一种用于大型二部图重叠社区检测的计算高效框架

Yue Zeng 北京理工大学
bruceez@163.com

Rong-Hua Li 北京理工大学 lironghuabit@126.com

Qiangqiang Dai 北京理工大学
qiangd66@gmail.com

Guoren Wang 北京理工大学 wanggrbit@126.com

arXiv:2512.19426v1 [cs.SI] 22 Dec 2025

摘要

社区检测旨在发现网络中紧密连接的顶点群组，对于社交网络、推荐系统及其他领域的应用至关重要。现实世界中的网络通常具有二部结构（顶点分属于两个不相交集合，集合间存在连接），这给专门的社区检测方法带来了独特挑战。双团渗流社区（BCPC）被广泛用于检测二分图中的紧密结构。完全二部图是完全二部子图；当极大二分团通过邻接关系相连（共享一个 (α, β)-完全二部图）时，就形成了 BCPC。然而，由于潜在的极大二部团邻接图（MBAG）规模可能极其庞大，现有的 BCPC 检测方法面临较高的时间复杂度。为解决这一问题，我们提出了一种新颖的基于部分 BCPC 的方法，其核心思想是利用部分 BCPC 来缩小 MBAG 的规模。部分 BCPC 是 BCPC 的子集。属于同一部分 BCPC 的极大二分团也必然属于同一个 BCPC。因此，可以将这些极大二分团在 MBAG 中归为一个顶点，从而显著缩小 MBAG 的规模。此外，我们突破了 MBAG 的局限，提出了一种基于 (α, β)-完全二部图枚举的新型 BCPC 检测方法。该方法通过枚举所有 (α, β)-完全二部图，并连接共享同一 (α, β)-完全二部图的极大二分团来检测 BCPC；共享同一完全二部图正是极大二分团彼此邻接的条件。该方法还利用部分 BCPC，大幅剪枝 (α, β)-完全二部图的枚举空间。实验表明，我们的方法性能优于现有方法，速度提升近三个数量级。

1 引言

社区检测旨在发现网络中紧密连接的顶点群，是网络分析中的一项核心任务 [10]。它在社交网络 [26]、生物系统 [25] 和推荐系统 [7]等应用中发挥着关键作用。在许多现实场景中，网络天然适合建模为二分图，表示为 $G = (U, V, E)$，其中顶点被划分为两个不相交集合 U和 V，且连接只发生在U与 V 之间。典型示例包括推荐系统中的用户–物品网络 [22] 以及文献计量学中的作者–论文网络 [4]。这种结构给社区检测带来了新的挑战，促使人们开发针对二分图的专用方法。

双团渗流社区（BCPC）是一种被广泛研究的方法，用于检测二分图中的凝聚结构 [6, 11–15, 33]。完全二部图是一种完全二部子图，而二部团渗流社区被定义为通过邻接相连的二部团的极大集合。在（α, β）-二部团渗流社区中，如果两个二部团在一侧至少共享 α 个顶点，在另一侧至少共享 β 个顶点（共享一个（α, β）-二部团），则认为它们相邻 。双团渗流社区（BCPC）在各种二部网络场景中有着广泛的应用。在社交网络中（用户和实体作为顶点，交互作为边），BCPC能够发现基于兴趣的用户群体，从而支持兴趣推荐或用户推荐 [11]。在维基百科网络中（条目和编辑者作为顶点，编辑活动作为边），BCPC能够识别簇，揭示由主题驱动的编辑者聚集和协同内容创作 [13]。在互联网资源访问场景中（资源和访问者作为顶点），BCPC用于分析学生与课程资源之间的关系及其时间演化 [12, 33]。在企业异构图中（主机和用户作为顶点，交互作为边），BCPC用于检测簇，以帮助识别用户异常行为，并追踪危害网络安全的恶意活动 [15]。BCPC允许簇之间存在重叠，这一特性至关重要，因为现实世界中的用户或实体很可能属于多个社区。这一特性也已得到广泛研究 [11–13, 15]。

尽管 BCPC 方法具有广泛的实际应用价值，但由于其时间复杂度较高，仍面临显著的计算挑战。据我们所知，现有的 BCPC 方法均基于 [14]。其框架后来在 [27] 中实现，并用于为个性化搜索问题构建索引 [6]。其基本思想基于最大二部团邻接图 （MBAG）。该方法首先枚举所有极大二部团，然后将其作为顶点构建最大二部团邻接图。在该图中，当且仅当两个相应极大二部团的重叠部分包含一个（α， β）-二部团时，这两个顶点之间才存在一条边。邻接图中的连通分量与（α， β）-BCPC 一一对应。然而，现实世界的二分图可能包含大量极大二部团（理论上最多可达 $2^{n/2}$ [9]），由这些极大二部团构成的邻接图也可能包含大量边（最多可达 10^{11}，，见表1）。遍历如此大规模的邻接图非常耗时。

为克服现有方法的局限性，我们首先提出一种基于部分 BCPC 的解决方案，旨在缩小 MBAG 的规模。其核心思想是在枚举过程中利用极大双团枚举树的前缀。具体而言，我们引入部分 BCPC 这一新概念，它

在枚举极大双团时即时计算。在枚举过程中，如果枚举树的某个前缀包含来自顶点集 U 的至少 α 个顶点，以及来自顶点集 V 的至少 β 个顶点，我们就可以确定该分支下的所有极大双团都属于同一个 BCPC。这些二部团共同构成一个不完整的 BCPC，称为部分 BCPC。同一部分 BCPC 中的极大双团可视为 MBAG 中的单个顶点，从而缩小 MBAG 的规模。实验表明，我们的方法能够大幅减少 MBAG 中的顶点数量，降幅达两个数量级（见实验3的图 5(c)），从而实现更高效的 BCPC 遍历与计算。

尽管部分 BCPC 方案在简化 MBAG 上运行，但仍需对该结构进行完整遍历，而该结构的规模可能依然大得难以承受。为克服这一局限，我们提出了一种基于 (α,β)-二部团枚举的新方法。其基本思想是在二分图中枚举所有 (α,β)-二部团，并针对每个枚举出的 (α,β)-二部团 B，使用并查集连接所有包含 B 的极大二分团。这一方法之所以有效，是因为两个极大二分团只有在共享一个 (α,β)-二部团时才相邻。由于 (α,β)-二部团的数量会随着 α,β [30, 31] 呈指数增长，我们再次利用部分 BCPC 剪枝 (α,β)-二部团枚举树。在枚举过程中，对于生成的每个中间小型二部团，我们都可以维护包含它的极大二分团集合。如果该集合为空，或集合中的所有极大二分团均已属于同一个部分 BCPC，则可以安全地剪去当前枚举分支。实验表明，使用部分 BCPC 可将 (α,β)-二部团枚举树中的节点数量减少五个数量级以上（Exp-4）。因此，与最先进的算法（SOTA）（实验1）相比，该方法的速度提升近三个数量级。

我们的贡献总结如下。

新颖的部分BCPC 方法。我们提出了一种新颖的部分 BCPC 方法，其特点是引入了 BCPC 的一种宽松定义，即部分 BCPC。借助部分 BCPC，我们可以显著缩减现有方法所用 MBAG 的规模，从而直接加速遍历过程。

MBAG。我们证明，借助部分 BCPC，MBAG 中的顶点数量最多可减少两个数量级（即消除 MBAG 中多达 99% 的顶点，见实验3）。

新颖的 $(\alpha,\ \beta)$-二部团方法。不同于依赖遍历 MBAG 的传统方法，我们提出了一种新颖的框架，直接枚举 $(\alpha,\ \beta)$-二部团，并利用并查集连接共享这些二部团的极大二部团。为降低枚举 $(\alpha,\ \beta)$-二部团的成本，我们再次借助部分 BCPC，使枚举树中的节点数量最多减少五个数量级（Exp-4）。

大量实验。我们在10个真实世界数据集上进行了实验，实验结果证明了所提出算法的有效性和效率。具体而言，我们在广泛的参数设置下测试了算法的性能。例如，在数据集 YouTube 上，当 $\alpha=2,\beta=2$ 时，最先进方法（MBAG）耗时 13,376 秒，而基于部分 BCPC 的解决方案（ PBCPC+ ）耗时 197 秒，性能提升近两个数量级，

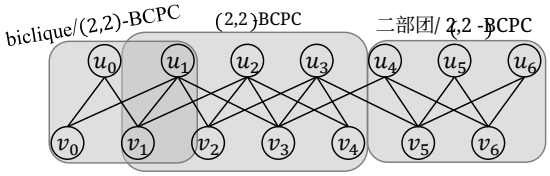


图1：二分图 G与$(\alpha,\ \beta)$-BCPC $(\alpha=2,\ \beta=2)$

相比之下，基于（α,β）-完全二部图的解决方案（BicliqueP）耗时 34 秒，性能提升近三个数量级。此外，我们对 BCPC 在社区检测中的应用进行了案例研究。结果表明，与其他传统社区模型（如完全二部图、Bitruss、双核）相比，BCPC 能够发现更多具有实际意义的社区，这证明了 BCPC 的有效性。

2 预备知识

设 $G=(U,V,E)$ 为无向无权二部图，其中 U,V 是两个不相交的顶点集，E 是由形如 (u,v) $u\in U,v\in V$ 的边构成的集合。对于顶点 $u\in U$，$Nei_G(u)$ 表示 u 的邻居集合，即 $Nei_G(u)=\{v|(u,v)\in E\}$。若二分图 $G'=(U'V'E')$ 满足 $U'\subseteq U,V'\subseteq V,E'\subseteq E$，则它是 G 的子图。

定义 1（完全二部图）。给定一个二分图 $G=(U,V,E)$，完全二部图是由顶点集 $B=(X,Y)$ 构成的一对，其中 $X\subseteq U,Y\subseteq V$ 和 $\forall u\in X,v\in Y,\ (u,v)\in E$。

为简单起见，我们令 $B.X=X,B.Y=Y$，若 $B=(X,Y)$。对于两个二部团 $B,B',B\subseteq B'$ 表示 $B.X\subseteq B'.X,B.Y\subseteq B'.Y$。极大二分团 B 是指不存在其他满足 $B\subseteq B'$ 的二部团 B'。

定义 2（(α,β)-二部团邻接）。给定两个整数 α β B_1 B_2 B_1 B_2 α β $|B_1.X\cap$、以及两个二部团，，和，$B_2.X|\geq\alpha\ |B_1.Y\cap B_2.Y|\geq\beta$ 若，，则它们是(,)-邻接的。

定义 3（(α,β)-二部团连通性）。给定两个整数 α,β以及两个二部团 B_1,B_n,B_1 和 B_n，若存在一个二部团序列 $B_1,B_2,\ldots,B_n$，使得 $\forall i=1,\ldots,n-1,B_i$ 和 B_{i+1} 是(α,β)-邻接的，则 B_1和 B_n是(α,β)-连通的。

定义 4 ((α,β)-二部团渗流社区 ((α,β)-BCPC)). 给定一个二分图 G以及两个整数 α、β，G中的一个(α,β)-二部团渗流社区是由极大二分团组成的极大集合，且集合中的任意两个极大二分团均(α,β)-连通。

示例*1*. 设 $\alpha=2$、$\beta=2$，图*1*展示了一个二分图 G以及其中一些 *(2, 2)-BCPC*。这些 *BCPC* 分别位于左侧、中间和右侧。中间的 *(2, 2)-BCPC* 由三个极大二分团组成：$B_1=(\{u_1$、$u_2\}$，$\{v_1$、v_2、$v_3\})$、$B_2=(\{u_1$、u_2、$u_3\}$，$\{v_2$、$v_3\})$ 和 $B_3=(\{u_2$、$u_3\}$，$\{v_2$、v_3、$v_4\})$。在这些二部团中，B_1 和 B_2 相邻，因为它们的交集 $(\{u_1$、$u_2\}$，$\{v_2$、$v_3\})$ 满足定义 *2* 中的条件。同理，B_2 和 B_3 也相邻。因此，B_1、B_2 和 B_3 可以共同构成一个有效的 *(2, 2)-BCPC*。图*1*的左侧和右侧分别是两个极大二分团。这些极大二分团与其他任何极大二分团都不相邻（定义 *2* ），因此可以视为两个特殊的 *BCPC*。

算法1： 基本的极大双团枚举 [9]中的框架

输入： 二分图 $G=(U,\ V,\ E)$
输出： G中的所有极大双团
1 枚举 $(\emptyset,\ \emptyset,\ U,\ V,\ \emptyset,\ \emptyset)$;
2 **过程** 枚举 $(R_U,\ R_V,\ C_U,\ C_V,\ X_U,\ X_V)$
3 **如果** $C_U \cup C_V = \emptyset$ **则**
4 **如果** $X_U \cup X_V = \emptyset$ **则** 输出 $(R_U,\ R_V)$ 作为一个极大二分团;
5 **返回**;
6 **对于每个** $u \in C_U$ **do**
7 分支 $(R_U,\ R_V,\ u,\ C_U - \{u\}\ C_V,\ X_U,\ X_V)$;
8 $C_U \leftarrow C_U - \{u\}; X_U \leftarrow X_U \cup \{u\}$;
9 **对每个** $v \in C_V$ **执行**
10 分支 $(R_V,\ R_U,\ v,\ C_V - \{v\}\ C_U,\ X_V,\ X_U)$;
11 $C_V \leftarrow C_V - \{v\}; X_V \leftarrow X_V \cup \{v\}$;
12 **过程** 分支 $(R_U,\ R_V,\ u,\ C_U,\ C_V,\ X_U,\ X_V)$
13 $C'_V \leftarrow C_V \cap Nei_G(u)$; $X'_V \leftarrow X_V \cap Nei_G(u)$;
14 枚举 $(R_U \cup \{u\}\ R_V, C_U, C'V, X_U, X'V)$;

