# 大规模图数据中kmax-truss问题的求解和算法优化

## 摘要

**k-truss是一种聚合的子图，它比clique的限制更严格，在寻找图中稠密子图和社团发现等领域ktruss有重要作用。目前关于k-truss的研究很多，k-truss分解的效率优化的很高，但是针对于kmax-truss分解研究较少。本文提出了一种基于k-core分解的kmax-truss求解优化算法（KK算法），通过k-core分解和k-truss分解之间的关系确定图G的上下界，并根据点的core值进行切图，切出一块kmax-truss和G等同的子图G’，但G’的规模远远小于G，能够更方便的将图读取进内存或显存中进行计算。同时，我们从该下界开始对G’进行k-truss分解。因此，我们提出的KK算法可以快速确定图G的kmax-truss。**

## 简介

在大图中找到具有凝聚力的结构是一种重要的数据挖掘技术，可用于各种领域的信息检索，例如基因组学，社交媒体，网络安全，计算机网络，公共卫生等等。近年来大数据和网络迅速发展，用图来存储数据的情况越来越多，这些图数据集呈指数增长。因此，迫切需要开发创新的算法，软件和硬件，以在合理的时间内有效地处理这些大数据集。随着图数据规模的不断扩大，对大型图进行分析处理的成本越来越高，因此研究人员经常通过分析大型图中的稠密子图来获得大型图的主要特征。为了衡量社交网络中子图的稠密程度，许多子图模型被提出，例如clique,k-core,k-truss等。

k-truss分解是一种图形分析技术，也是本文的重点。k-truss是一个内聚子图，其中每个边都是至少k-2个三角形的一部分。该子图放宽了团(clique)的概念，可以在多项式时间内进行计算。对于给定的k值，k-truss分解算法[3]，[4]，[5]的最新实现包括三个主要步骤：

1）确定图中每个边的trussness值，即每个边被多少个三角形包含；

2）对于给定的k值，删除其trussness值小于k-2的边缘，通常称为“剥离”；

3）移除受影响的边缘，直到图中的边缘均不小于k−2。如果该图不为空，则k的值将增加1，并重复上述三个步骤，直到达到图形为空的最大k-truss值。

但是，如果仅要确定具有最大k-truss分解的边，即kmax-truss，则在每次迭代之后k值增加1的增量方法会导致额外的计算。在这种情况下，需要更快的方法。

为了解决以上问题，本文主要着眼于减少需要进行k-truss分解的图形规模来实现快速确定kmax-truss。与[4]不同，本文通过k-core分解和k-truss分解之间的关系和根据点的core值进行图的裁剪切分，具体步骤如下：

1. 对需要处理的大规模无向无权原图G，首先进行k-core分解，并从原图G中裁剪出点的core值为kmax-core的子图G1，其中，kmax-core是kmax-truss的上界。
2. 对子图G1计算支持边信息进行truss分解（子图G1的规模很小，所以此次分解过程花费的时间远远小于原图G的k-truss分解时间），得到子图G1的kmax-truss(k1),即为原图G中kmax-turss的下界。
3. 根据k1切下原图G中的(k1-1)-core子图G2，并对子图G2进行k-truss分解（从k1下界开始），得到的kmax-truss即为原图G的kmax-truss。

具体的原理证明和过程在下面介绍。通过以上过程，首先子图G2的规模远远小于原图G（后面有简单具体例子说明）；其次子图G2的k-turss分解无需从1开始，而是从k1开始，提高了分解效率。

## 基础知识

k-core [1] 是⼀种基于⼦图中点的度数来对⼦图进⾏稠密程度评估的标准，在⼀个k-core图中，所有点必须满⾜⾄少与k个点相邻。图1中最外⾯到最⾥⾯的圈起来的⼦图分别为0-core，1-core，2-core和3-core。

