = (idx\_t)xadjncyIn[i];

    }

    // Weights of vertices

    for (i = 0; i < nVertices; i++) {

        vwgt[i] = (idx\_t)vwgtIn[i];

    }

    for (i = 0; i < xadjncyLength; i++) {

        adjwgt[i] = (idx\_t)adjwgtIn[i];

    }

    // Initialise partition

    for (i = 0; i < nVertices; i++) {

        part[i] = 0;

    }

    idx\_t options[METIS\_NOPTIONS];

    METIS\_SetDefaultOptions(options);

    // options[METIS\_OPTION\_CCORDER] = 1; // Sepecifies if the connected components of the graph should be identified and ordered separately

    options[METIS\_OPTION\_UFACTOR] = (idx\_t)ufactorIn;

    options[METIS\_OPTION\_CONTIG] = (idx\_t)contig; // enforce contiuous partition

    int ret;

    switch (method[0])

    {

    case 'k':

        ret = METIS\_PartGraphKway(&nVertices, &nCon, xadj, adjncy,

            vwgt, NULL, adjwgt, &nParts, NULL, NULL, &options, &objval, part);

        break;

    case 'r':

        ret = METIS\_PartGraphRecursive(&nVertices, &nCon, xadj, adjncy,

            vwgt, NULL, adjwgt, &nParts, NULL, NULL, &options, &objval, part);

        break;

    default:

        ret = METIS\_PartGraphKway(&nVertices, &nCon, xadj, adjncy,

            vwgt, NULL, adjwgt, &nParts, NULL, NULL, &options, &objval, part);

    }

    //partition results to return, convert back to int[]

    for (i = 0; i < nVertices; i++) {

        partAns[i] = (int)part[i];

    }

    free(xadj);

    free(adjncy);

    free(adjwgt);

    free(part);

    free(vwgt);

    return ret;

}