问题陈述。 给定一个二分图 G 和两个整数α、 β，我们的目标是检测所有 $(\alpha$、 $\beta)$-BCPC（BCPC 检测）。

2.1 极大双团枚举

现有最先进的极大二分团枚举方法 [1, 5, 9] 都基于集合枚举树结构：通过迭代遍历顶点来构建枚举树的节点，每个节点代表一个不同的完全二部图。本文采用 [9] 中的算法框架来进行极大二分团枚举，并在第 4 节设计基于部分 BCPC 的方法。这是因为 [9] 中的方法在极大二分团枚举方面达到了最先进的性能；此外， [9] 中的框架适用于范围更广的凝聚子图模型，因此我们在第 4 节提出的基于部分 BCPC 的方法也有望扩展到二分图中的其他凝聚子图模型。

基本框架 [9]。算法1介绍了在 [9]中枚举极大二分团的基本框架。该框架通过递归探索候选顶点集来构建集合枚举树，同时利用排除集避免枚举非极大二分团。算法1首先在第1行调用递归过程 Enum。Enum 的参数包括当前二部团（R_U, R_V）、用于扩展的候选顶点（C_U, C_V）以及已处理的排除顶点（X_U, X_V）。在 Enum 中，算法首先检查终止条件：$C_U \cup C_V = \emptyset$（第3行）。如果终止条件满足且$X_U \cup X_V = \emptyset$（第4行），则意味着（R_U, R_V）是一个极大二分团。随后，算法遍历候选集中的每个顶点 u C_U （第6行），并调用

分支过程用于执行分支扩展（第 7 行），并将 u从候选集 C_U移至排除集 X_U（第 8 行）。类似地，算法会对 C_V中的每个顶点 v执行相同操作（第 9–11 行）。在分支过程中，算法计算新的候选集$C'_V = C_V \cap Nei_G$（u）和新的排除集 $X'_V = X_V \cap Nei_G$（u），将 u添加到 R_U，并递归调用枚举过程，以进一步探索该分支中的极大二分团。

枢轴技术 [9]。 上述框架的一个显著问题是，它会产生大量不必要的

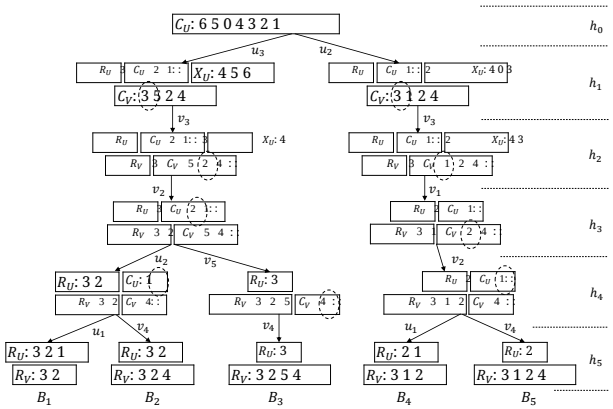


图 2：框架在[9] 中、针对图1所示二分图生成的部分枚举树。枢轴顶点以虚线圆圈标出。（B_i， h_j）表示枚举树中 B_i 分支上第h_j 层的树节点

计算，这些计算会生成非极大的二部团。我们首先提出以下观察：对于二分图 $G=(U,V)$ 中的任意顶点 $u \in U$，以及该二分图中的任意极大二分团 $B=(X,Y)$，以下两个条件中必有且仅有一个成立：

（1） B包含 u（即 $u \in X$）。（2） B包含一个顶点 v，其中 v是 V中与 u非邻接的顶点（即 $\exists v \in V - Nei_G$（u） $v \in Y$）。

基于上述观察，我们可以得出结论：在算法1的第6至11行中，从 X_U、 C_U、 X_V或 C_V中选择一个枢轴顶点 u，并且仅枚举u（如果 $u \notin X_U$， X_V）以及 C_V中不与 u 相邻的顶点（如果 $u \in X_V$， C_V，则为 C_U中的顶点），即可确保枚举出所有极大双团，同时减少产生非极大二部团的不必要枚举分支。选择枢轴顶点的一种有效方法是，从 C_V（或 C_U）中选择在 $X_U \cup C_U$（或 $X_V \cup C_V$）中非邻接点最少的顶点作为枢轴顶点。

顶点排序 [5]。 极大二分团枚举算法的效率通常受顶点处理顺序的影响。 [9] 将 [5] 中介绍的二跳度排序应用于其框架。此顶点排序仅应用于递归搜索的第一层，而前文提到的枢轴剪枝技术则用于递归搜索的后续层。

示例*2*。图 *2* 展示了图*1*中的二分图 $G=(U,V,E)$ 上极大双团枚举的部分枚举树 *[9]*。位于第 h_0 层的根节点，以该二分图的顶点集 U 初始化其候选集 C_U，并按二跳度排序。根节点仅包含 C_U，不包含 C_V。第 h_0 层的枚举过程遵循排序后的序列，首先处理 u_6、u_5、 u_0 和 u_4，随后处理 u_3 和 u_2；图中仅展示处理 u_3 和 u_2 的分支。对于树节点 $nd(B_1, h_1)$，将 u_3 加入 R_U，并更新 C_V，使其在 V 中包含 $Nei_G(u_3)$。$nd(B_1, h_1)$ 中的排除集 X_U 包含 u_4、 u_5 和 u_6，因为这些顶点已在先前的分支中枚举，并在第 h_0 层加入 X_U（最初，第 h_0 层的 X_U 为空，因此未显示）。

在 $nd(B_1, h_1)$ 处，框架选择 v_3作为枢轴顶点。因此，跳过 C_V中除 v_3外的顶点，以及 $Nei_G(v_3) \cap C_U$

算法 2： (α, β)-完全二部图枚举框架
在 [30] 中

输入： 二分图 $G=(U, V, E)$；两个整数 α，β
输出： G中的所有(α, β)-二部团
1 在顶点集 U [30] 上构造二跳图 $\vec{H}=(U, \vec{E_h})$；/* $\vec{H}=(U, \vec{E_h})$：一个直接图，$(u, v) \in \vec{E_h}$ 表示 $u<v, u, v$ 在 V */
2 双团列举 ($\vec{H}, \emptyset, V$);
3 **过程**双团列举 ($\vec{H}, R_U, C_V$)
4 **如果** $|R_U|=\alpha$ **那么**
5 **对于每个** $R_V \subseteq C_V, |R_V|=\beta$ **执行** 输出 (R_U, R_V) 作为一个(α, β)-完全二部图;
6 **返回**;
7 **对于每个** $u \in \vec{H}$ **do**
8 $C'_V \leftarrow C_V \cap Nei_G(u)$;
9 **如果** $|C_{r_V}|<\beta$ 或 $Nei_{\vec{H}}(u)<\alpha-|R_U|-1$ **那么继续**;
10 Let $\vec{H}_r$ 的诱导子图 $\vec{H}$在 Nei-上 $_H(u)$;
11 双团列举（ $\vec{H}' R_U \cup\{u\}$ $C'V$);

(u_2, u_1) 会被跳过。因此，在第 h_2 层，仅扩展枚举 v_3 的分支。在第 h_2 层，将 v_3 添加到 R_V，并从 C_V 中移除。第 h_2 层的集合 C_U和 X_U也会以类似方式更新，以进一步细化搜索空间和排除集。当候选集 C_U和 C_V均为空（即位于枚举树的叶节点处），且排除集 X_U和 X_V也为空时，(R_U, R_V)是一个极大二分团。

2.2 (α, β)-完全二部图枚举

最先进的现有（α, β）-完全二部图枚举方法在 [30]中提出。给定一个二分图 $G=(U, V, E)$，其核心思想是在 U 集中枚举顶点组合，同时维护它们在 V集中的公共邻居。此外，该方法还在 U集的顶点上构建一个普通的有向图，称为二跳图，以加速枚举过程。

算法 2 展示了基本框架。它首先构建二跳图 H，并以 U作为其顶点集（第 1 行）。如果二分图中集合 V内的节点 $u<v$和 u, v至少有一个共同邻居，则存在一条边 (u, v)。在第 4 行，如果 R_U已包含 α个顶点，则只需从 C_V中枚举所有由 β个顶点组成的组合，即可得到 (α, β)-二部团。在第 7 行，算法枚举个顶点 u，这些顶点位于 H中；随后在第 8 行更新 V中的候选集 C'_V，并在第 11 行继续进行下一层枚举。

2.3 并查集 [8]

并查集，也称为不相交集数据结构，是一种高效管理一组互不相交集合的方法。每个集合都表示为一棵树，树中的节点代表集合的元素。在这棵树中，每个节点都指向其父节点，而树的根节点则作为该集合的代表。这种树形结构支持快速的**合并**和**查找** 操作，因此并查集非常适用于图连通性问题等任务。**查找(i)** 用于获取包含 i的唯一集合的代表；**合并(i, j)** 用于将分别包含 i和 j的两个集合合并为一个集合。具体而言，在相应的树结构中，将一个集合的代表节点设为另一个集合代表节点的父节点。

在所有实际场景中，并查集的**union**和**find** 操作都能达到常数均摊时间复杂度。

算法 3： 现有的基于 MBAG 的解决方案

输入： 二分图 $G=（U, V, E）$；两个整数 α，β
输出： G中的所有 BCPC
1 $\mathcal{B} \leftarrow$ 个极大二分团的集合;
2 B $\leftarrow\{B | B \in$ B $|B.X| \geq \alpha, |B.Y| \geq \beta\}$;
3 $Nei（B）\leftarrow$ 个与 B共享顶点的极大二分团集合;
4 $UF \leftarrow$一个初始并查集，即 $UF.find(B)=B$对于任意输入 B;
5 **对于** $B \in \mathcal{B}$ **执行**
6 **对于** $B_r \in Nei(B)$ **执行**
7 **如果** $|B.X \cap B_r.X| \geq \alpha$且 $|B.Y \cap B_r.Y| \geq \beta$**则** $UF.union(B, B_r)$;
8 **返回** UF;

该结构的空间复杂度为 $O(n)$，其中 n 表示其管理的元素总数 [8]。

3 现有解决方案

现有的 BCPC 检测方法在 [6, 14, 27] 中被提出并实现。据我们所知，尚无其他工作主要关注 BCPC 检测的算法实现与改进。其基本思想基于 最大二部团邻接图 （MBAG）。该方法首先枚举所有极大二分团，然后将其作为顶点构造最大二部团邻接图。在该图中，当且仅当相应两个极大二分团的重叠部分包含一个（α, β）-二部团时，两个顶点之间才存在一条边（满足定义 2）。邻接图中的连通分量与（α, β）-BCPC 一一对应。在该方法中，通过经典并查集 [8] 维护（α, β）-BCPC 中极大二分团的（α, β）-二部团连通性。

算法 3 是现有的基于 MBAG 的算法，用于识别二分图 $G=$（U, V, E）中的所有（α, β）-BCPC。该算法首先枚举 G 中的所有极大二分团，并根据大小约束对其进行筛选（第 1–2 行）。原因在于，对于极大二分团 B，如果 $|B.X|<\alpha$ 或 $|B.Y|<\beta$，则 B 无法与其他极大二分团共同形成 BCPC。在第 4 行初始化并查集 UF。随后，算法遍历每个极大二分团 B 及其邻居 $B' \in Nei\ B$ $|B.X \cap B'.X| \geq \alpha$ $|B.Y \cap B'.Y| \geq$（ ），β 5 7 B B' 检查，（第 - 行）。如果满足条件，则通过 $UF.union$（$*$）将 和 合并到同一个 BCPC 中（第 7 行）。处理完所有极大二分团后，并查集 UF 表示二分图中的最终 BCPC。

这种方法的主要缺点是，MBAG 的规模可能极其庞大（最大可达 10^{11}, ，见表1）。因此，遍历如此庞大的 MBAG 以获取所有 BCPC 显然是不切实际的。

4 新颖的部分式-BCPC 基础解决方案

正如我们在第 3 节中所讨论的，当前依赖 MBAG（算法 3）的方法由于 MBAG 规模庞大而面临重大挑战。这一庞大的规模导致遍历操作效率低下。因此，很明显，优化基于 MBAG 的方法的性能关键在于有效缩减 MBAG 的规模。

关键思想。 我们通过合并 MBAG 中的顶点来缩小其规模。关键是在枚举过程中利用极大二分团枚举树的前缀。在枚举过程中，如果枚举树的某个前缀包含至少 α

个顶点于 U和 β个顶点于 V中，我们可以确定该分支下的所有极大双团都属于同一个 BCPC。无需遍历这些极大双团之间的连接。这些二分团共同构成一个不完整的 BCPC，称为部分 BCPC。

