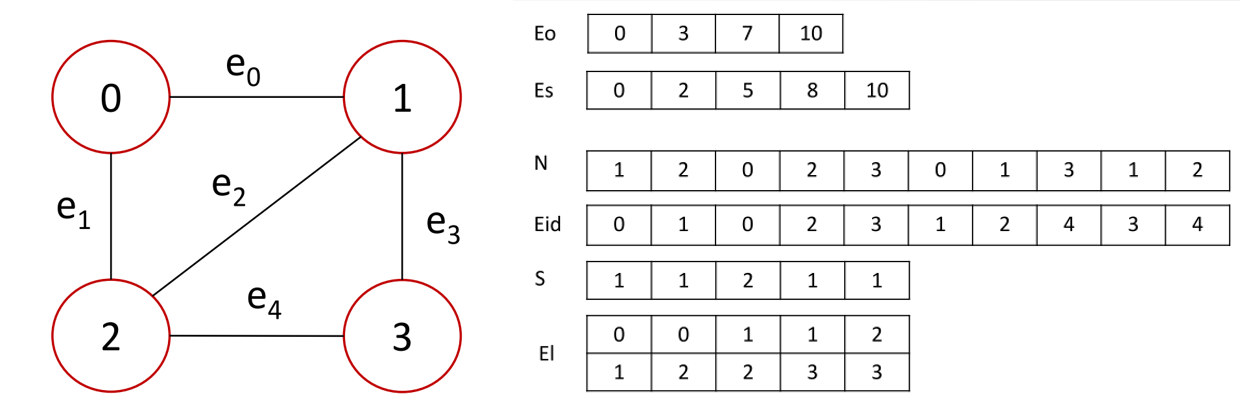
# kmax-truss技术报告文档

1. 算法介绍

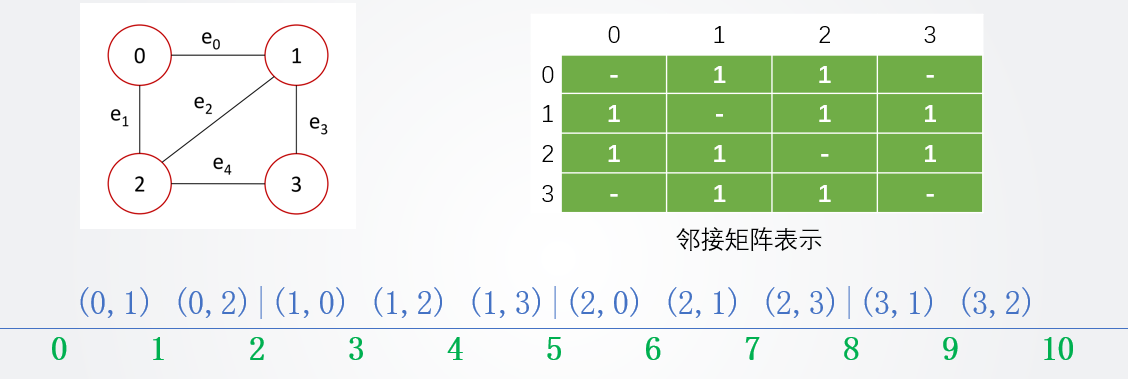
本算法计算kmax-truss主要经过三个过程。

### 1.1 CSR编码方式存储图

使用CSR编码存储图，这样使得图存储的内存消耗只与边长度成正相关，便于并行计算。结构如图1、2所示。



**图1 CSR结构图**



**图2 CSR计算图**

该方法主要是按邻接矩阵的方式将节点以行顺序连接起来，且下标对应如图2所示，10个节点分布穿插在0-10的下标之中。

**Eo**：存放比自己下标大的邻居节点，如节点1下标比自己大的节点为2，即（1，2）所对应下标值为3；

**(N, Es)：**存放4个节点的中的边分布，Es的数量为节点数+1，以这5个点划分区间，如Es[1]-Es[0]=2,表示节点0所对应的边节点有2个，N为边的数量\*2,具体下标为N[0]、N[1]，可推断为(0,1)(0,2)俩个节点；

