Finding Large Diverse Communities on Networks: The Edge Maximum k*-Partite Clique

Alexander Zhou
Hong Kong University of Science and Technology
Hong Kong
atzhou@cse.ust.hk

Yue Wang
Shenzhen Institute of Computing Sciences
Shenzhen University
Shenzhen
yuewang@sics.ac.cn

Lei Chen
Hong Kong University of Science and Technology
Hong Kong
leichen@cse.ust.hk

ABSTRACT

In this work we examine the problem of finding large, diverse communities on graphs where the users are separated into distinct groups. More specifically, this work considers diversity to be the inclusion of users from multiple groups as opposed to homogeneous communities in which the majority of users are from one group. We design and propose the k^*-Partite Clique (and the *edge-maximum* k^*-Partite Clique Problem) which modifies the k-Partite Clique structure as a means to capture these large, diverse communities in a way that does not currently exist. We then design a non-trivial baseline enumeration algorithm, which is further improved via heuristics to significantly reduce the running time whilst avoiding excessive memory requirements. Moreover, we propose a core as well as a truss structure for the k-Partite environment aimed at finding the *edge-maximum* k^*-Partite Clique structure on the network. Comprehensive experiments on real-world datasets verify both the effectiveness of the k^*-Partite Clique at finding diverse communities as well as the efficiency of the proposed heuristics to our algorithms compared to reasonable baselines.

PVLDB Reference Format:
Alexander Zhou, Yue Wang, Lei Chen. Finding Large Diverse Communities on Networks: The Edge Maximum k*-Partite Clique. *PVLDB*, 13(11): 2576-2589, 2020.
DOI: https://doi.org/10.14778/3407790.3407846

1. INTRODUCTION

1.1 Motivation

On a network, nodes may be partitioned into groups based on a common attribute shared among them. Common examples are ethnicities, religions or physical locations on human-based networks and tags/categories on networks for items commonly bought together in a retail environment. Networks are also the home to communities, which are collections of nodes that are closely connected to each other. Traditional communities may contain users from any group but they may just as equally contain no users from the majority of groups. By being able to find and identify communities that are diverse, that is they contain a balance of nodes from multiple groups, over ones that cannot allow operators of these networks to gain significant insight and benefits to their system.

Before we define a diverse community, we first acknowledge that the term 'diverse' (as well as 'representation') holds multiple meanings in the eyes of different people. The notion of 'diverse' that we utilise in this paper is not one that aims to model communities with the same underlying distribution as the population the users come from. Instead, we use the 'Value-in-Diversity' Hypothesis [11], which favours the idea diversity promotes the most 'value' when two individuals of differing backgrounds interact.

Maintaining a diverse network socially has commonly been attributed to providing positive effects towards members of the network. Noted benefits of enjoying such a diverse network range from improved physical health (such as reduced mortality rate in the elderly [3]) to increased success and productivity in the workforce [12, 31]. Empirical studies and surveys have observed that a diverse executive group is correlated to increased financial performance [20] and that increased ethnic identity in scientific papers can be connected to increased citations and publications in higher impact journals [14].

Additionally, individuals with similar backgrounds and opinions can congregate and form 'echo chambers', which reinforces these popular (yet not necessarily true) opinions via confirmation bias [8, 37]. This promotes the spread of 'fake news' [4] in their community, thus exacerbating the issue. Looking at diverse communities from another perspective, operators of social media platforms or employers of a workforce may want to be able to identify a collection of well connected individuals that is 'diverse', thus allowing for future actions to maintain and grow these communities.

Despite the power of diversity being a well-established concept, work regarding finding and identifying diverse communities in a network is still a relatively unexplored field. For the most part, previous works focus on issues in which users have binary opinions [2, 10, 15, 24], with the common motivation being political affiliation in the United States (Democrat vs. Republican). However, in reality, most diversity metrics cannot be simply limited to binary options. Even in the U.S. political example, multiple different candidates from the same party in the primary race will have

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407846

distinct support bases with different values. Additionally, countries such as Germany consist of multiple political parties that cannot be modelled through polarisation.

Importantly, the concept of a diverse community is a fluid one, given that there is no concrete, universally accepted definition of the term. However, in line with the 'Value-in-Diversity' hypothesis, any diverse community relative to the network it exists upon should satisfy two core requirements:

1. The community should encourage the inclusion of more groups;
2. The community should encourage more balanced participation from each of its groups.

In this paper we address the problem of finding diverse communities from any number groups by proposing a structure to model such a community in a network. Moreover, we are interested in finding large diverse communities as these groups of users tend to be of more interest to operators of networks. To the best of our knowledge, no other structure designed to model diverse communities with more than two groups exists.

Our structure, the k^*-Partite Clique (k^*-PC) modifies the established k-Partite Clique (k-PC) structure [33] with a key distinction. The k-PC requires that each node from one group contained within the structure must be connected to each other node from another group inside the structure. A particularly popular case of the k-PC is the biclique, which operates on a bipartite graph ($k = 2$).

In the traditional k-PC on a k-Partite Graph problem, at least one node must be present from each of k groups found in the graph. However, when the number of groups increases, it becomes increasingly unrealistic to expect a large structure to exist which contains at least a member from each group. One real-life example is using nationality as the diversity metric. It is unreasonable to define a diverse community as one that must include a member of each nationality on the Earth.

1.2 Our Contribution

To accommodate for this issue, we define the k^*-Partite Clique to allow for tightly connected communities from at least k groups on a network which contains n total groups ($k \leq n$), thus turning k into a lower bound requirement for the number of groups to be considered as desirably diverse.

An evident drawback to the k^*-PC is that for certain, smaller k values it may become overly broad, with enumeration allowing for outputs that may be considered not diverse. As an example, if we have a network with twenty groups but the structure we output contains only two (albeit the outputted community contains the most users out of any community) then it is difficult to argue that the structure is diverse given the context. We are thus incentivised to define our output structure to include more groups in a way that obeys the two fundamental requirements mentioned previously.

To accommodate for this, the measurement we use for quantifying a larger community is the number of cross-group edges as opposed to the number of users inside of it. That is, our top output is the one that is *edge-maximum*. In other words, we wish to find the maximal k^*-PC with more cross-group edges than any other k^*-PC on the given graph. Given that the number of edges can be derived by the product of the number of nodes in each group of the k-PC, not only does this metric encourage a structure with more groups participating but also a more balanced number of nodes from each group. An illustrated example is in Figure 1 where C2 (8 users from 7 groups) contains less users than C1 (10 users from 2 groups) but ultimately has more cross-group edges (27 vs 25), thus making C2 a more diverse community than C1 under our definition.

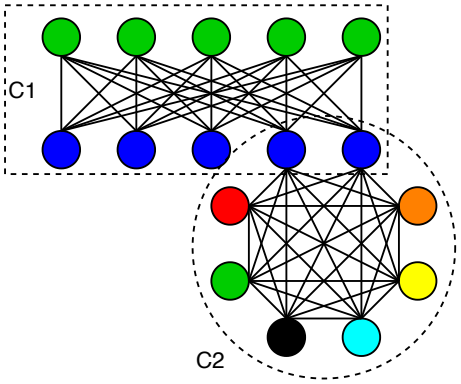

Figure 1: A network divided into different groups separated by colour. Despite C1 containing more users than C2 (C1: 10 vs C2: 8), C2 is favoured in our model as a diverse community given that it contains more groups (C1: 2 vs C2: 7) which is reflected in the cross-group connections (C1: 25 vs C2: 27).

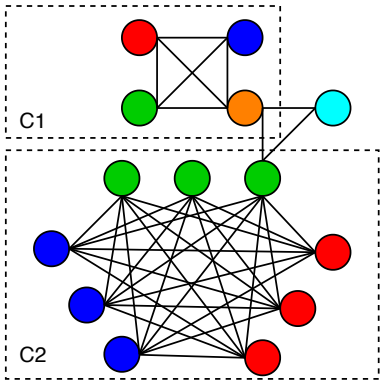

Figure 2: A network containing 5 groups separated by colour, yet no 5-PC exists upon it. C1 and C2 are two 3^*-PC, with only C1 being a 4^*-PC. Despite C1 containing more groups than C2, C2 is favoured in our model given that it contains significantly more cross-group edges (C1: 6 vs C2: 27).

Given that we are interested in finding large, diverse communities relative to the input network, the *edge-maximum* requirement assists in that regard. Figure 2 provides a visual example of when the more preferable community is not the one with the most groups. On this graph, C2 is selected over C1 despite having fewer groups (C1: 4 vs C2: 3) due to having significantly more cross-group edges (C1: 6 vs C2: 27).

In this work, we wish to find the *edge-maximum* k^*-PC over the entire network (containing n groups), (for a given

$k \leq n$) which we then determine to be a large diverse community on our network.

Given the properties of our structure, we only require the assumption that each user in the network has a known group for the desired diversity metric. This is the only hurdle for our model, with the level of difficulty to model being dependent on the desired diversity metric itself. This is unlike other existing models that require the ability to gauge the susceptibility of each user to the opinions of others [24].

