# **BK算法流程**

BK通过回溯搜索的方式来检查所有的点以枚举所有的极大完全图。BK搜索过程中的搜索路径组成了一个树状搜索结构。BK算法每次访问一个与搜索路径中已经访问的所有节点都相邻的点来扩展搜索，直到搜索路径不能再扩展，这样一条搜索路径上的所有访问点就形成一个极大完全图。

算法CliqueEnumerate显示了BK搜索树的访问过程。搜索树的每一个节点由下面三个点集组成，也就是搜索树的节点状态表示结构。

1. Result集，现有的搜索路径中已经访问过的节点集合
2. Candidate集，不在Result集中且与Result中的每一个点都相邻的点集
3. Not集，与Result中点都相邻且如果与Result集中的点组合将导致产生冗余或者被包含的结果的点集

算法CliqueEnumerate在访问一个节点时将其加入到现有结果集中并重新构建新的Candidate集合Not集。新的Candidate集通过筛选现有的候选集要与包括新加入到结果集中的点在内的所有Result集中的点都相邻。同样Not集也是通过筛选现有的Not集要求与包括新加入的点在内的所有Result点都相邻。CliqueEnumerate算法通过访问候选集中的点。对于每一个搜索树节点，首先访问的点是当前候选点中连接了最多候选点的点（也就是在候选点组成的子图中度数最大的点）。在搜索完第一个点之后只用那些与第一个点不相邻的候选点来扩展搜索路径。这样保证了在每一节点的搜索路径中只搜索最少的那些有可能生成新的且不冗余的点。一旦候选点Candidate中的任意一个点被访问后都会将其加入到Not集中，以表示包含这个点和当前结果集中的的极大完全图已经在新生成的子图状态中考虑了，搜索树中其他兄弟子图节点状态不应当在搜索相关节点。当CliqueEnumerate已经搜索完所有的子节点状态后将回溯到之前的一个未搜索子图状态。显而易见，CliqueEnumerate的搜索过程可以通过栈来实现。

|  |
| --- |
| **算法1**：BK算法 |
| **输入**：数据图G，图G的节点集V以及边集E  **输出**：所有的且不重复、不冗余的极大完全图 |
| 1. Result 🡨 2. Candidate 🡨 V 3. Not 🡨 4. 调用方法CliqueEnumerate(Result, Candidate, Not) |

BK算法

|  |
| --- |
| **方法**CliqueEnumerate(Result, Candidate, Not) |
| 1. **If** Candidate =  **then** 2. **If** Not =  **then** 3. 输出结果Result 4. **Else** 5. fixp 🡨 Candidate 中与其他Candidate中相邻点最多的点 6. cur\_v 🡨fixp; 7. **While** cur\_v  NULL **do** 8. new\_not 🡨Not集中所有与cur\_v相邻的点 9. new\_cand 🡨 Candidate集中与cur\_v相邻的点 10. new\_res 🡨 Result + cur\_v 11. **CliqeEnumerate**(new\_res, new\_cand, new\_not) 12. Not 🡨 Not + cur\_v 13. Candidate 🡨 Candidate – cur\_v 14. **If** Candidate中存在点v与fixp不相邻 **then** 15. Cur\_v 🡨 v 16. **Else** 17. Cur\_v 🡨 NULL 18. **return** |

递归极大完全图枚举方法







## KPlex单机算法

|  |
| --- |
| KPLEX单机算法 |
| **输入**：数据图G，图G的节点集V以及边集E  **输出**：所有的且不重复、不冗余的KPlex |
| 1. Result 🡨 2. Candidate 🡨 V 3. Not 🡨 4. **While** Candidate 5. 依次取 6. Result 🡨 Result + cur\_v 7. 调用方法**FindAllMaximalKplex**(Result, Candidate, Not) 8. Not 🡨 Not + cur\_v 9. Candidate 🡨 Candidate – cur\_v 10. Result 🡨 Result – cur\_v |