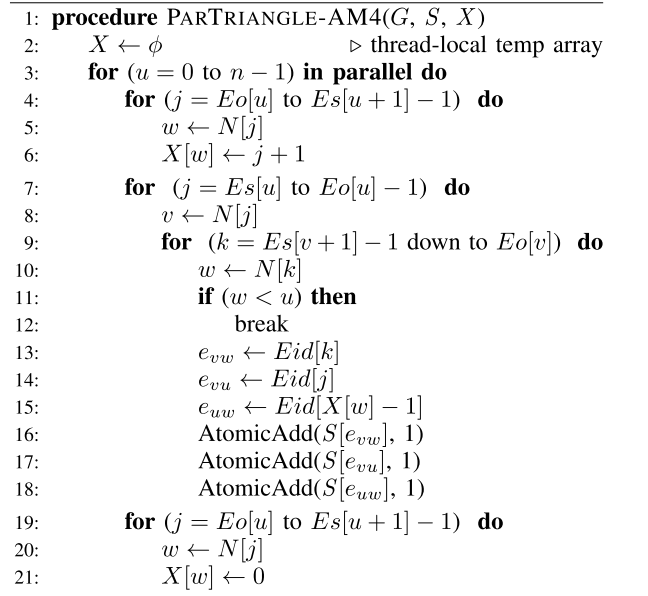
El：代表边的顶点，如边ID为0的边对应顶点为0、1；

Eid：代表节点所处的边ID，如(0,1)所对应的边ID为0；

S：代表边所处的三角形数量，即边支持度。

### 1.2 计算边的支持度

该算法使用基于边缘的方法并行计算三角形。 每个线程初始化大小为节点数量n的临时数组X。 为了计算对边u，v的支持度。首先在数组X中标记u的邻居。然后线程访问v的邻居w，并检查是否标记了X [w]。 如果标记了该条目，则u，v，w形成三角形uvw，因此u，v的支持度增加1，防止计算重复，按边从小到大计算。如图3所示。