As a baseline for the *edge-maximum* k^*-PC problem, we design a k^*-PC enumeration algorithm which utilises merging of two maximal $(k-1)$-PCs to form a k-PC. Additionally, we discuss added heuristics to increase the efficiency of the algorithm. After which, we discuss an improved *edge-maximum* specific algorithm which introduces a k-Partite specific Core and Truss structure able to prune useless nodes in polynomial time, thus heavily narrowing the search space.

Our contribution in this paper can be broken down into the following points:

1. We propose the k^*-Partite Clique structure as a means to model diverse communities on a network;

2. We formally introduce *edge-maximum* k^*-Partite Clique problem (which is NP-Hard and NP-Hard to approximate) and introduce a baseline which consists of a k^*-PC enumeration algorithm. We then propose heuristic-based improvements that reduce the runtime of the enumeration algorithm, including introducing a Core and Truss structure for a k-Partite Graph;

3. We demonstrate the effectiveness of our structure by comparing it to given baseline structures. We experimentally show the efficiency of our enumeration algorithm as well as the *edge-maximum* specific algorithms compared to a baseline as well as the improved efficiency of our heuristic improvements.

In the following sections of this paper, we first discuss related works to our topic (Section 2) before formally defining our problem (Section 3). Then, we propose an enumeration algorithm (with heuristic improvements) which serves as the baseline (Section 4). After that, we introduce our *edge-maximum* specific heuristics (Section 5). The effectiveness and efficiency of our structure and algorithms are examined afterwards (Section 6). Finally we discuss future work (Section 7) and conclude the paper (Section 8).

2. RELATED WORKS

Community Detection: The area of community detection is one of great interest in the field of data mining, with a particular focus on social networks. Given the fluid definition of the term 'community', a significant number of structures have been used to best work towards their own understanding of the term. These include very basic subgraph structures such as the k-core [32], k-truss [9, 39] and clique [40] as well as more sophisticated structures and methodologies.

Such advanced structures include using tie-strength [44, 47], leveraging observed and manifested attributes [42] or using both similarity and engagement [45]. Other techniques focus on achieving specific benchmarks such as GN [17] or LFR [21] as their definition of community. Whilst these methods are suited towards their own specific branch of community detection, they are not diversity-aware.

Importantly, we note that there exists another definition of diversity in community detection which aims to output structures that are not encompassed by other outputs (i.e. structural diversity). Examples of this include Diversified Top-K Clique search [43] as well as Top-k Structural Diversity Search [19]. However, these models do not address label diversity.

Biclique and k-Partite Clique: The biclique as a structure has been the focus of a significant amount of work for a very lengthy period of time. It is not only a noted and utilised structure in data mining [26] but also well-documented in the fields of cryptography [7], textile engineering [18] and bioinformatics [38]. Maximal biclique enumeration as a problem has also been studied over time, with algorithms in place such as iMBEA [46] and an alternative method using MapReduce [26].

A problem which is encompassed by ours, the *edge-maximum* biclique problem over a graph has been established to be NP-Complete [28] as well as hard to approximate [13]. Previous work that tackle this problem tend to employ heuristics in union with traditional enumeration techniques, though an algorithm that focuses on Rank-one Matrix Factorisation does exist [16].

On the other hand, k-Partite Cliques (for $k \geq 3$) have had significantly less focus as an area of research despite still being of interest in multiple fields. In terms of k-PC enumeration, there are two methods we need to consider. The first method involves creating a set intersection graph on the original graph and then perform maximal biclique enumeration [30]. The second method works by finding a maximal $(k-1)$-PC and then adding an additional group to the previously found structure [22]. Importantly, these methods only find structures with a fixed k value and they are hence only able to model a specific number of groups with no leeway. A simplified case of the k -PC, the k-clique (where it is required that only one node from each of the k groups is a part of the structure) has had enumeration techniques proposed [18, 25], though their fixed size makes them unattractive from the perspective of large communities.

Diversity in Social Networks: For the most part, previous works that have examined the concept of diversity have worked from the perspective of detecting [2, 10] and reducing polarization [15] or maximising diversity [24]. In particular, they focus on issues in which users have binary opinions with the common motivation being political affiliation in the United States (Democrat vs Republican). These works are not designed with networks containing more than two groups in mind, which make up a significant portion of real world systems. Additionally, they may require difficult assumptions such as knowing the influence that any given user has on other users or the susceptibility of each individual to the opposing viewpoint. Furthermore, they are not aimed at community detection.

3. PROBLEM DEFINITION

Let us have an undirected, unweighted graph $G = (U, E)$ with a set of users (nodes) U and edges E (which represents a connection between two users). Now suppose we have a set of n non-overlapping and distinct groups (or 'labels') $\eth = \{g_1, g_2, \ldots, g_n\}$ with each node being a member of exactly

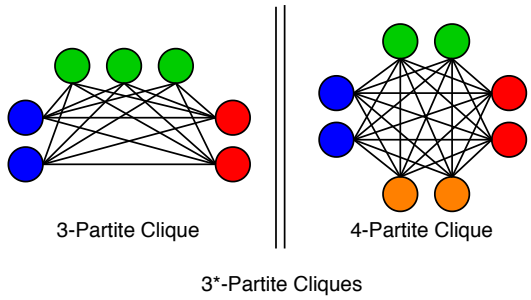

Figure 3: The subgraph on the left is a 3-PC whilst the one on the right is a 4-PC. They are both valid 3*-PCs.

one group. We denote the set of all nodes in U that belong to group g_i as the set U_{g_i} $(U = \bigcup_{i=1}^{n} U_{g_i}, U_{g_i} \cap U_{g_j} = \emptyset, \forall i \neq j)$.

Before we introduce the k^*-Partite Clique, we lay the foundations with the established k-Partite Clique [33].

Definition 1. k-Partite Clique (k-PC): Suppose we have any set of groups $\eth' \subseteq \eth, |\eth'| = k$. A k-Partite Clique C^k is a subgraph on G such that it contains a set of nodes $U'_{g'_i} \subseteq U_{g'_i}$, $U'_{g'_i} \neq \emptyset$ for all $g'_i \in \eth'$ in which there exists an edge $e_{\alpha,\beta}$ for all $u_\alpha \in U'_{g'_i}$ $u_\beta \in U'_{g'_j}$, $g'_i, g'_j \in \eth'$, $i \neq j$. We may also use the notation $C^k_{\eth'}$ to denote a k-Partite Clique with groups $\eth'$.

The problem of determining if a k-PC exists on G for $k \geq 3$ has been determined to be NP-Complete [30]. Furthermore, a k-Partite Clique is not guaranteed to exist on all graphs.

Given the formal definition of the k-PC, we then relax the requirement of needing exactly k groups in the structure. With this, we derive the definition of the k^*-Partite Clique.

Definition 2. k^*-Partite Clique (k^*-PC): For a value k and graph G, if there exists a value k' $(k \leq k' \leq |\eth| = n)$ such that a k'-PC exists on G, then we say G contains a k^*-Partite Clique. We may also use the notation $C^{k*}_{\eth^*}$ to denote a k^*-Partite Clique with nodes from groups in $\eth^* \subseteq \eth$ where $|\eth^*| \geq k'$.

If the value of $k' = k = n$, then the set of all k^*-PC is equivalent to the set of all k-PC. We are interested in larger values of k' as (demonstrated later in the paper) the subgraphs induced by these values are more likely to produce a large, diverse community. Figure 3 illustrates an example where two structures, a 3-PC on the left and a 4-PC on the right, are both valid 3*-PC. However, only the structure on the right is a valid 4*-PC and neither are a 5*-PC.

In essence, the value k is the lower bound of the number of groups required (out of n total) for a community to be considered diverse.

Definition 3. Maximal k^*-Partite Clique: A k^*-PC C (with nodes U) is maximal if there exists no other k^*-PC C' (with nodes U') such that $U \subset U'$.

The set of all maximal k-PCs are encompassed by the set of all maximal k^*-PCs, giving our structural definition more flexibility to find larger groups that would reasonably still be

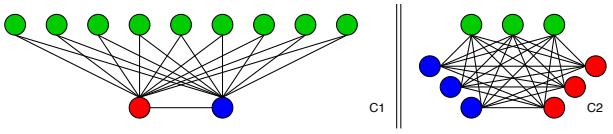

Figure 4: C1 contains 11 nodes and 19 cross group edges as opposed to C2 with 9 nodes but 27 cross group edges

Table 1: Key Notation and Definitions

Notation	Definition
G	Undirected and Unweighted Graph
g_i	Group i
$\eth$	Set of groups
$C^k_{\eth}$	k-Partite Clique containing groups $\eth$
$C^{k*}_{\eth}$	k^*-Partite Clique containing groups $\eth$
C^{k*}_{emax}	*edge-maximum* k^*-Partite Clique
$E(C^{k*})$	# cross-group edges on C^{k*}

considered diverse despite not consisting of all groups in our network. We define a cross-group edge as an edge between nodes of different groups.

Definition 4. Edge-Maximum k^*-Partite Clique: The *edge-maximum* k^*-Partite Clique C^{k*}_{emax} is the k^*-PC with the greatest number of cross-group edges in G.

