大规模图数据中 kmax-truss 问题的求解和算法优化

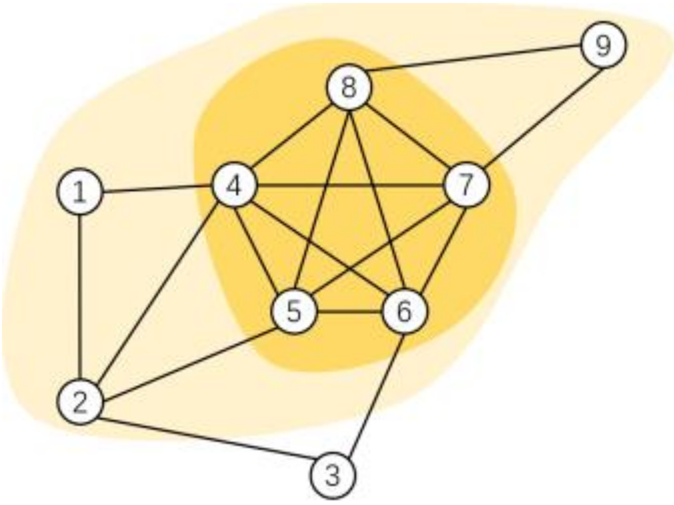
HEB-MATH

1. 基本原理

1.1 kmax-truss

kmax-truss（简称 k-truss）是一种子图结构，用于在图中寻找凝聚子图。计算 k-truss 要用到支持度(support)的概念，即图的一条边包含在 k 个三角形中，则这条边的支持度是 k。

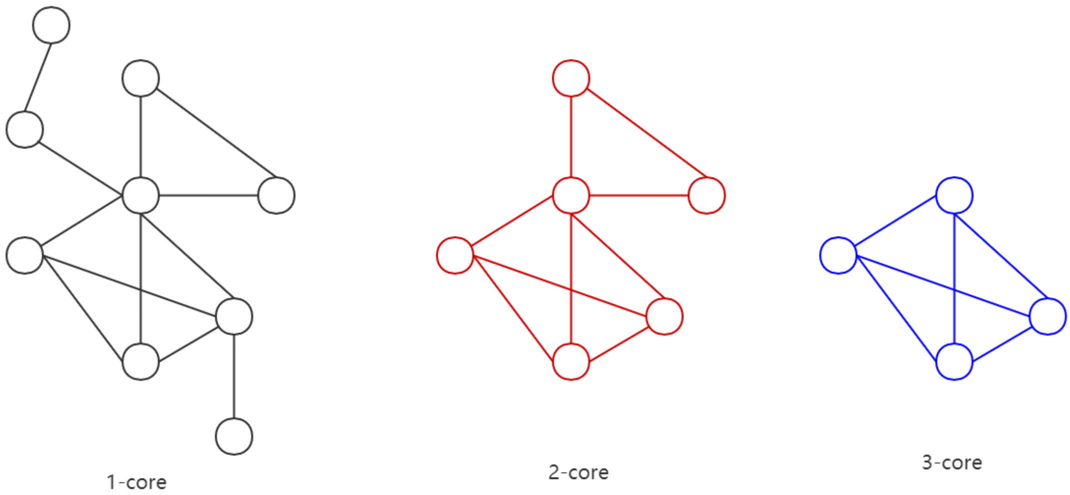
计算图 G 的 k-truss，即要找出该图的极大子图 g。图 g 中，每条边的支持度均要>=k-2。显然，对于同一图中，如果有 k1>k2，即有 k1-truss⊆k2-truss。



1.2 k-core

k-Core 算法是一种用来在图中找出符合指定核心度的紧密关联的子图结构，在 k-Core 的结果子图中，每个顶点至少具有 k 的度数，且所有顶点都至少与该子图中的 k 个其他节点相连。对于同一图中，如果有 k1>k2，同样有 k1-core⊆k2-core。

此外，在一个 k-truss 图中，每个顶点必有度 k-1，那么 k-truss 图必然也是一个(k-1)-core 图，即 k-truss⊆(k-1)-core。例如，那么 5-truss 图必然也是一个 4-core 图。



1.3 truss decomposition

Truss decomposition 是计算图数据 k-truss 的常用算法。

首先计算图中每条边所属的三角形数量。然后从 k=3 开始，通过将三角形数量小于(k-2)的边剥离，从而得到图的 k-truss。然后逐渐增大 k 值，直到到找存在的最大的 k 值，即 kmax-truss。