**图3 AM4三角形计算**

### 1.3 计算k-truss

本方法采用自下而上的k-truss方法，从支持度为0向上计算直到所有节点都计算完毕。介绍用到的数据结构：

S：代表边支持度

Curr：向当前Level层级添加边；

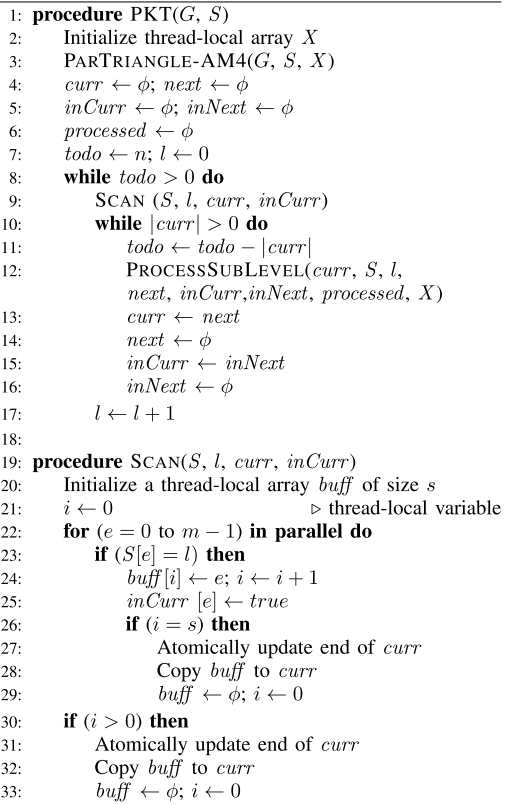
Next：向下一级Level添加边；

InCurr、InNext：避免重复计算；

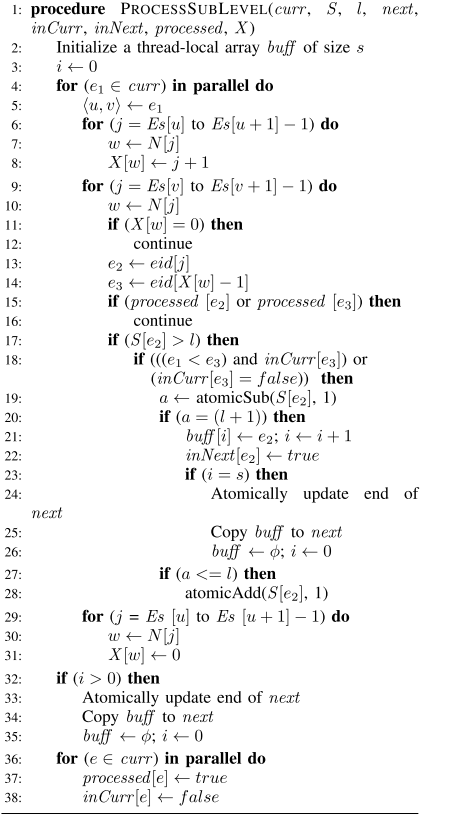
Todo:代表所需要处理的节点数量；

Processed:代表边被处理过。

一开始扫描边为level的节点放入curr，在并行化的过程防止大量原子操作造成时间的浪费，采用缓冲机制先将部分节点放入缓冲数组中，待一定量之后再放回。之后我们对curr进行计算，计算level。如图4、5所示



**图4 ktruss计算方法**



**图5 分解三角形计算方法**

2. 详细算法设计与实现

### 2.1 CSR编码方式存储图

1. // 读图
2. **void** load\_graph(Graph\* g, **char**\* path) {
3. **FILE**\* infp = fopen(path, "r");
4. UI u, v, t, \_max = 0;
5. **long** m = 0;
6. **while** (fscanf(infp, "%u %u %u\n", &u, &v, &t) != EOF) {
7. //只获取一半
8. **if** (u > v) {
9. \_max = figuremax(\_max, u);
10. vts.push\_back(u);
11. vts.push\_back(v);
12. }
13. }
14. g->nodeNums = \_max + 1;
15. //初始化边
16. g->num\_edges = (UI\*)malloc((g->nodeNums + 1) \* **sizeof**(UI));
17. #pragma omp parallel for
18. **for** (**long** i = 0; i < g->nodeNums + 1; i++) {
19. g->num\_edges[i] = 0;
20. }
21. //读边
22. **for** (UI i = 0; i < vts.size(); i += 2)
23. {
24. g->num\_edges[vts[i]]++;
25. g->num\_edges[vts[i + 1]]++;
26. }
27. //需要n+1个点 跟边一样
28. UI\* temp\_num\_edges = (UI\*)malloc((g->nodeNums + 1) \* **sizeof**(UI));
29. temp\_num\_edges[0] = 0;
30. **for** (**long** i = 0; i < g->nodeNums; i++) {
31. m += g->num\_edges[i];
32. temp\_num\_edges[i + 1] = m;
33. }
34. //g下的m是edges的俩倍
35. g->m = m;
36. g->adj = (UI\*)malloc(m \* **sizeof**(UI));
37. #pragma omp parallel
38. {
39. #pragma omp for schedule(static)
40. **for** (**long** i = 0; i < g->nodeNums + 1; i++)
41. g->num\_edges[i] = temp\_num\_edges[i];
42. #pragma omp for schedule(static)
43. **for** (**long** i = 0; i < m; i++)
44. g->adj[i] = 0;
45. }
46. **for** (UI i = 0; i < vts.size(); i += 2)
47. {
48. g->adj[temp\_num\_edges[vts[i]]] = vts[i + 1];
49. temp\_num\_edges[vts[i]]++;
50. g->adj[temp\_num\_edges[vts[i + 1]]] = vts[i];
51. temp\_num\_edges[vts[i + 1]]++;
52. }
53. vts.clear();
54. free(temp\_num\_edges);
55. }
56. //获取边ID和边列表Populate eid and edge list
57. **void** getEidAndEdgeList(Graph\* g, Edge\* els) {
58. g->eid = (UI\*)malloc(g->m \* **sizeof**(UI));
59. assert(g->eid != NULL);
60. UI\* num\_edges\_copy = (UI\*)malloc(g->nodeNums \* **sizeof**(UI));
61. **for** (UI i = 0; i < g->nodeNums; i++) {
62. num\_edges\_copy[i] = g->num\_edges[i];
63. }
64. **long** edgeId = 0;
65. //TODO 无向图计算应该只需要一半对称 计算所有边并给出ID
66. **for** (UI u = 0; u < g->nodeNums; u++) {
67. **for** (UI j = g->num\_edges[u]; j < g->num\_edges[u + 1]; j++) {
68. UI v = g->adj[j];
69. **if** (u < v) {
70. Edge e;
71. e.u = u;
72. e.v = v;
73. g->eid[j] = edgeId;
74. num\_edges\_copy[u]++;
75. **if** (g->adj[num\_edges\_copy[v]] == u) {
76. g->eid[num\_edges\_copy[v]] = edgeId;
77. num\_edges\_copy[v]++;
78. }
79. els[edgeId] = e;
80. edgeId++;
81. }
82. }
83. }
84. }