By definition, the *edge-maximum* k^*-PC on G is a maximal k^*-PC. From mathematical intuition, the number of cross-group edges greatly improves with the introduction of more groups which helps to push our outputted structure towards the inclusion of more groups. Additionally, the focus on the number of edges also pushes us towards structures with a greater balance in the number of each participants from each group. As an example in Figure 4, suppose we have two 3*-PCs $C1$ with $|U_{g_1}| = 9, |U_{g_2}| = 1, |U_{g_3}| = 1$ and $C2$ with $|U_{g_1}| = 3, |U_{g_2}| = 3, |U_{g_3}| = 3$. Despite $C1$ consisting of more vertices, $E(C2) = 27 > E(C1) = 19$ and thus $C^{3*}_{emax} = C2$.

Table 1 provides a summary of key notations that may be used in this paper. Here, we formally define the problem we wish to tackle in this work:

Definition 5. The Edge-Maximum k^*-Partite Clique Problem: Given a graph $G = (U, E)$ with n groups $\eth = \{g_1, \ldots, g_n\}$, in which the nodes U are partitioned into these groups $(U = U_{g_1} \cup \cdots \cup U_{g_n})$, as well as an input value $k \leq n$ we wish to find the *edge-maximum* k^*-Partite Clique on G.

THEOREM 1. *The edge-maximum k^*-Partite Clique problem is NP-Hard and also NP-Hard to approximate.*

PROOF. Given that the *edge-maximum* k-Partite Clique problem on a k-Partite graph, which has been established to not only be NP-Hard but also NP-Hard to approximate [30], is a special case of the *edge-maximum* k^*-Partite Clique problem, it is trivial to determine that our problem is also NP-Hard an NP-Hard to approximate. □

Our problem and structure do not require intra-group connections inherently, however in the event in which they are

of interest to form the community a constraint can be added which requires each user from a group to be connected to each other user in the same group in the final community. Modifications to our k^*-PC algorithms to suit this problem are noted as necessary.

4. ENUMERATION ALGORITHM

The trivial solution in finding the *edge-maximum* k^*-PC would be to enumerate all maximal k^*-PCs on G and then scan them for the structure with the most edges.

Despite the k-PC being an identified structure for a considerable period of time, methodologies for maximal k-PC enumeration (when $k > 2$) are still costly. As k^*-PC enumeration is evidently expensive, it is imperative to find as many cost saving measures as possible.

An obvious, trivial baseline for enumeration would be finding each maximal m-PC for all values $m = k, \ldots, n$ independently (e.g. for a 2^*-PC find all 2-PCs, 3-PCs, ..., n-PCs) and combining the outputs to satisfy our requirement. However, this technique is not only prohibitively slow but also holds redundancy when moving from lower values of k to higher ones.

We propose a method that sequentially enumerates all k^*-PCs in a more efficient manner.

LEMMA 1. *A maximal $(k-1)$-Partite Clique C^{k-1} with groups $\eth' = \{g_1, \ldots, g_{k-1}\}$ and another group g_k contains all required nodes to find a maximal k-PC C^k over the entire graph. As a result, a set of all maximal k-PCs from $\eth'$ and all nodes from g_k are all that is required to produce the set of all maximal k-PCs from $\eth' \cup g_k$.*

PROOF. Suppose we have a graph G with k groups, a maximal $(k-1)$-PC C^{k-1}, and the final group g_k not in C^{k-1}. In this situation, there are four possible outcomes for any node u in C^{k-1}.

1. u is not connected to any nodes in g_k and thus is not in any maximal k-PC.
2. u exists in a k-PC with only nodes in C^{k-1} and g_k.
3. u exists in a k-PC with nodes in $g_1, \ldots, g_{k-1}$, but not in C^{k-1}, and g_k. In this situation, these nodes from $\eth'$ will be all be found in another maximal $(k-1)$-PC.
4. Any combination of Item 2 and 3 (each item may occur multiple times).

Importantly, it is impossible to have a k-PC that does not only contain nodes from one (and only one) maximal $(k-1)$-PC and g_k. This is fairly trivial, given that all nodes from $\eth'$ in a k-PC $C^k_{\eth}$ form a $(k-1)$-PC $C^{k-1}_{\eth'}$ which must either be maximal or be part of a maximal structure over G. □

From Lemma 1, all maximal k-PCs on G can be found by knowing all maximal $(k-1)$-PCs on G, which can be further stated in the following theorem:

THEOREM 2. *A maximal k-PC (where $k > 3$) must exist in the combined nodes of two (and only two) maximal $(k-1)$-PCs given that they share at least one common node from exactly $(k-2)$ groups.*

PROOF. Suppose we have two maximal $(k-1)$-PCs $C1$ and $C2$ with common nodes in $(k-2)$ groups with the other group in each clique being different (denoted by g_1, g_2 for $C1$ and $C2$ respectively). Via Lemma 1 we may see that $C1$ and another group is all that is required to (potentially) form a maximal k-PC on G. By knowing $C1$ and $C2$, this is the equivalent of having only the common nodes of $C1$ and $C2$ and the nodes from g_1 in $C1$ and nodes from g_2 in $C2$ to consider.

By finding all maximal bicliques between nodes from $C1 \in g_1$ and $C2 \in g_2$, if we combine each of these bicliques with the common nodes of both structures $C1 \cap C2$ (which we know to be fully connected to all nodes in $C1$ and $C2$), the resulting combination of nodes forms a maximal k-PC on the nodes in $C1 \cup C2$, which may be maximal on G. Furthermore, all maximal k-PCs on G exist in the combinations of two maximal $(k-1)$-PCs from all such structures on G. □

Theorem 2 guides the foundation of our enumeration algorithm as we can use previously found results to construct future results.

Algorithm 1: MaximalkPCEnumeration

```
   Input  : G: Input Graph
            k: k value
            C^{k-1}: All maximal (k-1)-PCs
   Output: C^k: All k-PCs
 1 C_k ← ∅;
 2 foreach C1 and C2 in C^{k-1} do
 3   if C1 and C2 share common nodes in k-2 groups then
 4     U_common ← Common nodes of C1 and C2;
 5     U_C1 ← Nodes in uncommon group of C1;
 6     U_C2 ← Nodes in uncommon group of C2;
 7     C^2 ← MaximalBicliqueEnumeration(U_C1, U_C2);
 8     foreach Maximal Biclique C_i^2 in C^2 do
 9       C^k_max ← C_i^2 ∪ U_common;
10       if C^k_max is maximal AND C^k_max ∉ C^k then
11         C^k.insert(C^k_max);
12       end
13     end
14   end
15 end
```

Our maximal k-PC Enumeration algorithm for a fixed k value (Algorithm 1) takes a Graph G, the k value and all $(k-1)$-PCs ($\mathbb{C}^{k-1}$) on G as an input. For any two maximal $(k-1)$-PCs $C1$ and $C2$ in $\mathbb{C}^{k-1}$, we check if they share nodes in exactly $(k-2)$ groups (the final group in each of the two structures must be different) (Line 3). We then determine the common nodes between $C1$ and $C2$ (denoted as U_{common}) (Line 4) as well as the nodes from the off-group in $C1$ and $C2$ (denoted by U_{C1} and U_{C2} respectively) (Lines 5-6). We then perform a maximal biclique enumeration technique (with the current benchmark being iMBEA [46]) on U_{C1} and U_{C2}(Line 7). For each discovered maximal biclique, we combine them with U_{common} to create a maximal k-PC over $C1 \cup C2$, which we then add to the output given it is maximal over G (an easy check) and not already in the output (Line 8-12).

The time cost of Algorithm 1 is:

$$O\left(\sum_{C1,C2\in\mathbb{C}^{k-1}} |U| + \Gamma(C1,C2)\right),$$

as we can check if $C1$ and $C2$ shares common nodes in $(k-2)$ groups in $O(|U|)$ time. $\Gamma(C1,C2)$ is the cost of performing maximal biclique enumeration for the nodes in the two off groups between $C1$ and $C2$ if the common node check is successful. If it is not, $\Gamma(C1,C2)=0$.

Algorithm 2: Maximalk*PCEnumeration

```
Input  : G: Input Graph
         k: k value
Output: ℂ^{k*}: All k*-PCs
1 ℂ^{k*} ← ∅;
2 ℂ^k ← MaximalKPCEnum(G);
3 ℂ^{k*} ← ℂ^{k*} ∪ ℂ_k;
4 for i = k+1, ..., n do
5 |   ℂ^i ← kPartiteCliqueEnumeration(G, i, ℂ^{i-1});
6 |   ℂ^{k*}.insert(ℂ^i);
7 end
```

In essence, our baseline maximal k^*-PC enumeration algorithm (Algorithm 2) is simple. We first find all maximal k-PCs using any state-of-the-art maximal k-PC enumeration algorithm without all maximal $(k-1)$-PCs as input. Then for each other value between k and n, we call Algorithm 1 method iteratively (Lines 4-7).

Algorithm 2 takes $O(\Psi+\sum_{i=k+1}^{n}\Xi^i)$ time to run where Ψ is the cost of maximal k-PC enumeration without without all maximal $(k-1)$-PCs as input and Ξ^i is the cost of i-PC enumeration using Algorithm 1, described previously.