例 3. 设 $\alpha = \beta = 1$。考虑图 2，其中以节点 $nd(B_1, h_2)$为根的子树包含三个极大二分团分支：B_1、B_2 和 B_3。这三个极大二分团都包含 $nd(B_1, h_2).R_U$和 $nd(B_1, h_2).R_V$。因此，我们可以将 B_1、B_2 和 B_3视为构成一个不完整的 BCPC，即部分 BCPC。

接下来，我们将正式定义部分 BCPC、极大二分团枚举树，以及用于计算部分 BCPC 的特殊前缀。

定义 5（部分 BCPC）。给定一个二分图 G和两个整数 α、β，部分 BCPC $\mathcal{P}$ 是满足 $\mathcal{P} \subseteq \mathcal{B}$ 的极大二分团集合，其中 $\mathcal{B}$ 是 G 中的一个 (α, β)-BCPC 。

与定义 4 的区别在于，定义 5 不要求部分 BCPC 中的极大二分团集合是极大的。这体现了部分 BCPC 作为不完整 BCPC 的特性。在实际应用中，部分 BCPC 也通过并查集维护。

定义 6（极大双团枚举树（*MBE* 树））。给定一个二分图 $G = (U, V, E)$。*MBE* 树 $T = (N, \vec{E})$ 表示在 G 上运行极大二分团枚举算法的搜索空间。这里，N 表示树中的节点集合，其中每个节点 $nd \in N$ 都包含一个六元组 $(R_U, R_V, C_U, C_V, X_U, X_V)$，其定义见第 2.1 节中的 *MBE* 基本框架。集合 $\vec{E}$ 由形如 (nd_i, nd_j, u) 的有向边组成，其中 $u \in U$ 或 $u \in V$。一条边 $(nd_i, nd_j, u) \in E$ 表示节点 nd_j 是节点 nd_i 的子节点，它是通过从 $nd_i.C_U$ 或 $nd_i.C_V$ 中选取顶点 u 生成的。若 $nd.C_U \cup nd.C_V = \emptyset$，则节点 $nd \in N$ 是叶节点。对于叶节点 $nd \in N$，若 $nd.X_U \cup nd.X_V = \emptyset$，则 $(nd.R_U, nd.R_V)$ 是一个极大二分团。

为简洁起见，我们分别用 $nd.RCX_U$ $nd.RCX_V$ $nd.R_U \cup nd.C_U \cup nd.X_U$ and $nd.R_V \cup nd.C_V \cup nd.X_V$ 表示 and。

定义 7（(α, β)-前缀和 (α, β)-节点）。给定一个二部图 $G = (U, V, E)$ 及其 *MBE* 树 $T = (N, E)$，设 $nd_1 \in N$为 T的根节点，$PF(nd_l) = (nd_1, nd_2, ..., nd_l)$ 是从 nd_1到nd_l的路径，即 $\forall i = 1, 2, ..., l-1, (nd_i, nd_{i+1}, *) \in \vec{E}$。$PF(nd_l)$ 是一个 (α, β)-前缀，当且仅当 $|nd_l.R_U| = \alpha, |nd_{l-1}.R_U| < \alpha, |nd_l.R_V| \geq \beta$或 $|nd_l.R_V| = \beta, |nd_{l-1}.R_V| < \beta, |nd_l.R_U| \geq \alpha$。此时，$nd_l$是一个 (α, β)-节点。

简言之，(α β nd $|nd.R_U| \geq$, α $|nd.R_V| \geq \beta$ nd)-节点是满足条件且在其所在的所有分支中均成立的第一个节点。对于它的所有祖先节点 nd_i，不可能同时满足$|nd_i.R_U| \geq \alpha$和 $|nd_i.R_V| \geq \beta$。

定理 1. 对于任意两个(α, β)-节点 nd_1, nd_2，nd_1 都不能是 nd_2 的祖先或后代。

证明。根据定义 7，nd_1 是满足$|nd_1.R_U| \geq \alpha$和 $|nd_1.R_V| \geq \beta$的第一个节点，且位于 nd_1所在的所有分支中。nd_2也同理。因此，nd_1，nd_2 位于两个不同的分支中，而 nd_1 不可能是 nd_2的祖先或后代。□

算法 4:基于部分 BCPC 的解决方案

输入： 二分图 $G = (U, V, E)$；两个整数 α, β
输出： G中的所有 BCPC

令 $T = (N, \vec{E})$ 是 G的MBE 树，G中的极大双团集合$\mathcal{B}$；
$Nei(B) \leftarrow$ 与 B共享顶点的极大双团集合；
令 UF为一个初始并查集；
令$\mathcal{P} \leftarrow \emptyset$;
/* 计算部分BCPC */
对于(α, β)-节点 nd属于 T，**执行**
    ProcessNode (nd)；
    对 $B \in \mathcal{P}$ **中的每个元素执行** $UF.union(B, \mathcal{P}[0])$;
    $P \leftarrow \emptyset$;
/* 遍历 MBAG中的部分-BCPC */
对 $B \in \mathcal{B}$ **中的每个元素执行**
    对 $B\prime \in Nei(B)$ 和 $B, B\prime$ 属于不同的部分-BCPC **则**
        如果 $|B.X \cap B\prime.X| \geq \alpha, |B.Y \cap B\prime.Y| \geq \beta$**那么** $UF.union(B, B\prime)$;
返回 UF;
/* 递归遍历MBE树以收集所有极大双团在 $node$下 */
过程ProcessNode $(node)$
对于 $(node, node\prime u) \in \vec{E}$**执行**
    如果 $node\prime.C_U \cup node\prime.C_V = \emptyset$ **则**
        如果 $node\prime.X_U \cup node\prime.X_V = \emptyset$ **则**
            $P \leftarrow P \cup \{(node\prime.R_U, node\prime.R_V)\}$; /* $(node\prime.R_U, node\prime.R_V)$ 是一个极大二分团 */
    否则 ProcessNode $(node\prime)$;

4.1 基本框架

本节介绍基于部分 BCPC 的解决方案的基本框架。该框架包含两个基本步骤：（1）利用 MBE 树计算部分 BCPC；（2）基于部分 BCPC 遍历简化 MBAG，得出最终的 BCPC。在计算部分 BCPC 时，该框架使用并查集连接 MBE 树中特定节点所根植子树内的极大双团。具体而言，我们关注 MBE 树中的（α，β）-节点。(α，β)-节点的祖先会被忽略，因为它们不满足 $|R_U| \geq \alpha$和 $|R_V| \geq \beta$。(α，β)-节点的后代虽然满足 $|R_U| \geq \alpha$ 和 $|R_V| \geq \beta$，但它们是冗余的，因为任何共享与这些后代对应前缀的极大双团分支，也必然共享与该(α，β)-节点对应的（α，β）-前缀。因此，只关注（α，β）-节点即可高效计算部分 BCPC。

实现。算法 4 展示了基本框架。首先进行极大双团枚举，并提取 MBE 树$T = (N, \vec{E})$（第 1 行）。然后初始化极大双团的邻居列表和并查集 UF，以管理部分 BCPC 和 BCPC（第 2–3 行）。在第 5–8 行中，算法遍历每个 (α，β)-节点 nd（第 5 行），调用 ProcessNode 函数，将以 nd为根节点的子树下的所有极大双团收集到 P 中（第 6 行）。随后，将 P 中的所有极大双团合并到同一个部分 BCPC 中（第 7 行）。在第 9–11 行中，算法基于部分 BCPC 遍历规模更小的 MBAG。具体而言，对于每个极大双团 B，算法只遍历与其属于不同部分 BCPC 的极大双团 B'（第 10 行）。最后，算法返回 UF，即所有检测到的 BCPC。

MBE 树的存储。实际上，我们不会存储 MBE 树的所有分支。我们首先介绍 MBE 树中的两种节点。

定义 8（真实节点与虚拟节点）。 给定一个图 G 及其MBE树 $T=(N,E)$，对于任意节点 $nd \in N$，如果 nd 是叶节点且满足 $nd.X_U = nd.X_V = \emptyset$，那么 nd 是真实节点——此时，$(nd.R_U, nd.R_V)$ 是一个极大二分团（见定义 6）。否则，如果 nd 至少有一个真实节点后代，则它也是一个真实节点。如果这两个条件都不满足，那么 nd 是一个虚拟节点。

算法 4 中的 ProcessNode 过程只需识别包含极大二分团的分支（第 15–17 行），也就是由真实节点组成的分支。因此，实践中我们只在 MBE 树中存储真实节点。

分析。 我们首先分析算法 4 的正确性，然后考察算法 4 得到的部分 BCPC 的数量与极大双团数量以及 BCPC 数量之间的关系，最后分析算法 4 的复杂度。

定理 2。 *算法 4 能正确计算所有 BCPC。*

证明。我们首先证明这些部分 BCPC 均正确。ProcessNode 过程会将给定 *node* 为根节点的子树中的所有极大二分团收集到 P 中。由于参数始终是一个 (α,β)-节点，因此该过程每次执行时收集到的极大二分团必定共享一个 (α,β)-二分团。因此，这些二分团正确地构成一个部分 BCPC。然后我们证明最终得到的 BCPC 均正确。在第 9–11 行中，属于不同部分 BCPC、但满足邻接条件的任意两个极大二分团，都会通过并查集 UF连接。因此，并查集 UF维护的每个不相交集都对应一个有效的 BCPC。□

定理 3。 *给定一个二分图 G、两个整数 α,β和 MBE树 $T=(N,E)$，设 $n_{biclique}$ 为极大二分团的数量，$ND \leftarrow \{nd | nd \in N,\ nd$ 是一个 (α,β)- 节点，nd 是一个真实节点 $\}$，$pbcpc$ 为算法 4 计算出的部分 BCPC 数量，$bcpc$ 为 BCPC 的数量，则以下不等式成立：$n_{biclique} \geq |ND| = pbcpc \geq bcpc$。*

证明。根据定理 1，ND 中任意两个 (α,β)-节点都不能位于同一分支中，并且 ND 中的节点都是真实节点，因此，ND 中的每个节点在其子树中至少有一个极大二分团。因此，$n_{biclique} \geq |ND|$。由于算法 4 将 ND 中每个 (α,β)-节点子树中的所有极大二分团连接成一个部分 BCPC，并且不同 (α,β)-节点的子树中的极大二分团互不相同，因此，ND 中的每个 (α,β)-节点都对应一个部分 BCPC ($|ND| = pbcpc$)。根据定义 5，部分 BCPC 是不完整 BCPC，因此，$pbcpc \geq bcpc$。□

定理 4。 *算法 4 的时间和空间复杂度分别为$O((1-\frac{1}{pbcpc})n^2_{biclique}\overline{n})$和 $O(n_{biclique}n)$，其中 $n = |U| + |V|$，$pbcpc$ 表示部分 BCPC 的数量，$n_{biclique}$ 表示极大双团的数量。*

证明。设 $m = |E|$，极大双团枚举的时间复杂度为 $O(2^{n/2}m)$ [9]。与极大双团枚举相比，算法 4 引入了额外操作：遍历 (α,β)-节点，并从以这些节点为根的子树中收集极大双团分支。由于 (α,β)-节点不位于同一分支中（定理 1），这些额外操作的时间复杂度不超过 $O(2^{n/2}m)$。在第 9–11 行中，在最坏情况下，算法 4 仅跳过每个初始部分 BCPC 中极大双团之间的内部互访。设 $x = n_{biclique}, y = pbcpc$ 和 $p_1, p_2, ..., p_y$ 分别为每个部分 BCPC 中极大双团的数量，则第 9–11 行的时间复杂度为 $O(x^2n - (p_1^2 + p_2^2 + _... + p_y^2)n) = O(x^2n - (\frac{x}{y})^2 yn)$（根据柯西-施瓦茨不等式），可写为 $O((1-\frac{1}{pbcpc})n^2_{biclique}n)$。因此，算法 4 的时间复杂度为 $O(2^{n/2}m + (1-\frac{1}{pbcpc})n^2_{biclique}n)$。由于 $O(n_{biclique}) = O(2^{n/2})$ [9, 24],，时间复杂度可改写为 $O((1-\frac{1}{pbcpc})n^2_{biclique}n)$。

关于空间复杂度，算法 4 的主要空间开销来自 MBE 树和极大二分团。由于 MBE 树中只存储真实节点，因此 MBE 树中的分支数量等于极大二分团的数量，所以算法 4 的空间复杂度为 $O(n_{biclique}n)$。

□

讨论。 定理 3 中的不等式 $n_{biclique} \geq pbcpc$ 构成了利用部分 BCPC 缩减 MBAG 的基础。这是因为同一部分 BCPC 中的极大二分团在 MBAG 中被视为单个顶点，从而使 MBAG 的顶点数量显著减少。定理 4 中的时间复杂度也体现了这一点：如果将 $pbcpc$ 替换为$n_{biclique}$，结果恰好就是第 3 节中遍历完整 MBAG 的现有方法的复杂度（$O(n^2_{biclique}n)$）。实验（图 5 (c)）表明，$pbcpc$ 可能比$n_{biclique}$小一个数量级。