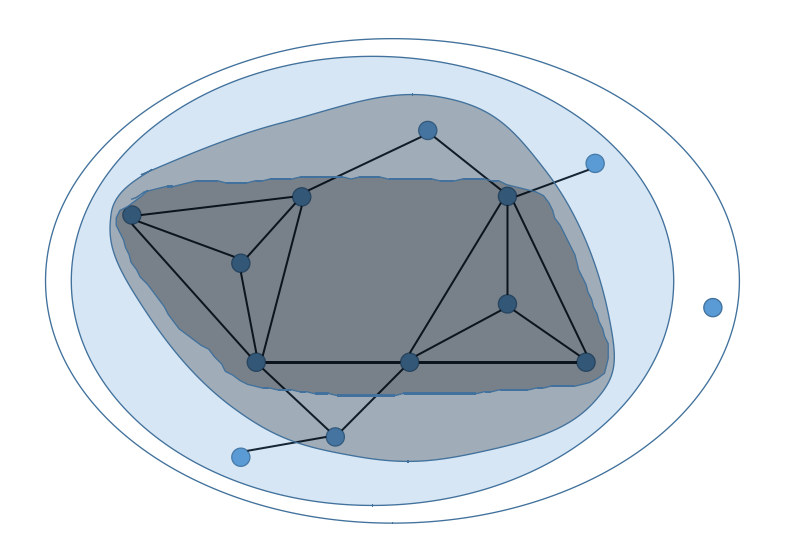


图1 core的示例

k-truss [2] 是由Jonathan Cohen于2008年提出的另一种稠密子图结构，它由团的概念衍生而来。k-truss放宽了结构约束，仅要求图中每条边至少属于（k-2）个三角形。k-truss是一种分层递进的子图结构，3-truss~kmax-truss刻画了不同稠密程度的网络核心，它们彼此间是包含关系，即一个4-truss可以是一个3-truss的子图，一个5-truss可以是一个4-truss的子图。图2展示了truss间的包含关系，其中浅色部分是图的3-truss部分（每条边至少属于一个三角形），而深色部分是图的4-truss部分（每条边至少属于两个三角形），同时也是图的5-truss（每条边至少属于三个三角形）。k-truss在现实生活中有着丰富的应用，例如在社交网络中，k-truss经常用于发掘关系紧密的团体。

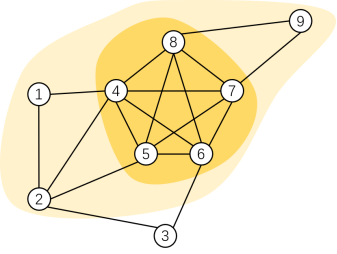


图2 truss间的包含关系

## 研究现状

对于k-truss分解已有了很多研究[3-12]，在此具体介绍[3]，从三大方面对k-truss分解进行效率提升，也是本文k-truss分解的主要参考代码：

1. 支持边初始化（三角形计数原图所有边的支持度）：设计了一种word-packing技术，以改进现有的基于位图的三角计数算法；
2. 优化剥离边的迭代过程包括三个方面：1）动态选择重新计算所有剩余边的支持度，或仅减少受影响所切边的支持度。2）具有数据偏斜处理和图形压缩的三角枚举3）基于索引的边过滤（每次剥离过程仅检索支持度在该范围内的所有边）；
3. 在多核CPU和GPU上并行算法，并设计分流策略以限制CPU和GPU之间的数据传输。

但是，对于只需要获取kmax-truss的研究比较少。如果采用k-truss分解来获取kmax-truss将产生大量的无效计算消耗。[4]is a winner of the Student Innovation Awards in the 2019 GraphChallenge2019，对kmax-truss求解做了一些优化，但存在大量问题，例如上界选取过于简单粗暴，需要迭代几次后才可以逼近实际kmax-truss；没有进行下界的选取，k-truss分解仍然是从1开始；根据点的度而不是k-core进行切图，所得到的子图规模会比根据k-core切图得到的子图规模大很多，因为k-core比点的度定义更严格。

## 基于k-core分解的kmax-truss求解优化算法

为了实现快速获取kmax-truss值，本文提出了基于k-core分解的kmax-truss求解优化算法（KK算法）。通过k-core分解和k-truss分解之间的关系，KK算法的具体过程如图3所示，包含以下步骤：

1. 获取kmax的上界并获取kmax-core的子图G1。
2. 对图G1进行k-truss分解获取kmax的下界并获取(k1-1)-core的子图G2.
3. 对图G2进行k-truss分解得到原图G的k-max。



图3 KK算法的流程图

1. **确定上界**

本文是通过k-core分解和k-truss分解之间的关系来确定k-max的上下界的，因此首先介绍本文提出的定义1

Definition 1: k-core子图中不一定存在（k+1）-truss子图，但存在（k2+1）-truss子图，其中k2<=k。

由定义可以知道，k-truss子图一定是(k-1)-core子图，反之则不正确，即图G的(k-1)-core子图包含图G的k-truss子图，或者说图G的k-truss子图比图G的(k-1)-core子图限制更严格。但存在一种特殊情况，如图4所示，在完全互连的情况下，这两者是等价的，即图G的(k-1)-core子图是图G的k-truss子图，所以k-core子图中不一定存在（k+1）-truss子图，但存在（k2+1）-truss子图，其中k2<=k。到此我们证明定义1是正确的。接下来确定图G的kmax上界。

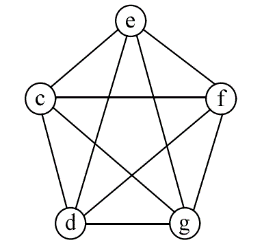


图4 完全互连图

首先读取文件，获得无权无向图G，即需要求取kmax-truss的图。通过对图G进行k-core分解，我们可以得到每个点的邻度，即相邻的点的数量，进一步我们可以得到图G中点的最大邻度,为kmax-core。根据定义1，我们可以得到图G的kmax-truss<=kmax-core+1，并裁剪图G的kmax-core子图，记为G1。