Our memory requirements for our technique is minimal as all objects that are accessed are either pre-loaded as part of the input or are a member of the output. However, on the time cost side, each call of maximal biclique enumeration (Line 7 in Algorithm 1) is evidently expensive as the problem is NP-Hard, though the search space in general on real-world graphs is significantly smaller than the worst case of all nodes from $g_1, g_2 \in \eth$.

Before we continue, let us form some notation to better separate the k-PCs in $\mathbb{C}^k$.

Definition 6. k-Group Set (k-GS): Let us denote the set of all k-PCs consisting of groups $\eth' \subseteq \eth$, $|\eth'| = k$ as $S^k_{\eth'}$. We will refer to this from hereon as a k-Group Set.

In a naive implementation of our maximal k^*-PC enumeration algorithm (Algorithm 2), we will call the maximum biclique enumeration:

$$O\left(\sum_{i=k+1}^{n}\sum_{S1,S2\in\mathbb{S}^i} \mathbb{I}\cdot|S1|\cdot|S2|\right)$$

times. Where $\mathbb{S}^i$ is the set of all i-GSs and $\mathbb{I}=1$ when $S1$ and $S2$ share exactly $(k-2)$ groups and $\mathbb{I}=0$ otherwise.

In the event we wish to include an intra-group connection requirement, we use our algorithms as described above with the following key change. When finding the initial bicliques which will be used as the foundation to form >2-PCs, we first find maximal intra-group cliques for each group. Then, instead of finding the initial bicliques normally for groups g_i and g_j, we instead find the maximal bicliques between nodes from one intra-group clique for g_i and another intra-group clique for g_j. We repeat this process for the combination of all intra-group cliques between the two groups and then check for redundant structures. Afterwards, we continue the algorithms as described.

4.1 Improvements to (k-1)-Partite Clique Combining

To avoid these unnecessary operations when combining $(k-1)$-PCs to form the set of all k-PCs, we propose multiple heuristics. These proposed heuristics reduce both the number of common node checks, via the use of Bloom Filters, as well as reduce the number of maximal biclique enumeration, via systematic merging of $(k-1)$-GSs, calls up to a factor of $\frac{1}{k}$ the number required in the baseline method.

4.1.1 Common Node Check

When determining whether two $(k-1)$-PCs ($C1$, $C2$) from all such structures $\mathbb{C}^{k-1}$ share common nodes from at least $(k-2)$ groups, the trivial method would be to individually examine all possible combinations of two structures from $\mathbb{C}^{k-1}$. This would imply $O(|\mathbb{C}^{k-1}|^2)$ total comparisons of $(k-1)$-PCs.

Evidently, we only need to compare structures which share $(k-2)$ groups in the first place, information which we can easily maintain at minimal cost throughout the enumeration process by grouping together $(k-1)$-PCs into appropriate k-GSs based on the groups in which they are a part of. With this, we can use a merging approach similar to the Apriori approach for Frequent Itemset Generation [1].

We can observe that the number of k-GSs on a graph G with n total groups can be derived via the basic combination formula, in which order does not matter. That is, there are $\binom{n}{k}$ k-GSs on G.

LEMMA 2. *When combining all $(k-1)$-PCs into all k-PCs, comparisons on $\binom{n}{k}$ combinations of two $(k-1)$-GSs are required.*

PROOF. Naively, when combining all $(k-1)$-PCs into all k-PCs, we would need to perform comparisons on $\binom{n}{k-1}\cdot\left(\binom{n}{k-1}-1\right)$ combinations of $(k-1)$-GSs to produce all maximal k-PCs. However, this is not necessary given that there are $\binom{k}{k-1}$) combinations of two $(k-1)$-GSs which produce the same k-GS.

For example, suppose we have $S^2_{g1,g2}$, $S^2_{g1,g3}$, $S^2_{g2,g3}$ and we wish to form $S^3_{g1,g2,g3}$. Despite there being three possible combinations of the 2-GSs which will sufficiently create the same 3-GS, we require only one to produce it perfectly.

In fact, if we do not conduct comparisons on any unnecessary combinations of two $(k-1)$-GSs when attempting to form all k-GSs, we only need to perform $\binom{n}{k}$ combinations of two $(k-1)$-GSs. Rather evidently, we know that $\binom{n}{k} < \binom{n}{k-1}\cdot\left(\binom{n}{k-1}-1\right)$ for all values of $n, k > 0$. □

There are of course also costs associated with determining if two $(k-1)$-PCs share the sufficient common nodes in order to even possibly create a k-PC. When finding common elements between two sets, on larger datasets even this process can be costly if not treated appropriately.

One notable data structure we can leverage is the Bloom Filter [6] or, if memory costs are not a concern, perfect "error-free" hashing methods. We will henceforth operate under the assumption our Bloom Filter has 0% chance of a false positive the description of our technique though in actual implementation a further check is necessary past the initial Bloom Filter check.

Algorithm 3: CommonNodeCheck

```
   Input  : C1, C2: Maximal (k − 1)-PCs which share
            (k − 2) groups
            B: Set of Bloom Filters for each group in
   C1
   Output: out: Boolean Output
 1 foreach Group g_i which is in both C1 and C2 do
 2 |   contain ← false;
 3 |   foreach Node u ∈ g_i in C2 do
 4 |   |   if B_i.contains(u) then
 5 |   |   |   contains ← true;
 6 |   |   end
 7 |   end
 8 |   if !contains then
 9 |   |   return out = false;
10 |   end
11 end
12 return out = true;
```

Algorithm 3 highlights a straightforward method to determine if two $(k-1)$-PCs $C1$ and $C2$, sharing $(k-2)$ groups, share nodes in those groups. Suppose $C1$ consists of groups $\{g_1, \ldots, g_{k-1}\}$ and $B = \{B_1, \ldots, B_{k-1}\}$ is a set of Bloom Filters for those groups with all nodes in C1 loaded into their corresponding filter. For each group (g_i) that is shared between $C1$ and $C2$, we use the corresponding Bloom Filter (B_i) to check if a common node is shared. This method is particularly attractive as it takes $O(|C1|)$ time and memory to build the set of Bloom Filters and $O(|C2|)$ time to find common nodes.

When comparing only two structures to each other, the cost of creating a Bloom Filter may not necessarily offset the minimal gains. Ultimately, our aim should be to maximise the number of uses for each Bloom Filter and avoid having to remake a filter when possible.

If we have a $(k-1)$-GS $S^{k-1}_{\eth'}$ on G, it is possible to form a total of up to $(n-k+1)$ k-GSs with the inclusion of that many other $(k-1)$-GSs. If we only create Bloom Filters for the structures in $S^{k-1}_{\eth'}$ (we henceforth refer to this as a pivot), we can use these filters to perform the necessary common node checks as opposed to creating Bloom Filters for up to $(n-k+1)$ $(k-1)$-GSs which may save us a non-trivial accumulated time cost.

4.1.2 Reducing Maximal Biclique Enumeration Calls

Another simple heuristic we may adopt when selecting which $(k-1)$-GS to use as the pivot is choosing the one which contains the least number of $(k-1)$-PCs (i.e. $|S^{k-1}_{\eth'}|$ is the smallest). This not only further reduces the total number of Bloom Filters we create when running the combining algorithm but also reduces the number of maximal biclique enumeration calls.

LEMMA 3. *When trying to form a k-GS S^k consisting of groups $\eth$ and given a set of $(k-1)$-GSs $\mathbb{S}$ in which each $(k-1)$-GS contains $(k-1)$ groups of $\eth$, only:*

$$O\left(\min\{|S1| \cdot |S2| : \forall S1, S2 \in \mathbb{S}\right)$$

maximal biclique enumeration calls are required.

PROOF. From mathematical observation, there are $\binom{k}{k-1}$ possible combinations of $(k-1)$-GSs which may form a specific k-GS S. Therefore, by selecting the two $(k-1)$-GSs $S1$, $S2 \in \mathbb{S}$ whose product of $(k-1)$-PCs contained within them is the smallest, we require at most $|S1| \cdot |S2|$ maximum biclique enumeration calls to form S. □

Compared to the naive solution where any two $(k-1)$-PCs which share nodes in $(k-2)$ groups requires a maximal biclique enumeration call, by only performing the action on $(k-1)$-GSs which satisfy Lemma 3 we require potentially as $\frac{1}{k}$ many calls compared to the baseline. Additionally, choosing the smallest product allows us to attempt to avoid excess unnecessary calls compared to selecting two random $(k-1)$-GSs at random.