4.2 优化

超越以 (α,β)-节点为根的子树。上一节中的基本框架会直接将以 (α,β)-节点为根的子树中的极大双团合并为一个部分 BCPC。然而，这种方法并不充分，因为对于一个 (α,β)-节点 nd，并非所有共享 $nd.R_U$和 $nd.R_V$的极大双团都位于以 nd为根的子树中。例如，在图 2 中，如果 $\alpha = \beta = 1, nd(B_4, h_2)$ 是一个 (α,β)-节点，但 B_1和 B_2不在以 $nd(B_4, h_2)$ 为根的子树中，尽管 B_1和 B_2都共享 $nd(B_4, h_2).R_U$和 $nd(B_4, h_2).R_V$。这种现象的出现是因为 B_1和 B_2中的某些顶点（例如 u_3）较早出现在根节点的候选集 C_U中，导致 B_1和 B_2被更早地枚举。

要合并（α，β）节点的子树内外所有相关的极大双团，必须充分利用（α，β）节点中维护的全部信息，即 R_U、R_V、C_U、C_V、X_U 和 X_V，以便在 MBE 树上进行系统搜索。具体而言，对于每个（α，β）节点 nd，我们使用 $nd.RCX_U$ 和 $nd.RCX_V$ 在 MBE 树中搜索所有包含 $nd.R_U$ 和 $nd.R_V$ 的极大双团分支，并将它们连接到同一个部分 BCPC 中。

例 4。 令 $\alpha = 1$、$\beta = 1$，图3（a）说明，在处理（α、β）-节点 nd 时，既要考虑位于以 nd 为根的子树内的分支，也要考虑该子树之外的分支。可以观察到，共享（$nd.R_U$、$nd.R_V$）的极大双团分支不仅包括 nd 下的 B_4 和 B_5，还包括位于以 nd 为根的子树之外的极大双团分支 B_1 和 B_2。

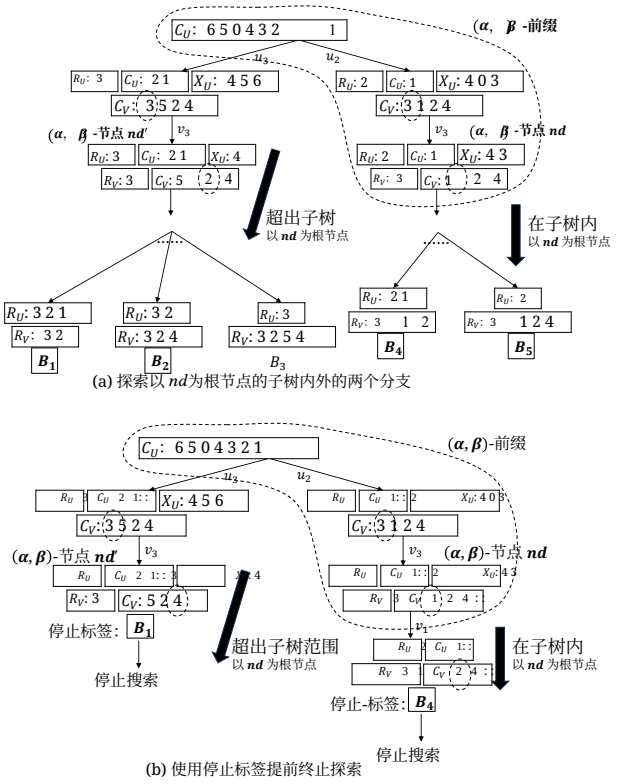


图3：优化示例

停止标签。上述方法在处理每个（α，β）-节点时，可能需要完整遍历所有相关的极大双团分支。为降低遍历所有共享 $nd.R_U$ 和 $nd.R_V$ 的极大双团分支所产生的开销（nd 是一个（α，β）-节点），同时不影响正确性，我们为 MBE 树中的某些真实节点分配停止标签。当访问此类真实节点时，停止标签可立即获取其子树中的所有极大二分团信息，从而无需继续遍历其后代。给定 α，β，下面介绍停止标签的具体细节：

(1) 停止标签仅分配给 MBE 树中的真实节点。(2) 对于 (α, β)-节点的后代节点 nd，设置 nd 的停止标签的前提是，将以 nd 为根节点的子树中的所有极大二分团连接到同一个部分 BCPC 中。nd 的停止标签设置为该部分 BCPC 中的一个极大二分团。(3) 对于 (α，β)-节点 nd，设置 nd 的停止标签的前提是，将共享 $nd.R_U$ 和 $nd.R_V$ 的所有极大二分团连接到同一个部分 BCPC 中。nd 的停止标签设置为该部分 BCPC 中的一个极大二分团

现在我们说明停止标签如何帮助剪枝以识别分支。

(1) 对于 (α,β)-节点的后代节点 nd，在为 nd设置停止标签之前，只需使用并查集将其所有子节点（也包括真实节点）的停止标签合并到同一个部分 BCPC 中。这样就无需遍历以 nd 为根节点的子树中的所有极大双团分支。

(2) 对于 (α，β)-节点 nd，在识别所有共享 $nd.R_U$和 $nd.R_V$的极大双团分支时，一旦遇到带有停止标签的节点，搜索便会终止，并直接将该停止标签加入集合 P。

算法 5：基于部分 BCPC 的解决方案及优化项

输入： 二分图 $G=(U, V, E)$；两个整数 α，β
输出： 所有BCPC 在 G

1 算法 4 的第 1-3 行;
2 令$\mathcal{P} \leftarrow \emptyset$, $root \leftarrow$ 的根节点 T;
3 令 $ST(nd)$ $nd \in N$为 nd的停止标签;
/* 计算部分-BCPCs */
4 后序遍历 ($root$);
/* 遍历 MBAG 中具有部分 BCPC 的节点 */
5 算法 4 的第 9–11 行;
6 **返回** UF;
7 **过程**后序遍历 ($node$)
8 **对** $node$ 的每个子节点 $node_i$，按 $node_i$ 的创建顺序**执行** 后序遍历 ($node_i$) ;
9 **如果** $node$ 是一个(α, β) 节点**则**
10 ProcessNode+ ($root$, $node.RCX_U \cup node.RCX_V$, $node.R_U \cup node.R_V$);
11 **否则，如果** $node$ 是一个真实节点，且是某个 (α, β)-节点的后代 **则**
12 ProcessNode+ ($node$);
13 **如果** $\mathcal{P}$ 非空 **则**
14 **对** $B \in \mathcal{P}$ **中的每个元素执行** $UF.union(B, \mathcal{P}[0])$;
15 $ST(node) \leftarrow \mathrm{P}[0]$, $\mathrm{P} \leftarrow \emptyset$;
16 **过程** ProcessNode+ ($node$, UV, RUV)
17 **对于**的每个子节点 $node$, $node_i$，按 $node_i$ **的创建顺序执行**
18 设$(node, node_i u) \in \vec{E}$;
19 **如果** $node_i$ 是一个真实节点**那么**
20 **如果** $u \in UV$ **那么**
21 **如果** $ST(node_i) = None$ **那么** ProcessNode+ ($node_i UV$, RUV) ;
22 **否则** $\mathrm{P} \leftarrow \mathrm{P} \cup \{ST(node_i')\}$;
23 **如果** $u \in RUV$ **则跳出**;
24 **过程**ProcessNode+ ($node$)
25 **如果** $node$ 是叶节点**则** $\mathcal{P} \leftarrow \mathcal{P} \cup \{(node.R_U, node.R_V)\}$;
26 **否则**
27 **对于每个** $node_i$, $node_i$ 是 $node$ 的子节点且为真实节点**执行**
28 $\mathrm{P} \leftarrow \mathrm{P} \cup \{ST(node_i')\}$

示例5。 设 $\alpha = 1, \beta = 1$，在图*3 (b)*中处理 nd 时：

搜索以 nd为根节点的子树时，遇到 nd的子节点的停止标签后，处理便会终止。该停止标签为 B_4，其中 B_5 与 B_4属于同一个部分-BCPC 。

在搜索以 nd为根节点的子树之外的部分时，遇到 nd'的停止标签后，搜索便会终止。

最后，两个停止标签（B_1 和 B_4）相连；与此同时，它们所属的部分-BCPC合并为一个更大的部分- BCPC。

实现。基于上述策略，我们修改了原有框架，并提出算法 5。算法 5 在第 1 行执行初始化步骤，其方式与算法 4 类似。在第 3 行，$ST(nd)$ 表示 nd的停止标签。算法 5 的核心组件是后序遍历函数（第 4 行和第 7 行）。该函数对 MBE 树执行后序遍历，这一过程可以在 MBE 进程期间执行。后序遍历函数负责处理（α，β）节点本身——在第 10 行调用 ProcessNode+ ($node$, UV, RUV)——以及其后代节点中的真实节点——调用 ProcessNode+ ($node$)

在第 12 行。两种处理例程都可能向 P 中添加某些极大双团。最后，在第 13–15 行，算法 5 将 P 中的所有极大双团连接到同一个部分 BCPC 中，并将 P 中的第一个二部团设置为 *node* 的停止标签（第 15 行）。与算法 4 类似，算法 5 基于部分 BCPC 遍历 MBAG，以获得最终的 BCPC（第 5 行）。

处理一个 (α,β)-节点时（第 10 行），ProcessNode+ 函数会从 MBE 树的根节点开始遍历。它主要探索真实节点（第 19 行），因为任何包含极大二分团的分支都必须由真实节点构成。此外，遍历路径上的任何顶点 u 都必须属于集合 UV（第 20 行）。遇到带有停止标签的节点后，递归调用会立即结束（第 22 行）。处理完一个子节点$node'$后，如果遍历路径上的顶点 u 属于集合RUV，则无需探索其余子节点及其对应的分支（第 23 行）。这是因为根据 MBE算法的枚举原理，后续分支中不可能找到包含 RUV 的任何极大二分团。由于该函数旨在识别包含 RUV的极大二分团，因此在这种情况下无需继续遍历。

处理 (α,β)-节点的后代节点时（第 12 行），如果参数 *node* 已经是叶节点，则可以直接将其表示的极大二分团加入 P。否则，将其所有子节点的停止标签加入 P。

分析。现在我们开始分析算法 5 的正确性。关键在于证明后序遍历中（第 3 行）计算出的部分 BCPC 是正确的。

定理 5。对于算法 *5*（第 *9* 行）中处理的每个 (α,β)-节点，ProcessNode+ 会将二分图中所有共享$(node.R_U, node.R_V)$ 的极大二分团连接到同一个部分BCPC中。

证明。考虑按后序遍历（即后序遍历）的处理顺序排列的一系列 (α,β)-节点 $(nd_1, nd_2, \ldots, nd_n)$。我们首先证明，在处理 nd_1期间，ProcessNode+ 会将所有共享 $(nd_1.R_U, nd_1.R_V)$ 的极大二分团连接到同一个部分 BCPC 中。由于 nd_1 是第一个 (α,β)-节点，因此它也必定属于第一个枚举出的极大双团分支（如果 nd_1 出现在更早的分支中，那么该分支或更早的某个分支很可能是第一个包含极大二分团的分支；如果 nd_1 出现在更晚的分支中，那么它不可能是第一个 (α,β)-节点）。因此，所有包含 $(nd_1.R_U, nd_1.R_V)$ 的极大双团分支都必须位于以 nd_1为根的子树中，并且不会出现在任何先于 nd_1处理的子树中。

根据数学归纳法，假设在处理 nd_1，nd_2，$\ldots nd_i$ 时，ProcessNode+连接了所有共享 $(nd_i.R_U,\ nd_i.R_V)$ 的极大双团。接下来，我们证明 $i+1$ 的情况。对于 nd_{i+1}，令 $R_U = nd_{i+1}.R_U$ 和 $R_V = nd_{i+1}.R_V$。根据 MBE 过程 [9],，包含具有 R_U 和 R_V 的极大双团的分支，要么在 nd_{i+1} 之前生成，要么位于以 nd_{i+1} 为根节点的子树中。对于在 nd_{i+1} 之前生成的极大双团分支，其对应的 $(\alpha,\ \beta)$ 节点 nd_j 必须满足 $j < i+1$，因此我们只需收集 $ST(nd_j)$，因为它表示这些极大双团的部分 BCPC。对于 nd_{i+1} 子树中的极大双团，我们只需收集 nd_{i+1} 的子节点的停止标签，因为每个子节点下的极大双团已经连接成部分 BCPC（因为遍历顺序为后序）。因此，nd_{i+1} 的情况得证。□

定理 6. 算法 *5* 能正确计算出所有 BCPC。

证明。该定理的证明与定理 2 的证明类似。定理 5 证明了，算法 5 中的部分 BCPC 是通过连接共享 (R_U和 R_V) 的 α,β 节点的极大二分团得到的，因此这些部分 BCPC 是正确的。此外，算法 5（第 5 行）中计算最终 BCPC 的过程与算法 4 中的过程相同，因此算法 5 能正确计算 BCPC。□