1.4 无向图的三角形数计算

Truss decomposition 方法的基础是三角形计算。

将待分析的关联数据处理成无向图，并用邻接矩阵 **A** 进行表示。图中的每一条边就可以用矩阵的元素位置来表示，即边 e(u, v)就可以表示成元素$a_{u,v}=1$。

为了更直观地表示邻接矩阵与图数据的对应关系，如图 3 所示，把图 3 中左边的图转化为右边的上三角矩阵表示，可见邻接矩阵的非零元素即对应图的一条边。共享边(a, b)的三角形个数，即边(a, b)支持度，即为邻接矩阵中行 a 和行 b 的元素列位置重合的数目。

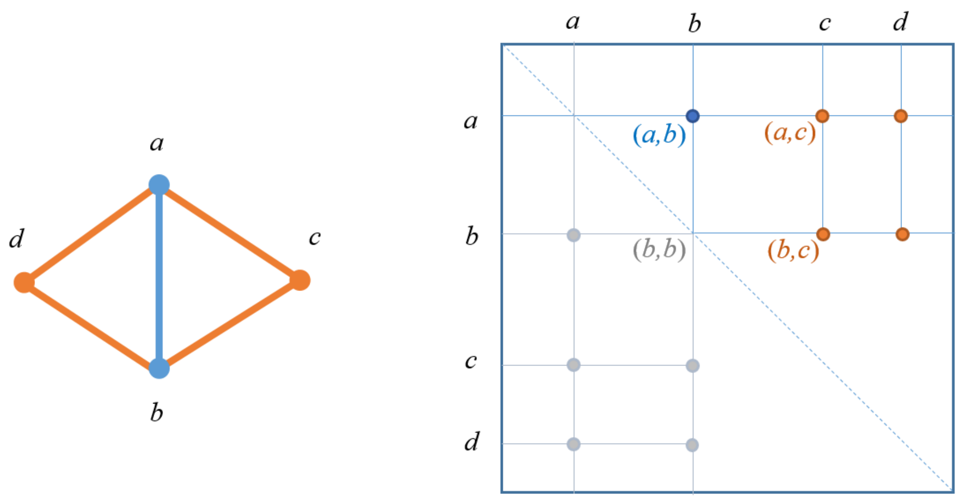


图 3 图表示与对应其邻接矩阵

2. 基本算法过程

基本算法实现，步骤如下：

Step 1. 将关联数据读入内存，并转化为二进制格式；

Step 2. 对数据进行排序、去重、去自环边，形成邻接矩阵 **A**；

Step 3. 利用三角形计算方法计算每条边的支持度；

Step 4. 给定一个较小的初始 k 值，循环去除支持度<k 的边，直到所有边的支持度都>=k；

Step 5. 适当增大 k 值，例如 k=k+1，重复步骤 4，直到找到 kmax。

3. 方案优化

相比于 k-truss 求解，k-core 的算法的复杂度要低很多。又因为 k-truss 是 k-core 的子图，所以我们可以 k-core 算法来增强 k-truss 的解法。

对于某个 k 值，可以先用 k-core 算法对原图数据规模进行精减。然后在 k-core 图上进行 k-truss 计算，这样可以大幅降低复杂度。

由于 k-truss 是 k-core 的子图，那么对于同一个图来说，k-core 得到的 kmax 值一定不小于 k-truss 得到的 kmax 值。则可以用 k-core 算法来估计 kmax 的上限。然后用步长控制方案来反向搜索 kmax 值，这次既可以在有限步数内得去结果，又可以把计算规模始终控制在较低水平。