4.1.3 Improved Enumeration Algorithm

Algorithm 4: ImprovedMaximalkPCEnumeration

```
   Input  : S_{k−1}: All (k − 1)-GSs
   Output: S_k: All k-GSs
 1 S_k ← ∅; S_target ← all possible k-GSs given S_{k−1};
 2 while S_target is not empty do
 3 |   S1 ← Smallest (k − 1)-GS which may produce a
   |     structure in S_target;
 4 |   S_candidates ← ∅;
 5 |   foreach Possible k-GS S created with S1 do
 6 |   |   S2 ← Smallest (k − 1)-GS that may be
   |   |     combined with S1 to produce S;
 7 |   |   S_candidates.insert(S2);
 8 |   |   S_target.delete(S);
 9 |   end
10 |   foreach (k − 1)-PC C1 in S1 do
11 |   |   B ← Bloom Filters populated by nodes in
   |   |     C1;
12 |   |   foreach (k − 1)-GS S in S_candidates do
13 |   |   |   S_discovered ← ∅;
14 |   |   |   foreach (k − 1)-PC C2 in S do
15 |   |   |   |   if CommonNodeCheck(C1, C2, B)
   |   |   |   |     then
16 |   |   |   |   |   S_discovered.insert(Combine(C1, C2));
   |   |   |   |   |
17 |   |   |   |   end
18 |   |   |   end
19 |   |   |   S_k.insert(S_discovered);
20 |   |   end
21 |   end
22 end
```

Algorithm 4 details our complete improved combining algorithm which leverages both previously discussed heuristics as well as utilising the k-GS idea to avoid wasted operations whilst guaranteeing all outputs are reached. We input all $(k-1)$-GSs (which contain all maximal $(k-1)$-PCs) and

output all k-GSs (containing all maximal k-PCs). We define S_{target} as all possible k-GSs that may be formed on G (Line 1). We then select the $(k-1)$-GS $S1$, which contains the least number of maximal $(k-1)$-PCs, that may produce a k-GS in S_{target} (Line 3) which allows us to leverage Lemma 3.

Then for each create-able k-GS S, we select the smallest $(k-1)$-GS $S2$ which may be combined with $S1$. Our set of all of these $(k-1)$-GSs is called $S_{candidates}$. When $S1$ and $S2$ have been properly identified, S is removed from S_{target} (Lines 5-9).

Now, we begin the merging process. For each maximal $(k-1)$-PC $C1$ in $S1$ we create a Bloom Filter for each of its groups containing the corresponding nodes (Line 11). We then check all maximal $(k-1)$-PCs $C2$ in $S_{candidates}$ to see if they have satisfied the requirements for combining. If they do, we combine them using the method discussed in Algorithm 1 and add the result (if any) to the output whilst making sure to keep note of which k-GS each maximal k-PC output belongs to (Line 12-19).

We repeat this entire process until there are no more k-GSs that can be created (i.e. $S_{target} = \emptyset$). This algorithm requires at most $(k-1)$ Bloom Filters in memory at any given time, with the size of each Bloom Filter being variable to suit individual system limitations (with the drawback of increased false positive likelihood and subsequent associated costs with checking correctness).

The time cost of Algorithm 4 is:

$$O\left(\sum_{S1\in S_{k-1}}\sum_{C1\in S1}|C1|\cdot\left(\sum_{S2\in S_{k-1}}\mathbb{I}\cdot\sum_{C2\in S2}|C2|+\Gamma(C1,C2)\right)\right)$$

where $\mathbb{I}=1$ if $S1$ and $S2$ (which forms a specific k-GS) are the pair of $(k-1)$-GSs which have have the smallest product as per Lemma 3. $\Gamma(C1,C2)$ is the cost of maximal biclique enumeration for the nodes in the two off groups of $C1$ and $C2$.

By using Algorithm 4 instead of the baseline (Algorithm 1), we reduce the total number of common node checks to form all maximal k-GSs from $\Theta(\binom{n}{k-1}\cdot(\binom{n}{k-1}-1)))$ to $\Theta(\binom{n}{k})$. Additionally, we require as potentially as low as $\frac{1}{k}$ the number of maximal biclique enumeration calls compared to the baseline solution.

5. EDGE-MAXIMUM ALGORITHM

We will now propose heuristics which help directly target the *edge-maximum* k^*-PC structure whilst avoiding enumerating all maximal k^*-PCs. We propose two key heuristics, that being a pruning value and an inexpensive upper bound structure.

5.1 $E_{potential}$ Pruning Value

The first, most straightforward technique is to know when to abandon a structure or search that is unable to produce the desired output.

By storing the current number of edges (E_{max}) in the largest structure discovered so far, we then aim to prune unpromising searches later down the line. We can accomplish quite simply by calculating the maximum possible number of edges that may exist based on the number of current nodes in the structure to be expanded on plus the remaining candidate nodes. This can quickly be done via:

$$E_{potential}=\sum_{(g_i\neq g_j)\in\eth',\ i<j}|U_{g_i}|\cdot|U_{g_j}| \quad (1)$$

for all such nodes. If $E_{potential} < E_{max}$, we can stop searching the current structure. Alternatively, as a tighter bound, you may redefine E_{max} as the number of edges connecting these nodes. However this method is more expensive as it requires a scan of the edge list as opposed to a straightforward calculation.

5.2 Upper Bound Structure

An upper bound structure is one that holds a k^*-PC as a subgraph inside of it. By defining an upper bound structure which is less costly to compute compared to k^*-PC enumaration, we can quickly prune nodes which cannot be included in the *edge-maximum* k^*-Partite Clique.

5.2.1 $(a_1,\ldots,a_n)$-Core

One upper bound structure is the $(a_1,\ldots,a_n)$-core, which is a variation of the well established k-core structure [32] adapted to work on a k-Partite graph scenario. We define it as follows:

Definition 7. $(a_1,\ldots,a_n)$-core: For a graph G with groups $\eth=\{g_1,\ldots,g_n\}$, we say a subgraph A is a $(a_1,\ldots,a_n)$-core if each node $u_i\in g_i$ in A is connected to a_j nodes from g_j in A for all $j=\{1,\ldots,n\}, i\neq j$. We may use the notation $A_{(a_1,\ldots,a_k)}$ to denote a $(a_1,\ldots,a_n)$-core.

An important note is that we may set the value of any a_j value to be equal to 0. This essentially means that we are not interested in g_j for this particular core.

We define a maximal $(a_1,\ldots,a_n)$-core as a $(a_1,\ldots,a_n)$-core that is not contained in a larger $(a_1,\ldots,a_n)$-core. That is for any maximal $(a_1,\ldots,a_n)$-core A, there exists no $(a_1,\ldots,a_n)$-core A' such that $A\in A'$.

Firstly, we demonstrate that the $(a_1,\ldots,a_n)$-core may serve as an upper bound structure for the k^*-PC.

THEOREM 3. *A maximal k^*-Partite Clique C^{k*} that exists on G must be located inside of a maximal $(a_1,\ldots,a_n)$-core $A_{(a_1,\ldots,a_k)}$ where $a_i\leq|U_{g_i}|, U_{g_i}\in C^{k*}$.*

PROOF. Firstly we note that the $(a_1,\ldots,a_n)$-core, similar to the k-core, obeys a nested property. If we have a maximal $(a_1,\ldots,a_n)$-core A_a, then it must be located in a maximal $(a'_i,\ldots,a'_n)$-core $A_{a'}$ where all $a_i\geq a'_i,\forall i=\{1,\ldots,n\}$. This statement is fairly trivial to verify, given that if a node $u_j\in g_j, A_a$ is connected to a_i $(i\neq j)$ nodes from g_i in A_a then it must be connected to a'_i nodes in A_a.

To verify $C^{k*}\subseteq A_{(a_1,\ldots,a_n)}$ is equally trivial. Every node $u_j\in g_j$ in C^{k*} is connected to $|U_{g_i}|$ nodes from g_i in C^{k*} $(i\neq j)$ as per the definition of a k-PC.

From these two properties, we can verify Theorem 3 to be true. □

Next, we introduce an algorithm for finding maximal $(a_1,\ldots,a_n)$-cores from an input graph, similar to existing maximal k-core enumeration algorithms [5].

Algorithm 5 highlights our method for maximal $(a_1,\ldots,a_n)$-core enumeration on any given graph G. Firstly for each

Algorithm 5: MaximalCoreEnumeration

Input : G: Input Graph with n groups
a: Values for $\{a_1, \ldots, a_n\}$
Output: A: All maximal $(a_1, \ldots, a_n)$-cores

```
1  A ← ∅;
2  foreach Node u ∈ g_i in G do
3  |  {δ_u1, ..., δ_un} ← # Edges from u to g_1, ..., g_n;
4  end
5  foreach Node u ∈ g_i in G do
6  |  if δ_uj < a_j for any j = {1, ..., n}, i ≠ j then
7  |  |  G.remove(u);
8  |  |  DecremementNeighbours(u);
9  |  end
10 end
11 foreach Connected Subgraph A_connected in G do
12 |  A.insert(A_connected);
13 end
```

node $u \in g_i$ in G, we determine its degree value to each other group (denoted by $\{\delta_{u1}, \ldots, \delta_{un}\}$), where the value of $\delta_{ui} = 0$ (Lines 2-4). Then for each node, we examine if any of their degree values are less than the desired input values except the nodes own group (i.e. $\delta_{uj} < a_j, i \neq j$) (Line 5). If any value fails to reach the requirement, we remove u and all edges connected to u from the graph and then decrement all previous neighbours of u. If any of them then subsequently fall below the degree requirement, then we also remove them from G and decrement their neighbours (Lines 7-8). The resulting connected subgraphs of G are the maximal $(a_1, \ldots, a_n)$-cores (Lines 11-13).

Similar to Batagelj's maximal k-core enumeration algorithm via decomposition [5] (which runs in $O(|E|)$ time), our enumeration algorithm runs in $O(|E_{cross-group}|)$ time (as we do not consider inter-group edges) since we traverse each edge at most three times, twice when calculating degree values and once if we must remove a node.