需要指出的是，算法 5 计算得到的部分 BCPC 与算法 4 计算得到的并不相同。下面的定理对此进行了说明。

定理 7. 给定一个二分图 G和两个整数 α,β，设 $pbcpc$为算法 *4* 计算得到的部分-BCPC的数量，$pbcpc$+为算法 *5* 计算得到的部分-BCPC的数量，$bcpc$为 BCPC的数量，则以下不等式成立：$pbcpc \geq pbcpc+ \geq bcpc$。

证明。我们只需证明 $pbcpc \geq pbcpc+$。在处理 (α,β)-节点时，算法 5 不仅会连接以该节点为根的子树中的极大二分团，还会进一步连接在先前分支中枚举且共享 (α,β)-节点的 R_U, R_V 的极大二分团。此外，算法 5 中的 ProcessNode+过程也会处理属于虚拟节点的 (α,β)-节点。因此，$pbcpc \geq pbcpc+$。□

定理 8。算法 *5* 的时间和空间复杂度分别为$O((1-\frac{1}{pbcpc+})n^2_{biclique}n)$和 $O(n_{biclique}n)$，其中 $n=|U|+|V|$，$pbcpc+$ 是部分 *BCPC* 的数量，$n_{biclique}$ 是极大二分团的数量。

证明。设 $m=|E|$，极大双团枚举的时间复杂度为 $O(2^{n/2}m)$ [9]。与算法 4 不同，算法 5 不仅处理 (α,β)-节点（第 9–10 行），还处理其子节点（第 11–12 行）。处理 (α,β)-节点的子节点时，算法 5 只需遍历其子节点的一层，因此总体时间复杂度不超过 $O(2^{n/2}m)$。对于 (α,β)-节点的处理，设 x 为此类节点的总数（包括真实节点和虚拟节点），d 为真实 (α,β)-节点的最大深度。处理这些节点中的每个节点时（第 10 行），ProcessNode+ 过程至多访问所有真实 (α,β)-节点。由于 MBE 树中每个节点的子节点数不超过 n，上述过程的总时间复杂度为 $O(xn^d) = O(2^{n/2}mn^d)$。算法 5 中 MBAG 遍历的时间复杂度分析与算法 4 类似。因此，算法 5 的时间复杂度为 $O(2^{n/2}mn^d + (1-\frac{1}{pbcpc+})n^2_{biclique}n)$。由于$O(n_{biclique}) = O(2^{n/2})$ [9, 24],，时间复杂度可改写为 $O((1-\frac{1}{pbcpc+})n^2_{biclique}n)$。

与算法 4 类似，算法 5 的空间复杂度也为$n_{biclique}n$。□

算法 6： 基于(α, β)-完全二部图的解决方案

输入： 二分图 $G = (U, V, E)$；两个整数 α, β
输出： G中的所有BCPC

令 UF为一个初始并查集;
$\mathcal{B} \leftarrow$ 中所有极大二分团的集合;
$\mathrm{B} \leftarrow \{B | B \in \mathrm{B} \ |B.X| \geq \alpha, |B.Y| \geq \beta\}$;
在顶点集 U [30]上构造二跳图 $\vec{H} = (U, \vec{E_h})$; /* $\vec{H} = (U, \vec{E_h})$: 一个有向图，$(u, v) \in \vec{E_h}$ 表示 $u < v$、u、v 在 V */
双团列举 ($\vec{H}, \emptyset, V$);
返回 UF;

过程双团列举($\vec{H}, R_U, C_V$)
如果 $|R_U| = \alpha$ **则**
  对于每个 $R_V \subseteq C_V, |R_V| = \beta$**执行** 连接共享 (R_U, R_V) 的极大双团在$\mathcal{B}$中，满足 $UF.union(*)$;
  返回;
对每个 $u \in \vec{H}$ **do**
  $C'_V \leftarrow C_V \cap Nei_G(u)$;
  如果 $|C'_V| < \beta$ 或 $Nei_{\vec{H}}(u) < \alpha - |R_U| - 1$ **则继续**;
  Let $\vec{H}'$ 是诱导子图的 $\vec{H}$ 开启 $Nei_{\vec{H}}(u)$;
  双团列举 ($\vec{H}', R_U \cup \{u\}$ $C'V$);

讨论。 定理 7 中的不等式 $pbcpc \geq pbcpc+$ 构成了算法 5 优于算法 4 的基础，因为它意味着算法 5 可以遍历规模更小的 MBAG。实验（图 5）表明， $pbcpc+$ 显著小于 $pbcpc$——例如小了一个数量级，而算法 5 的速度可以比算法 4 快一个数量级（表 2）。

5 基于（α， β）-完全二部图的新型解决方案

现有的 BCPC 检测方法都基于 MBAG 方法。在上一节中，我们利用部分 BCPC 缩减了 MBAG，但本质上仍然基于 MBAG；对于 MBAG 极其庞大的某些数据集，该方法仍然效率低下。本节将介绍一种新型 BCPC 检测框架，该框架也可以利用第 4 节提出的部分 BCPC 来加速检测过程。我们首先介绍这一框架。

5.1 基本框架

核心思想。 注意，在上一节中，MBAG 中的每条边本质上都表示一个（a， b）-二部团，其中 $a \geq \alpha$和 $b \geq \beta$。一种直接的想法是枚举二分图中的所有（α， β）-二部团，并对每个枚举出的（α， β）-二部团 B，使用并查集连接所有包含 B的极大二分团。

实现。 算法 6 展示了基于（α， β）-二部团的解决方案的基本框架。该框架建立在 [30], 提出的（α， β）-二部团枚举框架之上，其核心思想是在维护 V集中公共邻居的同时，枚举 U集中的顶点组合。该框架在 U集的顶点上构建了一个普通的有向图，称为二跳图，以加速枚举过程。

在第 2–3 行，算法 6 准备需要连接的极大双团。在第 4 行，它根据 [30] 构建二跳图。随后，在第 5 行，它执行 (α, β)-二部团枚举框架。在双团列举过程中，对于每个枚举出的 (α, β)-完全二部图（第 9 行），所有共享该完全二部图的极大双团都会连接在一起。

分析。 现在我们继续分析算法 6 的正确性和复杂度。

定理 9. 算法 6 能正确计算所有BCPC。

证明。该定理很容易证明如下：算法 6 能够枚举所有 (α, β)-二部团，并连接所有共享这些二部团的极大二分团。因此，最终得到的 BCPC 均正确。□

定理 10. 算法 6 的时间和空间复杂度分别为$O(a(\vec{H})^{\alpha-2}|\vec{E_h}||V|+\Delta n_{biclique})$和 $O(n_{biclique}n)$，其中 $n = |U| + |V|$, $a(H)$ 是 H [30]的树木度， Δ 是 (α, β)-二部团的数量，$n_{biclique}$ 是极大双团的数量。

证明。极大双团枚举的时间复杂度为 $T_{biclique} = 2^{n/2}|E|$ [9]。枚举 (α, β)-二部团的时间复杂度为 $O(a(\vec{H})^{\alpha-2}|\vec{E_h}||V| + \Delta)$ [30]。算法 6 连接所有共享一个 (α, β)-二部团的极大双团（第 9 行），因此，算法 6 的时间复杂度为 $O(T_{biclique} + a(\vec{H})^{\alpha-2}|\vec{E_h}||V| + \Delta n_{biclique})$ $O(n_{biclique}) = O(2^{n/2})$ [9, 24] $O(\Delta) = O(n^{\alpha+\beta})$ 。由于 ( ) ( )， ( ) ( ) [30], $O(a(\vec{H})^{\alpha-2}|\vec{E_h}||V| + \Delta n_{biclique})$，时间复杂度可重写为 (( ))。

算法 6 不会存储所有 (α, β)-二部团，因此，算法 6 的空间复杂度为 $O(n_{biclique}n)$。□

5.2 使用极大双团和部分-BCPC 进行剪枝

基本框架面临的挑战。 显然，算法 6 中的基本框架效率很低，因为 (α β α β ,)-二部团的数量会随着 和 的增加呈指数增长。因此，枚举所有 [30] α β (,)-二部团并连接相应的极大二分团是不切实际的。

使用极大二分团进行剪枝。解决上述挑战的关键是在 (α, β)-二部团的枚举过程中缩小搜索空间——也就是说，预先剪去那些无法促成连接多个极大二分团的分支。

定理 11。 给定一个二分图 G，对于一个二部团 B，如果与其共享的极大二分团数量为 0，或者所有这些极大二分团都属于同一个 BCPC，那么对于满足 $B \subseteq B'$ 的任意其他二部团 B'，与 B' 共享的极大二分团数量也为 0，或者它们属于同一个 BCPC。

证明。显然，任何包含B' 的极大双团也必须包含 B，因此，包含B' 的极大双团集合是包含 B的极大双团集合的子集。因此，定理 11很容易得证。□

定理 11 揭示了剪枝（α， β）-二部团枚举空间的核心思想：在枚举过程中，对于生成的每个中间小型二部团，我们可以维护包含它的极大双团集合。如果该集合为空，或集合中的所有极大双团已经属于同一个 BCPC，则可以安全地剪去当前枚举分支。

算法 7： (α, β)-基于完全二部图的解法，包含极大完全二部图

输入： 二分图 $G=(U, V, E)$；两个整数 α, β
输出： G中的所有BCPC

令 UF为一个初始并查集;
$\mathcal{B} \leftarrow$ 中所有极大双团的集合 G;
$\mathrm{B} \leftarrow \{B | B \in \mathrm{B}\ |B.X| \geq \alpha,\ |B.Y| \geq \beta\}$;
在顶点集 U [30]上构造二跳图 $\vec{H}=(U, \vec{E_h})$；/* $\vec{H}=(U, \vec{E_h})$: 一个有向图, $(u, v) \in \vec{E_h}$ 表示 $u<v$、 u, v 在{v?}中有公共邻居 */
双团列举+ ($\vec{H}, \emptyset, V$);
返回 UF ;

过程双团列举+ ($\vec{H}, R_U, C_V$)
$RB \leftarrow \{B | R_U \subseteq B.X, B \in \mathcal{B}\}$; /* 共享 R_U 的极大双团 */
如果 $|\{UF.find(B) | B \in RB\}| \leq 1$ **则返回；** /* 仅有 1 或 0 UF 不相交集 in RB */
如果 $|R_U|=\alpha$ **则**
  对每个 $R_V \subseteq C_V, |R_V|=\beta$ **执行**
    $RB \leftarrow \{B | R_V \subseteq B.Y, B \in RB\}$;
    如果 $|\{UF.find(B) | B \in RB\}| \leq 1$ **则继续；** /* 仅有 1 或 0 UF 在 RB中的不相交集 */
    将 RB中的极大双团与 $UF.union$（$*$）相连;
  返回;
对每个 $u \in \vec{H}$ **do**
  $C'_V \leftarrow C_V \cap Nei_G(u)$;
  如果 $|C'_V|<\beta$ 或 $Nei_{\vec{H}}(u)<\alpha-|R_U|-1$ **则继续**;
  Let $\vec{H}'$ 的诱导子图 $\vec{H}$ 于 $Nei_{\vec{H}}(u)$;
  双团列举+ ($\vec{H}' R_U \cup \{u\}\ C'V$);

实现。 算法 7 展示了使用极大双团剪枝（α, β）-二部团枚举过程的实现。算法 7 与算法 6 的区别在于 BicliqueListing+ 函数。具体而言，在第 8 行，BicliqueListing+ 收集与（R_U, $\emptyset$）共享的极大双团。在第 9 行，如果 RB 中的极大双团仅属于一个或零个并查集，则可以剪去整个枚举分支。在第 12–13 行，算法进一步检查与 R_V 共享 RB 的 RB 中极大双团是否属于多个并查集。如果是，则使用 $UF.union(*)$ 将它们连接起来。

分析。 现在我们继续分析算法 7 的正确性和复杂度。

定理12。 算法 7 能正确计算所有 BCPC。

证明。与算法 6 相比，算法 7 根据定理 11 剪除无法连接多个 UF 集的分支。因此，算法 7 与算法 6 具有相同的正确性。□

定理13。 算法 7 的时间和空间复杂度分别为$O((a(\vec{H}_m)^{\alpha-2}|\vec{E}_{hm}||V_m|+\Delta_m)n_{biclique})$和 $O(n_{biclique}n)$ $n=|U|+|V|$。设 U_m和 V_m 为任意两个极大二分团共享的顶点集， G_m 为由 U_m 和V_m诱导的子图，$\vec{H}_m=(U_m, \vec{E}_{hm})$ 为基于 G_m的二跳图。 $a(\vec{H}_m)$ 为 H_m [30]的树木度， Δ_m 为 G_m中(α, β)-二部团的数量，$n_{biclique}$ 为极大二分团的数量。

证明。极大双团枚举的时间复杂度为 $T_{biclique}=2^{n/2}|E|$ [9]。与算法 6 相比，算法 7 根据定理 11 剪除了无法连接多个 UF 集的分支。这意味着，最初仅包含在一个极大二分团中的顶点

算法 8： (α, β)-基于完全二部图的解决方案（部分）-BCPC

输入： 二分图 $G=(U, V, E)$；两个整数 α, β
输出： G中的所有BCPC
算法 5 的第 1–3 行；/* 部分BCPC 计算 */
算法 7 的第 2–5 行；/* 通过(α, β)-二分团连接极大双团列表 */
返回 UF;