### 2.2 计算边支持度

1. #pragma omp for schedule(dynamic,10)
2. **for** (UI u = 0; u < nodeNums; u++) {
3. **for** (UI j = startEdge[u]; j < g->num\_edges[u + 1]; j++) {
4. UI w = g->adj[j];
5. X[w] = j + 1;
6. }
7. **for** (UI j = g->num\_edges[u]; j < startEdge[u]; j++) {
8. UI v = g->adj[j];
9. **for** (UI k = g->num\_edges[v + 1] - 1; k >= startEdge[v]; k--) {
10. UI w = g->adj[k];
11. // 计算大的
12. **if** (w <= u) {
13. **break**;
14. }
15. **if** (X[w]) {
16. UI e1 = g->eid[X[w] - 1], e2 = g->eid[j], e3 = g->eid[k];
17. \_\_sync\_fetch\_and\_add(&edgeSupport[e1], 1);
18. \_\_sync\_fetch\_and\_add(&edgeSupport[e2], 1);
19. \_\_sync\_fetch\_and\_add(&edgeSupport[e3], 1);
20. }
21. }
22. }
23. // 归0
24. **for** (UI j = startEdge[u]; j < g->num\_edges[u + 1]; j++) {
25. UI w = g->adj[j];
26. X[w] = 0;
27. }
28. }