We utilise the structure as a pruning tool to remove nodes that cannot exist in a k-PC $C^k_{\eth'}$ by setting the value of $a_i = 1$ if $g_i \in \eth'$ and $a_i = 0$ otherwise. We utilise this binary allocation as it guarantees all k-PCs on groups $\eth'$ will be found, as if any node from a given group is not connected to any nodes from another group, it cannot be in a k-PC. This upper bound structure allows us to narrow down the search space when finding the k-PC.

Additionally if we find the $(a_1, \ldots, a_n)$-core consisting of $C^k_{\eth'}$ and all the nodes in groups $\eth/\eth'$, we can derive the $E_{potential}$ of any k^*-PC which may be produced by merging $C^k_{\eth'}$, thus potentially pruning it for future merges.

5.2.2 $(b_1, \ldots, b_n)$-Truss

Another upper bound structure we have designed for this work is a $(b_1, \ldots, b_n)$-truss ($n \geq 3$), based on the established k-truss structure [9]. Naturally, the standard k-truss idea does not translate directly into a bipartite situation (due to the impossibility of triangles).

We will utilise the majority of standard terminology regarding trusses in our definition. We slightly modify the definition of triangle ($\triangle_{u,v,w}$) such that each node that makes the triangle must be a member of a different group ($u \in g_u, v \in g_v, w \in g_w$, $u \neq v \neq w$).

We maintain the idea of the 'support' of an edge (denoted by $sup(e, G)$), being the number of triangles in G which include edge e. Further details regarding general truss terminology can be found in [9]. We define our structure as follows:

Definition 8. $(b_1, \ldots, b_n)$*-truss:* For a graph G with groups $\eth = \{g_1, \ldots, g_n\}$, we say a subgraph B is a $(b_1, \ldots, b_n)$-truss if each edge e connecting a nodes from groups g_i and g_j has $sup(e, G) \geq \min\{(b_i - 2), (b_j - 2)\}$. We may use the notation $B_{(b_1,\ldots,b_n)}$ to denote a $(b_1, \ldots, b_n)$-truss.

We show that the $(b_1, \ldots, b_n)$-truss is an upper bound structure of a k-PC.

THEOREM 4. *A maximal k-Partite Clique C with groups $(g_1, \ldots, g_n)$ that exists on G must be located inside a maximal $(b_1, \ldots, b_n)$-truss $B_{(b_1,\ldots,b_n)}$ where $b_i \leq \sum_{g_j \neq g_i}(|U_{g_j}|) + 2, U_{g_j} \in C$.*

PROOF. We can trivially prove the nested property of the $(b_1, \ldots, b_n)$-truss structure using the same logic as the proof for Theorem 3. By the definition of the k-Partite Clique, we can determine the support of any edge e connecting nodes $u \in g_\alpha$ and $v \in g_\beta$ to have support $\min\{(\sum_{g_i \neq g_\alpha} |U_{g_i}|) + 2, \sum_{g_j \neq g_\beta} |U_{g_j}|) + 2\}$. As a result, a k-PC may also be defined as a $(\sum_{g_i \neq g_1} |U_{g_i}|)+2, \ldots, \sum_{g_j \neq g_n} |U_{g_j}|)+2)$-truss. □

Our maximal $(b_1, \ldots, b_n)$-truss enumerating algorithm is based on the existing k-truss decomposition algorithms [9].

Algorithm 6: MaximalTrussEnumeration

Input : G: Input Graph with n groups
b: Values for $\{b_1, \ldots, b_n\}$
Output: B: All maximal $(b_1, \ldots, b_n)$-trusses

```
1  B ← ∅;
2  foreach Edge e(u ∈ g_i, v ∈ g_j) in G do
3  |  sup(e) ← Support of e;
4  end
5  foreach Edge e(u ∈ g_i, v ∈ g_j) in G do
6  |  if sup(e) < min{b_i − 2, b_j − 2} then
7  |  |  G.remove(e);
8  |  |  foreach Node w ∈ neigh(u) ∩ neigh(v) do
9  |  |  |  ReduceSupport(u, w);
10 |  |  |  ReduceSupport(v, w);
11 |  |  end
12 |  end
13 end
14 foreach Connected Subgraph B_connected in G do
15 |  B.insert(B_connected);
16 end
```

Algorithm 6 details the maximal $(b_1, \ldots, b_n)$-truss decomposition algorithm. Firstly, we calculate the support of each edge e (Line 2-4) which can be done in $O(|E_{cross-group}|^{1.5})$ time. For each edge $e(u \in g_i, v \in g_j)$ with $sup(e) < \min\{b_i - 2, b_j - 2\}$, we delete it (Line 6-7). Then, for each node w connected to both u and v (which previously formed $\triangle_{u,v,w}$, we reduce the support of edges $e(u, w)$ and $e(v, w)$ (*ReduceSupport*()) and check they are still valid in our structure (Line 8-11). If their support is no longer valid, we also remove them from G and reduce the support of their affected edges.

Algorithm 6 runs in $O(\sum_{e(u,v)\in E} deg(u) + deg(v))$ time similar to the k-truss algorithm. We may simplify that to $O(|U|^2|E_{cross-group}|)$ time, which is polynomial. If all values of b are equal, we may utilise a variation of Wang and Cheng's $O(|E|^{1.5})$ time k-truss enumeration algorithm which utilises sorting to streamline the process [39].

Whilst maximal $(b_1, \ldots, b_n)$-truss decomposition is more expensive than maximal $(a_1, \ldots, a_n)$-core decomposition, it is still polynomial. Furthermore, we can show that it is a tighter bound than the $(a_1, \ldots, a_n)$-core in certain situations.

THEOREM 5. *A $(b_1, \ldots, b_n)$-truss $B_{(b_1,\ldots,b_n)}$ is a subgraph of a $(a_1, \ldots, a_n)$-core $A_{(a_1,\ldots,a_n)}$ when $b_i \geq \sum_{j\neq i} a_j + 1$ given the largest b value has two instances in $(b_1, \ldots, b_n)$.*

PROOF. Since we require that the largest b value must appear twice, each node $u \in g_i$ must be connected to at least one edge with support b_i. Therefore, u must be connected to at least $b_i - 1 = \sum_{j\neq i} a_j$ nodes in other groups. □

Since the nested property holds true, similar to how the k-truss is tighter variation of the k-core in regards to pruning nodes to find cliques, the $(b_1, \ldots, b_n)$-truss prunes more nodes than the $(a_1, \ldots, a_n)$-core with the aim of finding k-PCs with the trade-off of a more expensive (yet still polynomial) runtime.

When creating an upper bound structure for a k-PC $C^k_{\eth'}$, we set the value of $b_i = k$ if $g_i \in \eth'$, since each edge connecting $u \in g_i$ must be connected to at least $(k-2)$ other groups. We set $b_i = +\infty$ otherwise, as to not include edges connecting nodes from groups we are currently unconcerned with. If any truss produced is missing nodes from any required group, we discard it.

We may further find a $(a_1, \ldots, a_n)$-core consisting of the nodes in the found truss and all nodes in groups $\eth/\eth'$ to prune the found truss for future merges using $E_{potential}$.

6. EXPERIMENTAL STUDY

In order to demonstrate the effectiveness and efficiency of our algorithm and heuristics, we have conducted extensive experiments on both real and semi-synthetic networks. In this section, we first present the experimental setting and then analyse the experimental results.

6.1 Experimental Setting

6.1.1 Datasets

All networks we utilise in our experimentation are publicly available[1], though some have been artificially labelled (thus semi-synthetic) in order to be eligible towards our problem (namely, having a known group).

To generate synthetic data in a way that is realistic, we utilise the method developed by Nettleton [27] which combines distribution profiles, real demographic data as well as the inherent ground truth communities in order synthetically populate a graph in a realistic manner. This allows us to capture the realistic likelihood that users connected to each other in traditional communities are more likely to share at least some common attributes.

The following datasets are used:

[1]All datasets are from https://snap.stanford.edu/data/

- **p2p-Gnutella4:** We assign each user an educational group (from 5 possible labels) based on 2019 information by the United States Census Bureau [36].
- **musae-Facebook:** Each page is categorised into one of five groups (Politicians, Governmental Organisations, Television Shows, Companies and Undefined).
- **com-Amazon:** We assign each node a label from 8 professions, based on information by the U.S. Bureau of Labor Statistics [34].
- **com-DBLP:** We assign each author one of seven religious groups based based on information and demographic distribution from the Pew Research Center [29].
- **com-Youtube:** We assign each user one of 16 groups based on age and sex demographics from information provided by U.S. Census Bureau [35].

These datasets are commonly used in the field of community detection [41]. Further information regarding these datasets may be found in Table 2.

Table 2: Dataset Information

Dataset	Groups	$\lvert U\rvert$	$\lvert E_{cross-group}\rvert$
Gnutella	5	10,876	23,419
Facebook	5	22,470	19,622
Amazon	8	334,863	535,624
DBLP	7	317,080	602,872
Youtube	16	1,134,890	2,056,737

6.1.2 Implementation

All code is implemented in Standard C++ and ran on a Intel(R) Core i7-8565U CPU @ 1.80GHz with 16GB of RAM.