无需在（α, β）-完全二部图枚举过程中予以考虑。因此，可以将算法 7 视为在 G_m上执行（α, β）-完全二部图枚举。考虑算法 7 的第 8–9 行和第 12–13 行，其时间复杂度为 $O(T_{biclique}+(a(\vec{H}_m)^{\alpha-2}|\vec{E}_{hm}||V_m|+\Delta_m)n_{biclique})$。由于$O(n_{biclique})=O(2^{n/2})$ [9, 24]， $O(\Delta_m)=O((U_m+V_m)^{\alpha+\beta})$ [30]，时间复杂度可改写为 $O\ a(\vec{H}_m)^{\alpha-2}|\vec{E}_{hm}||V_m|+(((\ ))\Delta_m n_{biclique})$。

同样，算法 7 的空间复杂度仍为 $O(n_{biclique}n)$。□

基于部分 BCPC 的剪枝BCPC。 如第 4 节所示，部分 BCPC 是不完整的 BCPC。基于部分 BCPC，我们可以预先确定某些极大双团属于同一个 BCPC。因此，它们使算法 7 中的 (α, β)-二部团枚举过程能够更早达到剪枝条件。

实现。 算法 8 是一种利用部分 BCPC 的算法，主要由两部分组成。首先，它计算部分 BCPC（第 1 行），并将其记录在并查集结构（UF）中。然后，在第 2 行中，利用这些部分 BCPC 进一步剪枝 (α, β)-二部团列举过程。

分析。 现在我们来分析算法 8 的正确性和复杂度。

定理 14。 算法 *8* 能正确计算所有BCPC。

定理 *15*。算法 *8* 的时间和空间复杂度分别为$O((a(\vec{H}_p)^{\alpha-2}|\vec{E}_{hp}||V_p|+\Delta_p)n_{biclique})$和 $O(n_{biclique}n)$。$n=|U|+|V|$。设 U_p 和 V_p是任意两个部分 *BCPC* 共有的顶点集， G_p是 U_p和 V_p上的诱导子图， $H_p=(U_p, \vec{E}_{hp})$是基于 G_p的二跳图。 $a(\vec{H}_p)$是H_p [30]的树木度， Δ_p 是 G_p中 (α, β)-二部团的数量， $n_{biclique}$ 是极大双团的数量。

定理 14 和定理 15 的证明与定理12 和定理13 的证明类似。

6 实验

6.1 实验设置

数据集。 我们使用了 10 个真实世界的二分图数据集，其详细信息见表1。所有数据集均下载自 http://konect.cc/。

算法。我们首先将 [6, 14] 中最先进的算法实现为 MBAG。

- MBAG（算法 3）直接基于极大双团邻接图（也简称为 MBAG）。它首先

表1：数据集，d_U是U中顶点的平均度，d_V是V中顶点的平均度，N_m是极大双团的数量，N_o是极大二部团邻接图中的边数，K=10^3、M=10^6、B=10^9，"-"表示超时

数据集	$\|U\|$	$\|V\|$	$\|E\|$	$\bar{d}_U$	$\bar{d}_V$	N_m	N_o
YouTube	94,238	30,087	293,360	3.11	9.75	1,769,331	2B
Bookcrossing	77,802	185,955	433,652	5.57	2.33	155,391	1B
GitHub	56,519	120,867	440,237	7.79	3.64	55,260,550	-
CiteSeer	105,353	181,395	512,267	4.86	2.82	54,083	962K
Stack Overflow	545,195	96,678	1,301,942	2.39	13.47	2,922,148	26B
Twitter	175,214	530,418	1,890,661	10.79	3.56	5,102,542	13B
Imdb	685,568	186,414	2,715,604	3.96	14.57	1,809,175	1B
Actor2	303,617	896,302	3,782,463	12.46	4.22	3,761,666	2B
亚马逊	2,146,057	1,230,915	5,743,258	2.68	4.67	5,702,211	108B
DBLP	1,953,085	5,624,219	12,282,059	6.29	2.18	1,281,831	85M

枚举所有极大双团，然后使用这些二部团构建极大双团邻接图，最后遍历该邻接图以计算 BCPC。

随后，我们实现了以下基于部分 BCPC 的解决方案和基于 (α, β)-二部团的解决方案。

- PBCPC（算法 4）是基本的基于部分 BCPC 的解决方案。它首先枚举极大双团，并构建 MBE 树。对于 MBE 树中的每个 (α, β)-节点，它会识别与该节点相关联的所有极大双团，并将它们连接成一个部分 BCPC。然后，它遍历经部分 BCPC 简化后的 MBAG，以获得最终的 BCPC。

- PBCPC+ （算法 5）是 PBCPC 的改进版本。其关键区别在于使用停止标签，以减少识别与 (α, β)-节点相关联的所有极大双团时的搜索开销。

- Biclique（算法 6）是基于（α, β）-二部团枚举的基本方案。其核心思想是枚举所有（α, β）-二部团，并针对每个这样的二部团，将所有包含它的极大双团连接起来，从而构造 BCPC。• BicliqueM（算法 7）是 Biclique 的改进版本。它通过检查与当前枚举的二部团相关联的极大双团是否已属于同一个 BCPC，来剪枝（α, β）-二部团枚举分支。

- BicliqueP（算法 8）也是 Biclique 的改进版本。与 BicliqueM 不同，它利用 PBCPC 的方法+ 预先计算部分 BCPC，从而识别出一些已属于同一个 BCPC 的极大双团。由此可以实现更强的剪枝效果。

参数。实验中使用了两个参数：α和 β。它们的默认值均设为 4，可取值为 2、4、6 和 8。

所有算法均使用 C++ 实现。实验在运行 Ubuntu 20.04.4 LTS 的系统上进行，系统配备 AMD Ryzen 3990X 2.2GHz CPU 和 256GB 内存。

6.2 实验结果

实验1：各种算法的效率。在本实验中，我们评估了所有算法在各个数据集上的性能，结果如表 2 所示。

总体而言，MBAG 和 PBCPC 算法在大多数数据集上的运行时间表现最差 α，β。这主要是因为 MBAG 算法会遍历原始极大二分团邻接图（MBAG），而该图通常规模很大（见表1）。尽管 PBCPC 算法尝试通过计算部分 BCPC 来缩小 MBAG 的规模，但这些部分 BCPC 不够全面，因此对 MBAG 的剪枝效果有限（我们将在实验3中探讨这些部分 BCPC 的效果）。

PBCPC+ 的表现略优于 MBAG 和 PBCPC，尤其是在 α和 β较小时。例如，在 YouTube 数据集上，当 $\alpha = \beta = 2$时，PBCPC+ 的表现比 MBAG 和 PBCPC 高出一个数量级以上。这是因为当 α和 β较小时，大于 (α, β)-二部团的极大二分团数量更多，从而扩大了最大二部团邻接图的规模。在这种情况下，PBCPC+ 使用部分 BCPC 缩小邻接图规模的效果更加明显。

只有当α和 β都较小时，完全二部图才有相对较好的表现，例如在 $\alpha = \beta = 2$的 YouTube、Bookcrossing 和 DBLP 数据集上。当 α或 β增大时，完全二部图的性能会急剧下降。这是因为随着 α和 β增大，(α, β)-二部团的数量呈指数级增长。

在几乎所有情况下，BicliqueM 和 BicliqueP 都是表现最优的两种算法，性能比 MBAG 高出近三个数量级。例如，在数据集 YouTube 上，当 $\alpha = \beta = 2$时，BicliqueP 仅需 34 秒，而 MBAG 则需要 13,376 秒。此外，BicliqueP 的表现优于 BicliqueM，且在数据集 GitHub 上这一优势更加明显（在许多参数设置下，只有 BicliqueP 能在 7 天内完成，即 604,800 秒）。这是因为 BicliqueP 使用部分 BCPC 进行剪枝，其剪枝能力比极大二分团更强（见实验4）。

实验2：计算部分-BCPC的开销。本实验评估计算部分 BCPC 的开销（其中包含极大双团枚举，这是必要步骤）。具体来说，我们比较算法 PBCPC 和 PBCPC+ 计算部分 BCPC 所用的时间，并将其与极大双团枚举所用的时间进行比较。实验结果如图 4所示。

如图4所示，尽管计算部分 BCPC 会在极大双团枚举的基础上引入额外开销，但新增成本仍在可控范围内。例如，在 YouTube 和 Twitter 数据集上，t_pbcpc 和 t_pbcpc+ 仅略高于单独进行极大双团枚举所花费的时间（t_mbic）。对于 Git Hub 数据集，由于极大双团的数量更多（见表1），且 MBE 树的规模更大，t_pbcpc+ 不再接近 t_mbic 和 t_pbcpc。不过，这种开销是有意义的——性能最佳的算法 BicliqueP 正是基于 PBCPC+ 中计算出的部分 BCPC 构建的（所花费的时间同样为 t_pbcpc+）。值得注意的是，BicliqueP 是唯一一个在 GitHub 数据集上成功通过所有测试的算法。