|  |
| --- |
| 方法FindAllMaximalKplex(Result, Candidate, Not) |
| 1. Connected\_Candidate 🡨 Candidate中与Result中任意一个点相邻的点 2. Connected\_Not 🡨 Not中与Result中任意一个点相邻的点 3. **While** Connected\_Candidate 4. **If** Connected\_Candidate 5. **If** Connected\_Not 6. 输出Result 7. **Else** 8. Result是冗余结果 9. cur\_v🡨**SelectExpandNode**(Result, Critical\_Res,Connected\_Candidate, Connected\_Not) 10. **If** cur\_v = NULL 11. Exit 12. Candidate 🡨 Candidate – cur\_v 13. Cur\_cand 🡨 Candidate; Cur\_not 🡨 Not; Cur\_res 🡨 Result 14. Cur\_res 🡨 Cur\_res + cur\_v 15. 更新Cur\_cand和Cur\_not中各个点的计数，同时移除计数counter1大于k-1的点 16. Critical\_Res 🡨 Result中与其他节点不相邻个数等于k-1的点 17. 移除Cur\_cand和Cur\_not中与Critical\_Res不相邻的点 18. FindAllMaximalKplex(Result, Candidate, Not) 19. Not 🡨 Not + cur\_v 20. Connected\_Not 🡨 Connected\_Not + cur\_v |

|  |
| --- |
| 方法SelectExpandNode(Result, Critical\_Res,Connected\_Candidate, Connected\_Not) |
| 1. **If** Result  && Critical\_Res 2. **Return** Connected\_Candidate中的第一个点 3. **Else** 4. Prunable\_Not 🡨 Connected\_Not中与Result中所有点都相邻的点 5. **If** Prunable\_Not 6. **If** 7. **Return** NULL 8. **Else** 9. **Return**, 10. **Else** 11. **Return** Connected\_Candidate中的第一个点 |

定义1：顶点与顶点集合相邻，要求满足用表示；顶点与不相邻，要求满足用表示。

用来扩展现有搜索路径S(R,C,N)的C中候选节点v需要满足以下两个条件

条件1：v与R中至少个点相邻

条件2：与中至少个点相邻

在算法中为Result、Candidate和Not集中点都关联了一个计数：counter1，counter1用来记录其关联的顶点与Result中的多少个点不相邻，其中Result的counter1表示Result中的点与中多少个点不相邻。为了满足条件1，只需要保证点的。

定义2：搜索路径上一个子图节点的结果集R中的点与中k-1点不相邻，则称是临界点

在算法中会使用到临界点的集合，为了满足条件2，当S中临界点集合Critical不为空时，要求Candidate中的点与临界点集合中的所有点都相邻。否则，中会至少存在一个临界点与K个点不相邻，也就不能生成正确的K-plex。

算法去冗余的方法是通过最后Connected\_Candidate为空且Connected\_Not不为空表示Connected\_Not中的任意一个点都可以和现有的结果集组成一个K-plex，但是包含点的所有K-plex已经在搜索路径的其他分支上已经生成了，故而无需输出。如果可以在搜索路径的过程中提前判断Connected\_Not中存在一个点不可能被过滤掉，那么就可以保证这个节点下的整棵子树都是冗余的，可以提前剪枝。如果Connected\_Not中存在一个点与Result和Candidate中的所有点都相邻，那么在此之后无论如何扩展这个搜索路径，都无法通过条件1和条件2来将这个点过滤掉。也就是说在此之后所有的结果中Connected\_Not都不为空，此分支可以剪去。

剪枝条件：

在算法中为Not关联一个新的计数器counter2，counter2表示Not中的点与Candidate中有多少个点不相邻。当Not中存在点的counter1 + counter2 = 0时也就是剪枝条件满足了。

为了加快剪枝速度，算法尽可能的使得counter1 + counter2降到0。为此提出一个新的集合Prunable\_Not，Prunable\_Not是Not的一个子集，包含Not中与Result集中所有点都相邻的点，也就是counter1=0的那些点。注意到，每当选择一个新的点v加到Result中来扩展搜索时，Not中与v不相邻的点的counter2计数就会每次降1。如果一直从Candidate中选与Prunable\_Not集中某个点v不相邻的那些点来扩展，那么点v的counter2很快降到0且将不可能存在Not集中的其他点会比v更快满足剪枝条件。显然，为了能够最快地达到剪枝条件应该选择Connected\_Candidate中与Prunable\_Not中counter2最小的点v相邻的那些点。但是在算法刚开始时还不存在临界点集合时Connected\_Candidate中不存在与Prunable\_Not不相邻的点，因此还是需要按顺序取扩展点。详细过程在方法SelectExpandNode中描述。

## BK并行化

Reading input graph

Initial work distribution

Load balancing

Termination Condition

# **Binary算法流程**