6.2 Experimental Results

In this subsection we report and examine the results of our experimentation.

6.2.1 Effectiveness Comparison

The metric that we will use to examine diversity is the Entropy value (E) [23] which is used to determine the diversity of a collection of items (from multiple groups) in a set of size n. To derive the E value we can use the following formula:

$$E = \sum_{i=1}^{n} \pi_i \ln(\frac{1}{\pi_i}) \quad (2)$$

π_i represents the proportion of users from group i in the community. If no member of group i is in our current community, we set the value of $\ln(\frac{1}{\pi_i}) = 0$. We then normalise this E value into an Entropy Index value (EI) between 0 and 1. An EI value of 0 implies that the community has minimal diversity (i.e. there exists only one group in the community) whilst an EI value of 1 implies that the community has maximal diversity (i.e. all groups have exactly equal participation).

6.2.2 *Effectiveness Analysis*

As a baseline to examine the diversity of the communities discovered by our method, we compare the top 500 k^*-PCs (based on the number of cross group edges) to the top 500 $(a_1, \ldots, a_n)$-core communities and top 500 $(b_1, \ldots, b_n)$-truss communities for appropriate values of a_i and b_i. The Core baseline is the set of all $(a_1, \ldots a_n)$-cores where k a values are set to 1 and the remaining are set to 0, which produces an output structure with at least k groups. The Truss baseline is the set of all $(b_1, \ldots, b_n)$-trusses of $> k$ groups where all b_i values are set to 3 (i.e. each edge is a member of at least one triangle) when g_i is to be included in the truss or ∞ as to not include g_i in the truss structure. If a truss is outputted that contains less than k groups, we erase it.

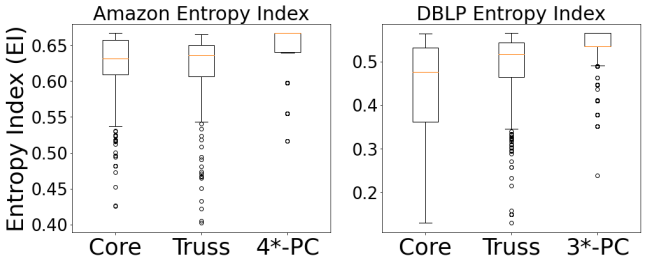

Figure 5: Boxplots detailing the distribution of the Entropy Index values for the Top 500 outputs over the Amazon and DBLP datasets for the Core and Truss baseline compared to the k^*-PC.

From Figure 5, we can see that the EI value of the top 500 k^*-PCs compared to the top 500 Core and Truss baselines (with members from at least k groups) on the Amazon and DBLP datasets displays an increased effectiveness from our model.

Figure 6 allows us to observe the top 500 k^*-PCs on both the Amazon and DBLP datasets. It is clear that as the k value increases, the diversity of our outputted structures also increases due to the inherent requirement of more groups. Importantly, the EI value produced by the top k^*-PC cannot be worse than the EI value of the top $< k^*$-PC. An important note is that there exists no 7-PC and subsequently no 8-PC on the Amazon dataset due to the way the groups were synthetically generated. This type of problem may often exist on real world datsets with either many groups or

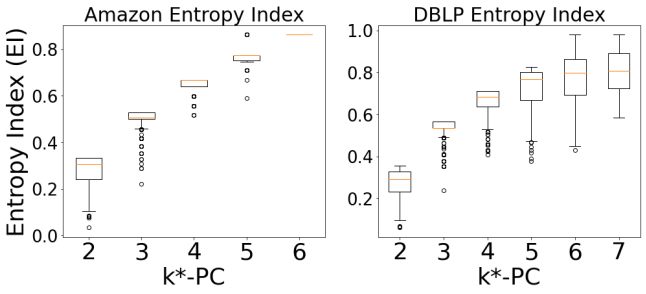

Figure 6: Boxplots detailing the distribution of the Entropy Index values for the Top 500 k^*-PC outputs over the Amazon and DBLP datasets for growing values k.

Table 3: The runtime (seconds) of the three enumeration methods when finding 2^*-PCs.

Dataset	Baseline	RandomProd	LowestProd
Gnutella	2,331	566	318
Facebook	1,539	683	340
Amazon	-	203,343	125,705
DBLP	-	-	141,523

groups with extremely small representation which further illustrates the need for the flexibility afforded by the k^*-PC compared to the standard k-PC.

By examining our effectiveness, we can draw the conclusion that the communities outputted by our k^*-PC structure is a reasonable structure for the problem of finding diverse communities on networks.

6.2.3 *Efficiency Comparison*

We examine the runtime cost of our baseline enumeration algorithm detailed in Section 4 in comparison to the trivial baseline consisting of independent k-Partite Enumeration for $k = 2, \ldots, n$ over all our datasets. Then we examine the effects of our heuristics (introduced in Section 4.1) in terms of improvements to efficiency when enumerating all k^*-PCs over the dataset.

Furthermore, we examine the efficiency gains of our *edge-maximum* specific improvements (detailed in Section 5). In particular we look at the costs associated with $(a_1, \ldots, a_n)$-core decomposition and the effects the upper bound structure has on the *edge-maximum* k^*-Partite Clique problem.

6.2.4 *Efficiency Analysis*

When enumerating all k^*-PCs on a given dataset, we consider the three following methods:

- **Baseline:** The baseline algorithm detailed in Algorithm 2 in Section 4 with no heuristics selected to avoid unnecessary comparisons and maximal biclique enumeration calls.
- **RandomProd:** An improved heuristic algorithm which utilises the Bloom Filter technique to avoid unnecessary comparisons when finding common nodes between two $(k-1)$-PCs. In Random Product, the $(k-1)$-GSs which are to be merged are chosen at random.
- **LowestProd:** The same heuristic implementation of Bloom Filters as in RandomProd with the added heuristic that the smallest $(k-1)$-GSs are chosen to build Bloom Filters and compare groups, which attempts to reduce the number of maximal biclique enumeration calls. This is the method described in Algorithm 4.

To demonstrate the power of the heuristics on k^*-PC enumeration, we examine the 2^*-PC enumeration scenario, which is the most expensive enumeration procedure that can be performed over any dateset as any other k^*-PC $(k > 2)$ is a subset of it. RandomProd is used to demonstrate the effects of the LowestProd heuristic.

Table 3 illustrates the runtime associated with each of the enumeration scenarios on all four datasets. Notably, Baseline for Amazon and DBLP as well as Random Product for DBLP did not terminate after being ran for 60 hours due

Table 4: Runtime (seconds) required to find the *edge-maximum* 3^*-PC.

Dataset	Enumeration	Core	Truss
Gnutella	190	128	15
Facebook	210	7	6
Amazon	93,422	11,041	9,721
DBLP	114,744	62,693	42,149
Youtube	-	-	154,683

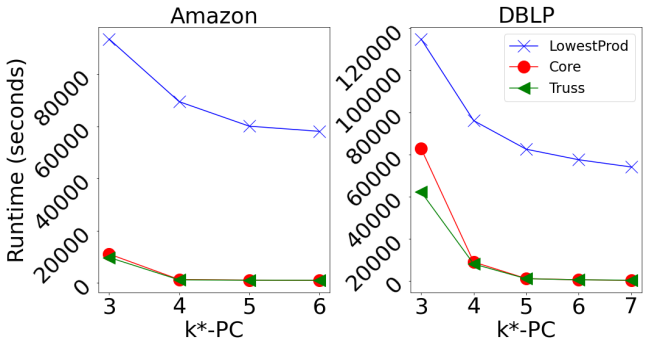

Figure 7: Effects of runtime for increasing values of k over Amazon and DBLP.

to the size of the dataset. The Youtube dataset did not terminate using any three of the enumeration techniques given its size. It is clear to see that the two heuristic approaches in RandomProd and LowestProd provide gains in reducing the runtime of k^*-PC enumeration, with a reasonable margin between RandomProd and LowestProd. In particular, LowestProd is observed to hold considerable improvements over the trivial Baseline.

Additionally, we wish the examine the improvement in efficiency when observing the edge-maximum specific heuristics. We utilise the $E_{potential}$ pruning value when conducing our search. We set the Baseline as k^*-PC Enumeration using the LowestProd method. For our heuristic methods, one method utilises the $(a_1, \ldots, a_n)$-core (using the binary allocation method described in Section 5.2.1.) as the upper bound structure whilst the other one uses the $(b_1, \ldots, b_n)$-truss (using the allocation method described in Section 5.2.2.) as the upper bound structure. We call them Core and Truss respectively.

From Table 4, we can see the improvement in runtime for our heuristic methods for finding the edge-maximum 3^*-PC on our datasets. Notably, both Core and Truss result in significant improvements due to the pruning of nodes which are unable to be a part of our final output. Both Enumeration and Core did not terminate after 100 hours for the Youtube dataset, but Truss terminated within a reasonable time frame indicating the effectiveness of that heuristic on that dataset. Being able to reasonably reduce the search space in polynomial time when conducing maximal biclique enumeration, which is the bottleneck of our algorithm, can result in significant gains in performance. Additionally we can see that Truss solution is in general faster than Core, despite $(b_1, \ldots, b_n)$-truss enumeration being more expensive than $(a_1, \ldots, a_n)$-core enumeration, since the truss is tighter than the core.