### 2.3 计算ktruss

1. //计算truss
2. **int** level = 0;
3. **long** todo = edgeNums;
4. **while** (todo > 0) {
5. scanLevel(edgeNums, edgeSupport, level, curr, &currTail, InCurr);
6. **while** (currTail > 0) {
7. todo = todo - currTail;
8. figureSubLevel\_intersection(g, curr, InCurr, currTail, edgeSupport, level, next, InNext, &nextTail, processed, els);
9. **if** (tid == 0) {
10. UI\* tempCurr = curr;
11. curr = next;
12. next = tempCurr;
13. **bool**\* tempInCurr = InCurr;
14. InCurr = InNext;
15. InNext = tempInCurr;
16. currTail = nextTail;
17. nextTail = 0;
18. }
19. #pragma omp barrier
20. }
21. level = level + 1;
22. #pragma omp barrier
23. }
24. // 扫描度为level的边
25. **void** scanLevel(**long** edgeNums, **int**\* edgeSupport, **int** level, UI\* curr, **long**\* currTail, **bool**\* inCurr) {
26. **const** **long** BUFFER\_SIZE\_BYTES = 2048;
27. **const** **long** BUFFER\_SIZE = BUFFER\_SIZE\_BYTES / **sizeof**(UI);
28. UI buff[BUFFER\_SIZE];
29. **long** index = 0;
30. #pragma omp for schedule(static)
31. **for** (**long** i = 0; i < edgeNums; i++) {
32. **if** (edgeSupport[i] == level) {
33. // 计算level的边
34. buff[index] = i;
35. inCurr[i] = **true**;
36. index++;
37. **if** (index >= BUFFER\_SIZE) {
38. // 使用原子操作 采用缓冲区 避免占用过大开销
39. **long** tempIdx = \_\_sync\_fetch\_and\_add(currTail, BUFFER\_SIZE);
40. **for** (**long** j = 0; j < BUFFER\_SIZE; j++) {
41. curr[tempIdx + j] = buff[j];
42. }
43. index = 0;
44. }
45. }
46. }
47. **if** (index > 0) {
48. **long** tempIdx = \_\_sync\_fetch\_and\_add(currTail, index);
49. **for** (**long** j = 0; j < index; j++) {
50. curr[tempIdx + j] = buff[j];
51. }
52. }
53. #pragma omp barrier
54. }
56. **void** figureSubLevel\_intersection(Graph\* g, UI\* curr, **bool**\* InCurr, **long** currTail, **int**\* edgeSupport,
57. **int** level, UI\* next, **bool**\* InNext, **long**\* nextTail, **bool**\* processed, Edge\* els) {
59. //缓存数量
60. **const** **long** BUFFER\_SIZE\_BYTES = 2048;
61. **const** **long** BUFFER\_SIZE = BUFFER\_SIZE\_BYTES / **sizeof**(UI);
62. UI buff[BUFFER\_SIZE];
63. **long** index = 0;
64. #pragma omp for schedule(dynamic,4)
65. **for** (**long** i = 0; i < currTail; i++) {
66. //处理边
67. UI e1 = curr[i];
68. Edge edge = els[e1];
69. UI u = edge.u;
70. UI v = edge.v;
71. UI uStart = g->num\_edges[u], uEnd = g->num\_edges[u + 1];
72. UI vStart = g->num\_edges[v], vEnd = g->num\_edges[v + 1];
73. UI numElements = (uEnd - uStart) + (vEnd - vStart);
74. UI j\_index = uStart, k\_index = vStart;
75. **for** (UI innerIdx = 0; innerIdx < numElements; innerIdx++) {
76. **if** (j\_index >= uEnd || k\_index >= vEnd) {
77. **break**;
78. }
79. **else** **if** (g->adj[j\_index] == g->adj[k\_index]) {
80. UI e2 = g->eid[k\_index];  //<v,w>
81. UI e3 = g->eid[j\_index];  //<u,w>
82. **if** ((!processed[e2]) && (!processed[e3])) {
83. **if** (edgeSupport[e2] > level && edgeSupport[e3] > level) {
84. **int** supE2 = \_\_sync\_fetch\_and\_sub(&edgeSupport[e2], 1);
85. **if** (supE2 == (level + 1)) {
86. buff[index] = e2;
87. InNext[e2] = **true**;
88. index++;
89. }
90. **if** (supE2 <= level) {
91. \_\_sync\_fetch\_and\_add(&edgeSupport[e2], 1);
92. }
93. **if** (index >= BUFFER\_SIZE) {
94. **long** tempIdx = \_\_sync\_fetch\_and\_add(nextTail, BUFFER\_SIZE);
95. **for** (**long** bufIdx = 0; bufIdx < BUFFER\_SIZE; bufIdx++)
96. next[tempIdx + bufIdx] = buff[bufIdx];
97. index = 0;
98. }
99. **int** supE3 = \_\_sync\_fetch\_and\_sub(&edgeSupport[e3], 1);
100. **if** (supE3 == (level + 1)) {
101. buff[index] = e3;
102. InNext[e3] = **true**;
103. index++;
104. }
105. **if** (supE3 <= level) {