1. **确定下界**

Definiton 2:

很明显，属于S的点和边都属于G，所以定义2是正确的。

对图G1进行k-truss分解，将G1的最大kmax-truss记为k1。由定义2我们可以得到G的kmax-truss>= k1。

1. **确定kmax-truss**

Definiton 3: 图G的k-truss子图包含于（k-1）-core子图。

k-truss子图中一定是（k-1）-core子图，反之则不对，这是大家公认的定理，在此简要说明一下。

k-truss子图中每条边e至少被k-2个三角形包含，即每条边e至少和k-2点相连，再加上每条边e中的两点互连，所以每条边中的点至少和k-1个点互连，即为（k-1）-core子图。

但（k-1）-core子图不一定是k-truss子图。如图5所示，该图为3-core图，但图中的红边没有被任何三角形包含，即该图不是2-truss图，所以（k-1）-core子图不一定是k-truss子图



图5 3-core图

综上，我们可以得到图G的k-truss子图包含于（k-1）-core子图，即定义3是正确的。

我们已经获取图G的kmax下界k1，则图G的k1-truss子图包含于（k1-1）-core子图，即图G的kmax-truss子图一定包含于（k1-1）-core子图，我们截取图G的（k1-1）-core子图，得到图G2。然后对G2从k1开始进行k-truss分解，得到的kmax-truss即为原图G的kmax-truss。

1. **KK算法的简单示例**

如图6是原图G，图中数字1-7代表点的度，即与多少边相邻。



图6 原图G

我们对图G进行k-core分解，分别得到1-core图，2-core图、3-core图和4-core图分别如图7、图8、图9和图10所示。



图7 原图G的1-core图



图8 原图G的2-core图



图9 原图G的3-core图



图10 原图G的4-core图

我们得到图G的kmax-core=4, 则图G的kmax的上界为5，并截取4-core子图得到G1，即图10，然后对图10进行k-truss分解，得到G1的最大kmax-truss为5，如图11所示，即下界为5，然后截取4-core的子图作为图G2，对G2从5进行k-truss分解即可得到G2的kmax=5，即为原图G的kmax。



图11 子图G1的5-truss图

通过对比图G(图6)、G1（图10）和G2（图11），可以看到本文中进行k-truss分解操作的G1和G2规模小于图G。

事实上现实中网络中的图比我们所举例子更加稀疏，采用本方法缩减的规模更明显，例如在s23.e15这个图中，原图是242387826条边，取kcore上界子图G1是2681340条边，取下界子图G2是18434084条边。

## 参考文献

[1] Seidman S B . Network structure and minimum degree[J]. Social Networks, 1983, 5:269.

[2] Cohen J. Trusses: Cohesive subgraphs for social network analysis[J]. National Security Agency Technical Report, 2008, 16 : 3-1.

[3] Che Y , Lai Z , Sun S , et al. Accelerating truss decomposition on heterogeneous processors[J]. Proceedings of the VLDB Endowment, 2020.

[4] Almasri M , Anjum O , Pearson C , et al. Update on k-truss Decomposition on GPU[C]// 2019 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2019.

[5] Conte A , Sensi D D , Grossi R , et al. Truly Scalable K-Truss and Max-Truss Algorithms for Community Detection in Graphs[J]. IEEE Access, 2020, PP(99):1-1.

[6] Date K , Feng K , Nagi R , et al. Collaborative (CPU + GPU) algorithms for triangle counting and truss decomposition on the Minsky architecture: Static graph challenge: Subgraph isomorphism[C]// 2017 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2017.

[7] Pearce R , Sanders G . K-truss decomposition for Scale-Free Graphs at Scale in Distributed Memory[C]// 2018 IEEE High Performance extreme Computing Conference (HPEC). IEEE, 2018.

[8] Bisson M , Fatica M . Update on Static Graph Challenge on GPU[C]// 2018 IEEE High Performance extreme Computing Conference (HPEC). IEEE, 2018.

[9] Low T M , Spampinato D G , Kutuluru A , et al. Linear Algebraic Formulation of Edge-centric K-truss Algorithms with Adjacency Matrices[C]// 2018 IEEE High Performance extreme Computing Conference (HPEC). IEEE, 2018.

[10] Conte A , Sensi D D , Grossi R , et al. Discovering $k$-Trusses in Large-Scale Networks[C]// 2018 IEEE High Performance extreme Computing Conference (HPEC). IEEE, 2018.

[11] Voegele C , Lu Y S , Pai S , et al. Parallel triangle counting and k-truss identification using graph-centric methods[C]// 2017 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2017.

[12] Smith S , Liu X , Ahmed N K , et al. Truss decomposition on shared-memory parallel systems[C]// 2017 IEEE High Performance Extreme Computing Conference (HPEC). IEEE, 2017.