From Figure 7, we examine the effects of our heuristic methods as the value of k increases over the Amazon and DBLP datasets. From this, we observe the significant effects of Core and Truss when pruning nodes for larger k values.

7. DISCUSSION AND FUTURE WORK

From our experimental results we may conclude that our structure does indeed model diverse communities relative to the network it exists upon. Given that the field of community detection for large, diverse communities is largely in its infancy there are multiple potential future work topics.

Firstly, we may look to improve the efficiency of the algorithms described in this paper by exploring other techniques used in network analysis and more specifically subgraph enumeration. Other ideas, such as leveraging previously discovered maximal bicliques when combining two $(k-1)$-PCs offer potentially significant time cost savings as a trade-off for memory requirements. Furthermore, local search approaches are an interesting direction for very large graphs. Further exploration of the $(a_i, \ldots, a_n)$-core and $(b_1, \ldots, b_n)$-truss structures are also interesting.

Additionally, work may be conducted on determining a suitable value of k given the inherent properties displayed by the input network. This area is of exceptional interest as it allows for the removal of either trial and error or examining values of k which are 'too low', which are simultaneously the most expensive queries.

8. CONCLUSION

In conclusion, we tackle the problem of finding large communities of diverse users on a network by proposing the k^*-Partite Clique (with a particular focus on the *edge-maximum* k^*-Partite Clique problem) as a means of modelling these communities. Given the problem was previously undefined, we show that it is NP-Hard and NP-Hard to approximate. Importantly, we demonstrate using the Entropy Index as a means of measuring diversity that our structure is indeed more diverse relative to a $(a_1, \ldots a_n)$-core and $(b_1, \ldots, b_n)$-truss baseline. We developed a non-trivial baseline enumeration algorithm for all k^*-PC over a network as well as introducing multiple heuristics that significantly reduce the costs associated with running our algorithms. Additionally, we propose and demonstrate some other heuristics aimed at finding the *edge-maximum* k^*-PC over the network. Our extensive experiments conducted on real-world graphs corroborate both the effectiveness and efficiency claims in this work.

Acknowledgements

Lei Chen's work is partially supported by the Hong Kong RGC GRF Project 16207617 , CRF Project C6030-18G, C10 31-18G, C5026-18G, AOE Project AoE/E-603/18, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, Didi-HKUST joint research lab project, and Wechat and Webank Research Grants. Yue Wang's work is partially supported by Guangdong Basic and Applied Basic Research Foundation 2019A1515110473. Corresponding Author: Yue Wang.

9. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. *VLDB'94*, pages 487–499, 1994.

[2] L. Akoglu. Quantifying political polarity based on bipartite opinion networks. *ICWSM*, pages 2–11, 2014.

[3] T. Ali, C. J. Nilsson, J. Weuve, K. B. Rajan, and C. F. M. D. Leon. Effects of social network diversity on mortality, cognition and physical function in the elderly: a longitudinal analysis of the chicago health and aging project (chap). *JECH*, 72(11):990–996, 2018.

[4] V. Bakir and A. Mcstay. Fake news and the economy of emotions. *Digit. Journal.*, 6(2):154–175, 2017.

[5] V. Batagelj and M. Zaveršnik. An o(m) algorithm for cores decomposition of networks. 2003.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13:422–426, 1970.

[7] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full aes. *ASIACRYPT*, pages 344–371, 2011.

[8] J. Brown, A. J. Broderick, and N. Lee. Word of mouth communication within online communities: Conceptualizing the online social network. *J. Interact. Mark.*, 21(3):2–20, 2007.

[9] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. Technical Report MD 20755-6513, National Security Agency, 9800 Savage Road, Fort Meade, 2008.

[10] M. Conover, J. Ratkiewicz, M. Francisco, B. Gonçalves, F. Menczer, and A. Flammini. Political polarization on twitter. *ICWSM*, pages 89–96, 01 2011.

[11] T. H. Cox, S. A. Lobel, and P. L. McLeod. Effect of ethnic group cultural differences on cooperative and competitive behaviour on a group task. *AMJ*, 34(4):827–847, 1991.

[12] N. Eagle, M. Macy, and R. Claxton. Network diversity and economic development. *Science*, 328(5981):1029–1031, 2010.

[13] U. Feige and S. Kogan. Hardness of approximation of the balanced complete bipartite subgraph problem. Technical Report MCS04-04, Weizmann Institute, Rehovot 76100, Israel, 2004.

[14] R. B. Freeman and W. Huang. Collaborating with people like me: Ethnic co-authorship within the us. *NBER*, 2014.

[15] K. Garimella, G. D. F. Morales, A. Gionis, and M. Mathioudakis. Reducing controversy by connecting opposing views. *WSDM*, pages 81–90, 2017.

[16] N. Gillis and F. Glineur. A continuous characterization of the maximum-edge biclique problem. *J. Global Optim.*, 58(3):439–464, 2013.

[17] M. Girvan and M. Newman. Community structure in social and biological networks. *PNAS*, 99:7821–7826, 2001.

[18] T. Grünert, S. Irnich, H.-J. Zimmermann, M. Schneider, and B. Wulfhorst. Finding all k-cliques in k-partite graphs, an application in textile engineering. *Comput. Oper. Res.*, 29(1):13–31, 2002.

[19] X. Huang, H. Cheng, R.-H. Li, L. Qin, and J. Yu. Top-k structural diversity search in large networks. *VLDBJ*, 24, 02 2015.

[20] V. Hunt, D. Layton, and S. Prince. Diversity matters. *McKinsey & Company*, 2015.

[21] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78(4), 2008.

[22] Q. Liu, Y.-P. P. Chen, and J. Li. k-partite cliques of protein interactions: A novel subgraph topology for functional coherence analysis on ppi networks. *J. Theor. Biol.*, 340:146–154, 2014.

[23] D. S. Massey and N. A. Denton. The dimensions of residential segregation. *SF*, 67(2):281–315, 1988.

[24] A. Matakos and A. Gionis. Tell me something my friends do not know: Diversity maximization in social networks. *ICDM*, 1:327–336, 2018.

[25] M. Mirghorbani and P. Krokhmal. On finding k-cliques in k-partite graphs. *Optim. Lett.*, 7:1155–1165, 2013.

[26] A. P. Mukherjee and S. Tirthapura. Enumerating maximal bicliques from a large graph using mapreduce. *IEEE Big Data*, 10, 2014.

[27] D. F. Nettleton. A synthetic data generator for online social network graphs. *SNAM*, 6:1–33, 2016.

[28] R. Peeters. The maximum edge biclique problem is np-complete. *Discret. Appl. Math.*, 131(3):651–654, 2003.

[29] Pew Research Center. America's changing religious landscape. Technical Report RLS-08-06, 2015.

[30] C. Phillips, K. Wang, E. Baker, J. Bubier, E. Chesler, and M. Langston. On finding and enumerating maximal and maximum k-partite cliques in k-partite graphs. *Algorithms*, 12(1):23, 2019.

[31] M.-É. Roberge and R. V. Dick. Recognizing the benefits of diversity: When and how does diversity increase group performance? *HRMR*, 20(4):295–308, 2010.

[32] S. B. Seidman. Network structure and minimum degree. *Soc. Netw.*, 5(3):269–287, 1983.

[33] P. Turan. On an external problem in graph theory. *KöMaL*, 48:436–452, 1941.

[34] U.S. Bureau of Labor Statistics. Employment by major industry sector, 2019.

[35] U.S. Census Bureau. Age and sex compositions: 2010, 2011.

[36] U.S. Census Bureau. Educational attainment in the united states: 2019, 2020.

[37] M. D. Vicario, G. Vivaldo, A. Bessi, F. Zollo, A. Scala, G. Caldarelli, and W. Quattrociocchi. Echo chambers: Emotional contagion and group polarization on facebook. *Sci. Rep.*, 6(1), 2016.

[38] H. Wang, W. Wang, J. Yang, and P. Yu. Clustering by pattern similarity in large data sets. *IEEE Big Data*, 3:394–405, 2002.

[39] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[40] J. Wang, J. Cheng, and A. W.-C. Fu. Redundancy-aware maximal cliques. *SIGKDD*, pages 122–130, 2013.

[41] J. Yang and J. Ledkovec. Defining and evaluating network communities based on ground-truth. *ICDM*, pages 745–754, 2002.

[42] J. Yang, J. Mcauley, and J. Leskovec. Community detection in networks with node attributes. *ICDM*, 2013.

[43] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Diversified top-k clique search. *ICDE*, pages 387–398, 2015.

[44] F. Zhang, L. Yuan, Y. Zhang, L. Qin, X. Lin, and A. Zhou. Discovering strong communities with user engagement and tie strength. *DASFAA*, pages 425–441, 2018.

[45] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: efficient (k, r)-core computation on social networks. *PVLDB*, 10(10):998–1009, 2017.

[46] Y. Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC Bioinformatics*, 15(1):110, 2014.

[47] F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *PVLDB*, 6(2):85–96, 2012.