表 2：所有算法在所有数据集上的性能（秒）。最佳和次佳结果分别以粗体和下划线标示。“-”表示超时（≥ 7 天）

数据集		YouTube				Bookcrossing				GitHub				CiteSeer				Stack Overflow			
α	Algo \ β	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8
2	MBAG	13,376	11,492	6,072	1,902	35	2	0.9	0.7	-	-	-	-	1	0.8	0.7	0.6	32,598 17	,967	4,287	844
	PBCPC	4,651	7,283	6,942	3,207	42	3	1	1	-	-	-	-	1	0.8	0.7	0.6	30,051 25	,220	8,811	2,177
	PBCPC+	197	1,914	2,787	1,484	20	3	1	0.8	44,276	-	-	-	1	0.9	0.8	0.6	2,564	7,144	3,467	987
	完全二部图	135	358	2,995	-	3	26	93,773	-	-	-	-	-	1	1	64	6,999	2,038	3,118	-	-
	BicliqueM	59	70	43	22	**2**	**1**	**0.8**	**0.6**	24,581	-	-	-	1	**0.8**	**0.7**	0.6	486	208	**119**	**80**
	BicliqueP	**34**	**43**	**31**	**20**	3	2	1	1	**7,718**	**34,478**	**33,948**	**14,272**	**1**	0.9	0.8	**0.6**	**339**	**159**	121	103
4	MBAG	13,113	11,358	5,941	1,793	3	**0.7**	**0.5**	**0.4**	-	-	-	-	0.1	0.08	0.07	0.06	27,582 14	,496	3,308	544
	PBCPC	7,748	9,568	7,802	3,197	4	0.9	0.6	0.5	-	-	-	-	0.1	0.09	0.08	0.06	37,632 25	,336	7,496	1,476
	PBCPC+	1,450	2,964	3,404	1,618	4	0.8	0.6	0.5	-	-	-	-	0.1	**0.08**	0.07	0.06	13,222	9,955	3,857	864
	完全二部图	3,754	5,445	5,522	2,877	2	1	0.8	0.6	-	-	-	-	0.1	0.1	0.1	0.3	1,210	5,446	748	11,595
	BicliqueM	54	235	140	50	1	0.8	0.6	0.5	-	-	-	412,886	0.1	0.09	**0.07**	**0.06**	306	539	199	56
	BicliqueP	**28**	**66**	**49**	**24**	**1**	0.9	0.6	0.5	**11,483**	**145,121**	**424,317**	**157,465**	**0.1**	0.09	0.08	0.07	**190**	**159**	**76**	**34**
6	MBAG	10,936	9,409	4,635	1,312	1	0.6	0.5	0.4	-	-	-	-	**0.03**	**0.03**	**0.03**	**0.03**	9,142	4,150	561	45
	PBCPC	7,736	8,370	5,855	2,080	1	0.7	0.5	0.4	-	-	-	-	0.05	0.04	0.04	0.04	12,482	6,893	1,134	89
	PBCPC+	2,500	3,622	3,133	1,121	1	0.7	0.5	0.4	-	-	-	-	0.04	0.03	0.04	0.03	5,758	3,833	775	76
	完全二部图	-	3,289	34,084	3,352	36	0.9	0.5	0.4	-	-	-	-	0.07	0.04	0.04	0.04	-	1,764	2,724	26
	BicliqueM	44	165	178	35	1	**0.6**	0.5	0.4	13,182	444,343	-	55,394	0.04	0.04	0.04	0.04	90	131	40	12
	BicliqueP	**24**	**57**	**56**	**16**	**1**	0.7	**0.5**	**0.4**	**9,450**	**141,003**	**158,184**	**20,785**	0.05	0.04	0.04	0.04	**75**	**72**	**22**	**10**
8	MBAG	7,236	6,102	2,832	744	1	**0.5**	0.4	0.3	-	-	-	-	0.03	0.03	0.03	0.03	1,377	398	21	7
	PBCPC	5,585	5,707	3,596	1,241	1	0.6	0.4	0.3	-	-	-	-	0.03	0.03	0.03	0.03	1,721	592	32	6
	PBCPC+	2,327	2,878	1,991	713	1	0.6	0.4	0.3	-	-	-	-	0.03	0.03	0.03	0.03	1,110	447	29	7
	完全二部图	-	109,345	4,728	8,555	2,315	6	0.4	0.3	-	-	401,158	23,651	0.09	0.04	0.04	0.04	-	77	20	14
	BicliqueM	33	117	101	38	1	0.6	0.4	0.3	5,219	146,064	103,754	2,324	**0.03**	**0.03**	**0.03**	0.03	30	22	9	7
	BicliqueP	**19**	**45**	**33**	**13**	**1**	0.6	**0.4**	**0.3**	**4,066**	**61,855**	**36,416**	**1,252**	0.04	0.04	0.04	**0.03**	**30**	**16**	**9**	**6**
数据集		Twitter				lmdb				Actor2				亚马逊				DBLP			
α	Algo \ β	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8	2	4	6	8
2	MBAG	11,652	5,290	1,817	657	859	706	622	409	7,303	6,650	5,217	3,502	16,282	3,974	580	137	18	**12**	10	9
	PBCPC	47,374	24,237	6,055	2,335	323	188	293	418	1,820	2,285	2,377	2,041	20,703	6,687	1,297	342	28	19	15	13
	PBCPC+	10,580	11,843	3,162	1,337	102	58	90	160	266	524	772	780	2,413	3,067	706	228	23	15	12	10
	完全二部图	685	5,339	4,584	-	55	147	58,284	-	190	875	39,764	-	964	727	-	-	20	176	296,803	-
	BicliqueM	**510**	777	208	108	**35**	**26**	**23**	**21**	88	56	48	47	618	**157**	**79**	**55**	**17**	13	**10**	**9**
	BicliqueP	518	**402**	**152**	**94**	42	33	28	25	**79**	**51**	**44**	**39**	**590**	176	108	82	25	19	16	14
4	MBAG	4,339	2,737	1,236	457	762	687	614	403	7,107	6,586	5,178	3,468	7,856	1,720	227	57	**7**	5	4	3
	PBCPC	11,436	6,764	3,176	1,482	408	355	446	514	1,622	2,251	2,462	2,125	9,234	2,496	427	100	8	5	4	3
	PBCPC+	6,282	4,237	2,004	1,001	99	74	121	185	201	516	794	840	4,219	1,508	304	78	8	5	4	4
	完全二部图	-	12,028	20,544	242,804	92	1,805	1,335	25,431	626	4,996	29,168	526,141	534	2,687	474	9,676	12	7	29	1,834
	BicliqueM	134	129	94	87	**22**	21	26	21	57	131	51	44	303	161	61	**39**	8	5	4	3
	BicliqueP	**134**	**100**	**69**	**64**	23	**18**	**16**	**15**	**47**	**48**	**34**	**31**	**280**	**111**	**61**	42	9	**5**	**4**	**3**
6	MBAG	2,184	1,596	891	298	623	600	536	340	6,859	6,376	5,085	3,343	1,576	349	63	32	3	2	2	2
	PBCPC	3,124	2,404	1,677	690	557	551	605	568	1,988	2,585	2,583	2,270	1,669	499	97	44	3	2	2	2
	PBCPC+	2,666	1,469	1,073	506	169	164	204	237	326	641	933	955	1,035	331	71	36	3	3	2	2
	完全二部图	-	-	134,078	27,608	1,748	835	11,021	10,968	194,453	22,912	49,668	137,995	33,212	1,595	525	806	10,632	6	17	30
	BicliqueM	**90**	81	68	52	**17**	17	22	23	51	121	62	52	**171**	80	**44**	**32**	3	2	2	2
	BicliqueP	95	**58**	**51**	**41**	19	**14**	**14**	**13**	**40**	**43**	**38**	**32**	213	**79**	47	33	**3**	**2**	**2**	**2**
8	MBAG	1,482	1,166	651	210	400	388	340	187	6,381	5,838	4,529	2,958	445	104	41	28	1	1	1	1
	PBCPC	1,479	1,297	994	369	440	437	450	331	2,631	2,922	2,867	2,398	539	143	54	34	1	1	1	2
	PBCPC+	880	729	669	271	187	186	209	193	625	894	1,093	1,023	338	101	45	31	2	1	1	1
	完全二部图	-	-	-	190,333	355,545	2,375	1,985	11,003	-	-	-	-	-	29,402	3,606	355	-	25	61	93
	BicliqueM	**74**	65	53	36	**15**	13	16	16	46	100	151	66	**143**	**59**	**38**	**28**	**1**	1	1	1
	BicliqueP	79	**45**	**36**	**28**	16	**12**	**12**	**11**	**35**	**38**	**46**	**36**	188	64	40	29	2	**1**	**1**	**1**

实验3：部分BCPC在部分BCPC基础解法中的有效性。本实验考察部分 BCPC 在 PBCPC 和 PBCPC+中的有效性。具体而言，我们将这两种算法得到的部分 BCPC 数量，与极大双团数量和 BCPC 数量进行比较。结果如图 5 所示。可以看出，在所有情况下，定理 3 和 7 中的不等式均成立：n_mbic ≥ n_pbcpc ≥ n_pbcpc+ ≥ BCPC。我们还可以观察到，与 n_mbic 和 n_pbcpc 相比，n_pbcpc+ 更接近 BCPC；在某些情况下，它比 n_mbic 小一个数量级以上（见图 5 (a)、(c)）。在 PBCPC 和 PBCPC+ 算法中，MBAG 中的顶点会根据部分 BCPC 重新分组，这意味着简化 MBAG 中的顶点数量等于部分 BCPC 的数量。因此，PBCPC+ 在几乎所有数据集上的表现都优于 PBCPC。

Exp-4：部分BCPC 在(α, β)-完全二部图基础上的解决方案。本实验研究了极大二分团和部分 BCPC 在(α, β)-完全二部图枚举过程中的剪枝效果。图 6 展示了三种算法 Biclique、BicliqueM 和 BicliqueP 在(α, β)-完全二部图枚举树中的枚举节点数量。可以观察到，在 Biclique 中（没有任何剪枝），枚举树中的节点数量比其他两种算法高出四个数量级以上，这

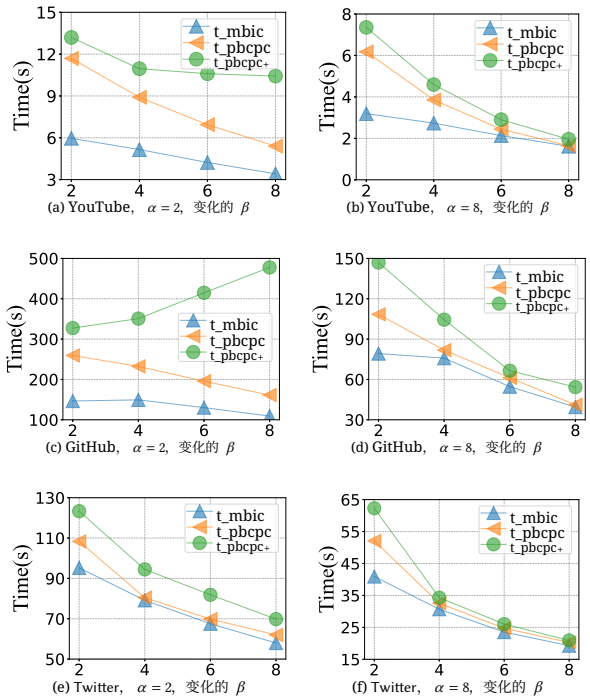


图4：极大双团枚举所用时间（t_mbic）、PBCPC 中的部分 BCPC 计算（包括作为必要步骤的极大双团枚举）以及 PBCPC+ （t_pbcpc+）中的 PBCPC（t_pbcpc）所用时间

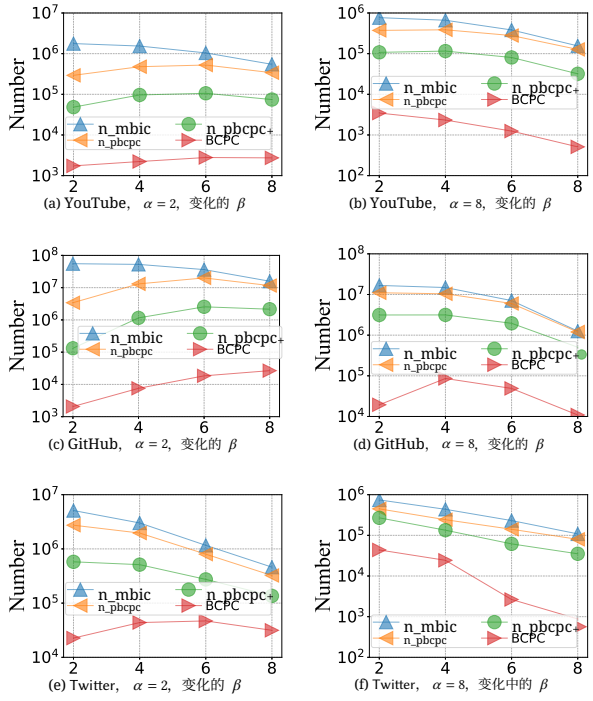


图 5：极大双团的数量（n_mbic，其中 X集和 Y集的大小至少分别为 α 和 β），通过 PBCPC （n_pbcpc）和 PBCPC+ （n_pbcpc+）得到的部分 BCPC，以及 BCPC 的数量

表 3：PBCPC+ 的可扩展性测试

	GitHub	Imdb	Actor2	亚马逊	DBLP
20%	5,779.7	3.4	1.2	8.3	0.8
40%	72,202.2	7.8	5.7	54.5	1.4
60%	119,608.0	13.5	14.7	85.2	2.8
80%	142,445.8	17.5	25.8	107.3	4.5
100%	145,121.7	18.2	49.0	111.2	5.8

表 4：各种算法的内存使用量（MB）

	MBAG	PBCPC	PBCPC+	完全二部图	BiclqueM	BiclqueP
GitHub	10,051	26,965	26,418	9,904	**9,904**	22,158
Imdb	**233**	364	361	260	260	343
Actor2	**409**	725	720	449	449	639
亚马逊	**711**	1,228	1,207	797	772	1,130
DBLP	935	957	970	1,200	**935**	940

充分证明了极大二分团和部分 BCPC 在(α, β)-完全二部图枚举过程中的剪枝能力。在 BiclqueM 和 BiclqueP 中，BiclqueP 中的部分 BCPC 展现出最强的剪枝能力，因为 BiclqueP 中的枚举节点数量比 BiclqueM 少一个数量级（例如，图 6 (c) 中的 $\beta = 2$）。这解释了为什么 BiclqueP 在表 2 的大多数情况下都取得了最佳性能，尤其是在数据集 GitHub 上。

实验 5：可扩展性。 本实验探究了我们表现最佳的算法 BiclqueP 的可扩展性。表 3 展示了 BiclqueP 在 GitHub 及另外四个大型数据集上的性能。具体而言，我们分别从每个数据集中抽取了 20%、40%、60% 和 80% 的边。可以观察到，随着数据集规模增大，BiclqueP 的运行时间增长仍在可控范围内，表明 BiclqueP 具有出色的可扩展性。

实验 6：内存。 表 4 展示了所有算法在最大的四个数据集以及包含最多极大双团的数据集 GitHub 上的内存使用情况。可以观察到，在所有数据集上，三种算法——MBAG、Biclique 和 BiclqueM—— 始终消耗最少的内存。这是因为这些算法无需存储 MBE 树。不过，尽管我们表现最佳的算法 BiclqueP 也会存储 MBE 树，其内存消耗仍在可控范围内。

实验7：案例研究。 我们在 DBLP 数据集上开展案例研究。该数据集包含数据库领域主要期刊（TODS、TOIS、TKDE、VLDBJ）和会议（SIGMOD、SIGKDD、ICDE、SIGIR、VLDB）发表的论文，其中两侧顶点分别表示论文和作者，而边（u, v）表示 v是 u的作者之一。我们将（α, β）-BCPC 与最大双团、k-Bitruss [28], 以及（α, β）-核 [18]进行比较。

给定一个二分图 $G = (U, V, E)$, (α, β)-核是 G $G_1 = U_1 V_1 E_1$ $\forall u \in U_1 \ |Nei_{G'} u| \geq$ 的一个极大子图，(, ,), α $\forall v \in V_1 \ |Nei_{G'} v| \geq \beta$ k G 且，(), , ()。-bitruss 是 的一个极大子图，$G_2 = (U_2, V_2, E_2)$，其中 E_2 中的每条边至少包含 k 个蝴蝶结构（蝴蝶结构是一个包含四个顶点的子图：$u\ w \in U_2\ v\ x \in$, , , , (,), (,), (,) V_2 $u\ v \in E_2$ $u\ x \in E_2$ $w\ v \in E_2$ $w\ x \in E_2$, (,)）。