106. \_\_sync\_fetch\_and\_add(&edgeSupport[e3], 1);
107. }
108. **if** (index >= BUFFER\_SIZE) {
109. **long** tempIdx = \_\_sync\_fetch\_and\_add(nextTail, BUFFER\_SIZE);
110. **for** (**long** bufIdx = 0; bufIdx < BUFFER\_SIZE; bufIdx++)
111. next[tempIdx + bufIdx] = buff[bufIdx];
112. index = 0;
113. }
114. }
115. **else** **if** (edgeSupport[e2] > level) {
116. **if** (e1 < e3 && InCurr[e3]) {
117. **int** supE2 = \_\_sync\_fetch\_and\_sub(&edgeSupport[e2], 1);
118. **if** (supE2 == (level + 1)) {
119. buff[index] = e2;
120. InNext[e2] = **true**;
121. index++;
122. }
123. **if** (supE2 <= level) {
124. \_\_sync\_fetch\_and\_add(&edgeSupport[e2], 1);
125. }
126. **if** (index >= BUFFER\_SIZE) {
127. **long** tempIdx = \_\_sync\_fetch\_and\_add(nextTail, BUFFER\_SIZE);
128. **for** (**long** bufIdx = 0; bufIdx < BUFFER\_SIZE; bufIdx++)
129. next[tempIdx + bufIdx] = buff[bufIdx];
130. index = 0;
131. }
132. }
133. **if** (!InCurr[e3]) {
134. **int** supE2 = \_\_sync\_fetch\_and\_sub(&edgeSupport[e2], 1);
135. **if** (supE2 == (level + 1)) {
136. buff[index] = e2;
137. InNext[e2] = **true**;
138. index++;
139. }
140. **if** (supE2 <= level) {
141. //edgeSupport[e2]++;
142. \_\_sync\_fetch\_and\_add(&edgeSupport[e2], 1);
143. }
144. **if** (index >= BUFFER\_SIZE) {
145. **long** tempIdx = \_\_sync\_fetch\_and\_add(nextTail, BUFFER\_SIZE);
146. **for** (**long** bufIdx = 0; bufIdx < BUFFER\_SIZE; bufIdx++)
147. next[tempIdx + bufIdx] = buff[bufIdx];
148. index = 0;
149. }
150. }
151. }
152. **else** **if** (edgeSupport[e3] > level) {
153. **if** (e1 < e2 && InCurr[e2]) {
154. **int** supE3 = \_\_sync\_fetch\_and\_sub(&edgeSupport[e3], 1);
155. **if** (supE3 == (level + 1)) {
156. buff[index] = e3;
157. InNext[e3] = **true**;
158. index++;
159. }
161. **if** (supE3 <= level) {
162. \_\_sync\_fetch\_and\_add(&edgeSupport[e3], 1);
163. }
164. **if** (index >= BUFFER\_SIZE) {
165. **long** tempIdx = \_\_sync\_fetch\_and\_add(nextTail, BUFFER\_SIZE);
166. **for** (**long** bufIdx = 0; bufIdx < BUFFER\_SIZE; bufIdx++)
167. next[tempIdx + bufIdx] = buff[bufIdx];                                  index = 0;
168. }
169. }
170. **if** (!InCurr[e2]) {
171. **int** supE3 = \_\_sync\_fetch\_and\_sub(&edgeSupport[e3], 1);
172. **if** (supE3 == (level + 1)) {
173. buff[index] = e3;
174. InNext[e3] = **true**;
175. index++;
176. }
177. **if** (supE3 <= level) {
178. \_\_sync\_fetch\_and\_add(&edgeSupport[e3], 1);
179. }
180. **if** (index >= BUFFER\_SIZE) {
181. **long** tempIdx = \_\_sync\_fetch\_and\_add(nextTail, BUFFER\_SIZE);
182. **for** (**long** bufIdx = 0; bufIdx < BUFFER\_SIZE; bufIdx++)
183. next[tempIdx + bufIdx] = buff[bufIdx];
184. index = 0;
185. }
186. }
187. }
188. }
189. j\_index++;
190. k\_index++;
191. }
192. **else** **if** (g->adj[j\_index] < g->adj[k\_index]) {
193. j\_index++;
194. }
195. **else** **if** (g->adj[k\_index] < g->adj[j\_index]) {
196. k\_index++;
197. }
198. }
199. }
200. **if** (index > 0) {
201. **long** tempIdx = \_\_sync\_fetch\_and\_add(nextTail, index);;
202. **for** (**long** bufIdx = 0; bufIdx < index; bufIdx++)
203. next[tempIdx + bufIdx] = buff[bufIdx];
204. }
205. #pragma omp barrier
206. #pragma omp for schedule(static)
207. **for** (**long** i = 0; i < currTail; i++) {
208. UI e = curr[i];
209. processed[e] = **true**;
210. InCurr[e] = **false**;
211. }
212. #pragma omp barrier
213. }

3. 详细程序代码编译运行使用说明

在项目根目录下运行make，生成ktruss文件

运行代码为 ./ktruss -f 文件路径全名

如:

./ktruss -f ./ktruss-data/soc-LiveJournal.tsv

默认线程数为系统最大线程数，若要设置线程数可在运行代码后添加：

./ktruss -f 文件路径全名 -p 线程数

如：

./ktruss -f ./ktruss-data/soc-LiveJournal.tsv -p 5