4. 大规模计算面临的问题及对策

由于我们要面对的大规模图数据，那么可以数据读入、数据转化、三角形计

数算法等方面对算法进行优化。

4.1 变量内存空间最大化利用

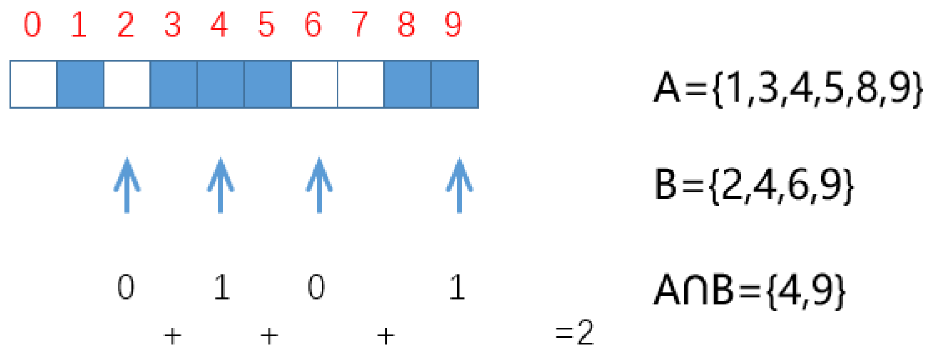
大文件必定占用大内存，我们之前将文件映射到内存，再拷贝到另一个数组中再进行后续操作，但变量的声明、初始化，数据交换都需要大量的计算开销。其实我们完全可以只在已经分配的内存上进行操作，实现数据交换、排序等操作，这样即可以节省大量的内存空间开销，还可以提高计算效率。

4.2 数据格式转化分块操作

本次赛题提供的是文本型数字，由于数据量大，利用数据流读取或顺序读取并转化的方法效率较低。那么我们可以按计算机的线程数来将数据进行等量分块，然后并行读取并转化。

4.3 三角形计数算法

我们在往届的三角形计数比赛中设计了高效的计数算法。例如我们求两个集合的交集，不是对集合元素进行比较的方式，而是在一个零值空间中将一个集合进行标记，另一个集合从相应位置取值进行求和，就得到了交集元素个数。



5. 详细算法设计与实现

STEP 1. 利用 mmap()将数据映射到内存

STEP 2. 将文本格式数据转化为二进制数据

step 2.1 浏览数据，确定数据规模

step 2.2 按线程数对数据进行分片，并将每段起始点与数据单元对齐

step 2.3 多线程并行转化数据为二进制格式

STEP 3. 对数据进行排序去重，并简化为邻接矩阵稀疏格式

STEP 4. 进行 k-core 求解，找到最大的 k 值

step 4.1 对顶点度进行统计，循环去除小于当前设定 k 值的边，直到所有顶点度不小于 k 值

step 4.2 将步长加倍，如果子图不为空，不断提高 k 值下限值 k_floor，直到子图规模为 0，将其标定为 k 的上限值 k_ceil

step 4.3 步长控制改为中位控制 k=(k_floor+k_ceil)/2，不断收敛到到 k_ceil-k_floor=1，k_floor 即为 k-core 的最大值

STEP 5. 以 k-core 最大值为基准，进行中位搜索，改进 truss decomposition 方法得到最终的值

step 5.1 按用 k-core 方法对原数据进行精减，同时对顶点序号按新的规模进行替换，然后调用三角形计数算法

step 5.2 用 truss decomposition 方法对图进一步修剪，直到所有边的支持度大于当前 k 值

step 5.3 搜索策略，将 k-core 的最大值设为当前的 k_ceil，将 k_floor 设置为较小数值（比如 4），不断用 k=(k_floor+k_ceil)/2 值重复步骤 5.1-5.2 的计算，如果修剪后子图不为空更新 k_floor，反之更新 k_ceil，直到 k_ceil-k_floor=1，则 kmax=k_floor+2

6. 程序代码编译说明

编译过程由 makefile 进行批处理，请在源文件目录下，运行命令 make 完成程序编译，将生成可执行文件 kmtruss。

7. 程序代码运行使用说明

运行程序请用命令格式：

./kmtruss -f [数据所在目录/图数据文件]

例如： ./kmtruss -f ../ktruss-data/soc-LiveJournal.tsv

8. 运行环境及测评分析

8.1 运行测试环境参数

CPU：Intel Xeon E5-2686 v4 (8 逻辑核心)

内存：60G

硬盘：500G

操作系统： Ubuntu 16.04.06 LTS（容器）

8.2 比赛数据集测试数据

表 1 比赛数据集测试数据

序号	数据集	顶点数	边数	kmax	Edges in kmax-truss	计时 (s)
1	s18.e16.rmat.edgelist	262145	7604036	164	225,529	4
2	s19.e16.rmat.edgelist	524389	15459034	223	334,934	8
3	cit-Patents	3774769	33037894	36	2,625	3
4	soc-LiveJournal	4847572	85702474	362	72,913	10