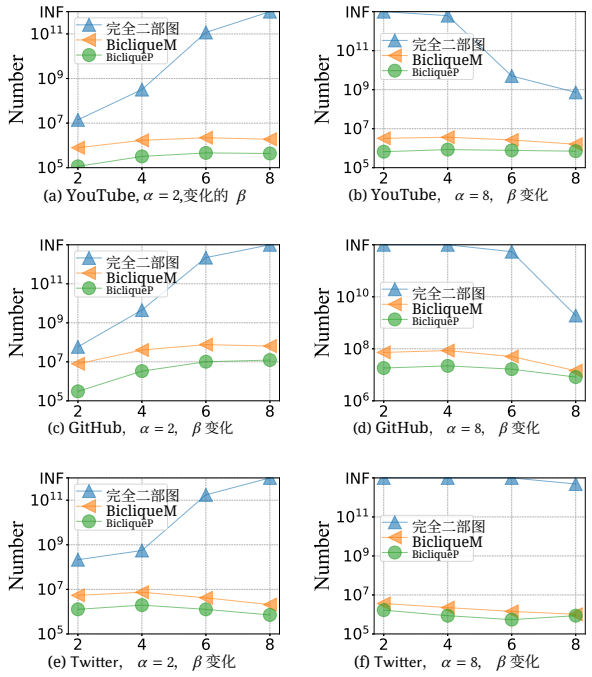


图 6：三种算法的 （α， β）-完全二部图枚举树中的枚举节点数量： Biclique （完全二部图）、BicliqueM （bicliquem）和BicliqueP （bicliquep）

在这个案例研究中， $\alpha = 4$， $\beta = 4$， $k = 12$。根据定义 4，(4，4)-BCPC 中最小的极大二分团要么是 (5， 4)-二分团，要么是 (4， 5)-二分团，并且这些极大二分团中的边至少存在于 $3 \times 4 = 12$ 个蝴蝶结构中，因此，我们将 $k = 12$ 设置为k-bitruss。

图 7 展示了所有模型中包含 *Jiawei Han* 的结果。其中，最大双团得到的结果最小。与 (4,4)-BCPC 相比，它少了一位作者 *Wei Fan*，而 *Jiawei Han*在谷歌学术上与这位作者有合作关系。尽管 12-蝴蝶支撑和 (4,4)-核包含 (4, 4)-BCPC 的结果，但它们还包含大量其他顶点，因此这两个模型的实用性远不及 (4,4)-BCPC。

7 相关工作

与我们工作最相关的是 BCPC 相关研究。最初， [14]提出了 BCPC 模型，并介绍了一种通过遍历极大二分团邻接图（MBAG）来检测 BCPC 的方法。随后， [27] 开发了一个在线网络工具包，实现了 [14]提出的方法。在社会网络分析领域，[11] 应用 BCPC 来识别用户和实体簇。在维基百科网络方面，[13] 利用 BCPC 探究编辑者与条目之间的关系，揭示了主题会自然地聚集编辑者。 [15] 将 BCPC 应用于企业网络，以分析主机与用户之间的交互模式，从而支持异常用户行为的检测。同时， [12, 33] 利用 BCPC 研究学生与在线学习资源之间的关系，深入探讨了学生与课程之间的时间演化模式。

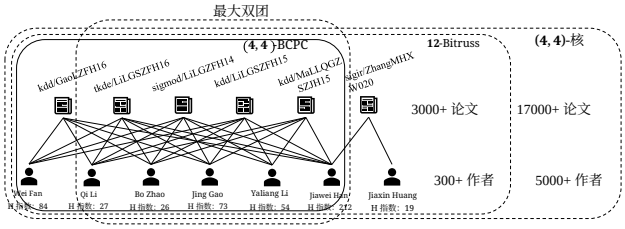


图 7： DBLP上的案例研究

此外， [6] 基于 BCPC 模型研究了个性化 BCPC 搜索问题。

我们的工作还与极大双团枚举和 (p, q)-二部团枚举相关。现有的极大双团枚举方法是支持 BCPC 模型实现的一项基础任务，主要基于枚举顶点集 [32] 的幂集这一思路。以此框架为基础， [1] 提出了枢轴剪枝策略，有效减少冗余搜索空间并提升效率。为进一步优化性能， [5] 提出了一种优化顶点枚举顺序的方法，从而提高计算速度。此外， [9]开发了一个适用于遗传凝聚子图的统一枚举框架。该框架在极大双团枚举问题上达到了最先进的性能，为相关任务提供了更高效且更具通用性的解决方案。关于 (p, q)-二部团枚举的代表性研究包括 [23, 30]。[23]仅将 (p, q)-二部团枚举作为更广泛图分析中的一个子过程，而 [30] 则专注于优化这一枚举任务。此外， [31] 等研究还讨论了 (p, q)-二部团的精确计数和近似计数。

此外，还有几种与我们工作相关的凝聚子图模型，例如 (α, β)-核和 k-bitruss。关于 (α, β)-核，代表性研究包括 [17–19], ，这些研究聚焦于该结构的高效计算和分布式实现。关于 k-bitruss， [16, 28, 34] 等研究致力于优化其分解与分析过程。对于最大双团， [21, 29]提出了高效的识别方法。除了特定的子图结构外，基于模块度的建模方法（例如 [2, 3]）和标签传播方法（例如 [20]）也得到了研究。

8 结论

在本研究中，我们解决了二分网络中的二部团渗流社区（BCPC）检测问题。针对现有 BCPC 检测方法因潜在规模巨大的极大二部团邻接图（MBAG）而导致的高时间复杂度，我们提出了两种创新方案。首先，基于部分 BCPC 的方案将极大二分团归组为不完整 BCPC，以减少冗余计算，从而减轻显式表示整个 MBAG 的负担。其次，我们提出了一种基于（α， β）-二部团枚举的新方法：通过枚举所有（α， β）-二部团，并连接共享此类二部团的极大二分团，来检测 BCPC。借助部分 BCPC，该方法显著缩减了（α， β）-二部团的枚举空间。实验结果表明，我们的方法性能出色，较现有方法提升近三个数量级。这些进展不仅提高了 BCPC 检测的效率，也为更具可扩展性的分析铺平了道路

应用于社交网络和推荐系统等多元领域的二分网络。

参考文献

[1] Aman Abidi、Rui Zhou、Lu Chen 和 Chengfei Liu。2020年。基于枢轴的极大双团枚举。在 *IJCAI*。3558–3564。[2] Rudy Arthur。2020年。二部网络的模块度与投影。《物理学A：统计力学及其应用》549（2020年），124341。[3] Michael J Barber。2007年。二部网络中的模块度与社区检测。《物理评论E：统计、非线性与软物质物理》76，6（2007年），066102。[4] Chiara Carusi 和 Giuseppe Bianchi。2020年。利用学者/期刊二分网络探讨跨学科性。《科学计量学》122，2（2020年），867–894。[5] Lu Chen、Chengfei Liu、Rui Zhou、Jiajie Xu 和 Jianxin Li。2022年。大型稀疏二部图的高效极大二分团枚举。《VLDB基金会论文集》15，8（2022年），1559–1571。[6] Zi Chen、Yiwei Zhao、Long Yuan、Xuemin Lin 和 Kai Wang。2023年。二分图上基于索引的双团渗流社区搜索。在 *2023*年 *IEEE*第*39*届国际数据工程会议（*ICDE*）。IEEE，2699–2712。[7] Chaitali Choudhary、Inder Singh 和 Manoj Kumar。2023年。推荐系统的社区检测算法：技术与指标。《计算》105，2（2023年），417–453。[8] Thomas H Cormen、Charles E Leiserson、Ronald L Rivest 和 Clifford Stein。2022年。《算法导论》。麻省理工学院出版社。[9] Qiangqiang Dai、Rong-Hua Li、Xiaowei Ye、Meihao Liao、Weipeng Zhang 和 Guoren Wang。2023年。二分图上的遗传凝聚子图枚举：基于枢轴的方法的力量。《ACM数据管理会刊》1，2（2023年），1–26。[10] Santo Fortunato。2010年。图中的社区检测。《物理学报告》486，3-5（2010年），75–174。[11] Tobias Hecking、Irene-Angelica Chounta 和 H Ulrich Hoppe。2015年。讨论论坛中的用户角色分析与主题的出现。在 *2015*年第二届欧洲网络智能会议。IEEE，114–121。[12] Tobias Hecking、Sabrina Ziebarth 和 H Ulrich Hoppe。2014年。混合式学习课程中的动态资源访问模式分析。在 第四届国际学习分析与知识会议论文集。173–182。[13] Rut J esus、Martin Schwartz 和 Sune Lehmann。2009年。维基百科条目与作者的二部网络：一种中观层面方法。在 第*5*届维基与开放协作国际研讨会论文集。1–10。[14] Sune Lehmann、Martin Schwartz 和 Lars Kai Hansen。2008年。双团社区。《物理评论E：统计、非线性与软物质物理》78，1（2008年），016108。[15] Qi Liao、Aaron Striegel 和 Nitesh Chawla。2010年。企业网络安全与管理中的图动态和相似性可视化。在 第七届网络安全可视化国际研讨会论文集。34–45。[16] Fengnian Lin、Boyu Ruan、Junhao Gan 和 Lei Li。2025年。不进行蝶形枚举的高效Bitruss分解。在 第*31*届 *ACM SIGKDD*知识发现与数据挖掘会议论文集第二卷。1718–1728。[17] Boge Liu、Long Yuan、Xuemin Lin、Lu Qin、Wenjie Zhang 和 Jingren Zhou。2019年。高效的（α, β）-核计算：一种基于索引的方法。在 万维网会议。1130–1141。[18] Boge Liu、Long Yuan、Xuemin Lin、Lu Qin、Wenjie Zhang 和 Jingren Zhou。2020年。二分图中的高效（α, β）-核计算。《VLDB期刊》29，5（2020年），1075–1099。[19] Qing Liu、Xuankun Liao、Xin Huang、Jianliang Xu 和 Yunjun Gao。2023年。二分图上的分布式（α, β）-核分解。在 *2023*年 *IEEE*第*39*届国际数据工程会议（*ICDE*）。IEEE，909–921。[20] Xin Liu 和 Tsuyoshi Murata。2010年。用于优化二部网络中二部模块度的高效算法。《先进计算智能与智能信息学期刊》14，4（2010年），408–415。[21] Bingqing Lyu、Lu Qin、Xuemin Lin、Ying Zhang、Zhengping Qian 和 Jingren Zhou。2020年。十亿规模的最大二部团搜索。《VLDB基金会论文集》（2020年）。[22] Cristina Maier 和 Dan Simovici。2022年。二分图与推荐系统。《信息技术进展期刊》-印刷中（2022年）。[23] Michael Mitzenmacher、Ja kub Pachocki、Richard Peng、Charalampos Tsourakakis 和 Shen Chen Xu。2015年。通过采样在大规模网络中实现可扩展的大型近似团检测。在 第*21*届 *ACM SIGKDD*知识发现与数据挖掘国际会议论文集。815–824。[24] Erich Prisner。2000年。图中的完全二部图 I：关于其数量的界。《组合数学》20，1（2000年），109–117。[25] Sara Rahiminejad、Mano R Maurya 和 Shankar Subramaniam。2019年。生物网络中社区检测算法的拓扑与功能比较。《BMC生物信息学》20，1（2019年），212。[26] Karsten Steinhaeuser 和 Nitesh V Chawla。2008年。大型真实社交网络中的社区检测。在 社会计算、行为建模与预测。施普林格，168–175。[27] Bei Wang、Jinyu Chen 和 Shihua Zhang。2018年。BMTK：用于确定生物二部网络中模块的工具包。《定量生物学》6（2018年），186–192。[28] Kai Wang、Xuemin Lin、Lu Qin、Wenjie Zhang 和 Ying Zhang。2020年。大规模二部图的高效Bitruss分解。在 *2020*年 *IEEE*第*36*届国际数据工程会议（*ICDE*）。IEEE，661–672。[29] Kai Wang、Wenjie Zhang、Xuemin Lin、Lu Qin 和 Alexander Zhou。2022年。高效的个性化最大二部团搜索。在 *2022*年 *IEEE*第*38*届国际数据工程会议（*ICDE*）。IEEE，498–511。[30] Jianye Yang、Yun Peng、Dian Ouyang、Wenjie Zhang、Xuemin Lin 和 Xiang Zhao。2023年。大型稀疏二部图的（p, q）-二部团计数与枚举。《VLDB期刊》32，5（2023年），1137–1161。[31] Xiaowei Ye、Rong-Hua Li、Qiangqiang Dai、Hongchao Qin 和 Guoren Wang。2023年。大型二部图中的高效二部团计数。《ACM数据管理会刊》1，1（2023年），1–26。[32] Yun Zhang、Charles A Phillips、Gary L Rogers、Erich J Baker、Elissa J Chesler 和 Michael A Langston。2014年。关于在二分图中寻找二部团：一种新算法及其在多样生物数据类型整合中的应用。《BMC生物信息学》15，1（2014年），110。[33] Sabrina Ziebarth、German Neubaum、Elias Kyewski、Nicole Kramer、H Ulrich Hoppe、Tobias Hecking 和 Sabrina Eimler。2015年。在线课程中的资源使用：分析学习者的主动与被动参与模式。国际学习科学学会公司。[ISLS]。[34] Zhaonian Zou。2016年。二分图的Bitruss分解。在 数据库系统高级应用国际会议。施普林格，218–233。