Efficient Multicore Discovery of Small, High-Quality k-Plex Teams in Multi-attributed Networks

Parisa Esmaeilian Ghahroudi
Department of Computer Science
Victoria, Canada
parisaesmaeilian@uvic.ca

Sean Chester
Department of Computer Science
Victoria, Canada
schester@uvic.ca

Alex Thomo
Department of Computer Science
Victoria, Canada
thomo@uvic.ca

ABSTRACT

Traditional community search identifies subgraphs as important because they maximise structural properties like cohesion or size. However, many real-world graphs additionally describe vertices with multiple properties. For these multi-attributed networks, one can instead maximise the vertex properties subject to a subgraph cohesion threshold, thereby finding high-quality teams composed of strong individuals per characteristics of user-defined relevance.

This work makes three complementary but orthogonal advances. First, it introduces a novel group skyline model that can compare social teams that have different sizes and different member specialisations. Second, it generalises algorithms for listing small cliques to quasi-cliques using a cousins-first search strategy that improves asymptotic complexity. Finally, we illustrate a previously unknown challenge that arises from composing selection, grouping, and skyline operators. This informs our integration of our subgraph listing method within a robust multicore skyline framework. Compared to the state-of-the-art—which can only handle cliques—and our semaphore-based parallelisation of it, we demonstrate up to a 47.9× improvement with our more general algorithm when using a single core and 950.9× with 48 cores.

1 INTRODUCTION

Recently, there has been a lot of interest in identifying communities in graphs that are maximal in terms of their vertex attributes [1, 20, 26, 28–30, 46, 49, 50]. Compared to traditional community search, which focuses only on structural properties, considering the attributes on vertices as well opens up a lot of new applications, such as comparing potential teams based on quantitative metrics rather than just on their social cohesion [28, 29, 49]. However, despite this wide research interest, a number of serious challenges remain: a) modelling group "dominance" can bias against diversity *and* require fixed team sizes; b) listing small teams in large networks requires artificially high cohesion thresholds; c) the effects of composing selection, grouping, and skyline operators are not well understood; and d) scaling up skyline community search misses opportunities for parallelism.

Modelling challenge The first challenge is simply how to compare groups in social networks where everyone is described by multiple attributes. The example in Figure 1 embeds a graph with six vertices into 2d space based on their vertex attributes: the nodes represent, say, employees in a company who should form pods for a project; the edges represent some notion of affinity, such as having worked together before; and the spatial axes plot each employee's proficiency in relevant skills, say, C# and NodeJS. How could we tell which groups are likely to be the most successful, considering both raw skill and team cohesion?

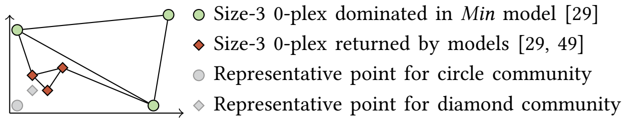


Figure 1: Illustration of community search in multi-attributed networks. The vertices of a toy graph are plotted in the plane. Vertices represent employees, spatial coordinates represent, say, skill level in C# and NodeJS, and edges between vertices represent prior collaboration. A k-plex team is one in which nobody has more than k new teammates and we want to find the "most skilled" such teams.

Zhang et al. [49] use the notion of group dominance [33] from the skyline literature. (We call this model *Permute*). In this model, a group is better than another group if one can construct some one-to-one mapping between them such that each member of the better group is paired with a distinct member of the dominated group who is equal or weaker on every attribute. Aside from being expensive to compute and a very high threshold to declare that one group is superior, this model is constrained to comparing groups of a preset size. This fundamentally limits which graph communities can be compared and therefore also which structural cohesion thresholds can be applied.

Li et al. [29] address this by instead applying a min-aggregation across the group independently to each attribute, thereby casting groups of arbitrary size into points of a fixed dimension. They can then be compared with a standard max-skyline operator. (We call this model *Min*). However, this "worst-case analysis" penalises diversity: if two people have different, complementary skills, the *Min* model evaluates the partnership according to the skill that each person *does not have* and ignores the strengths that they bring. Such a partnership would be represented by a weak aggregate point close to the origin, even though the partnership has excellent complementarity. The model favours groups of mediocrity in which nobody is especially bad at anything.

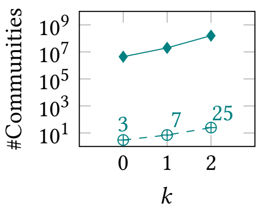
This work relaxes group dominance to allow a member of one group to "dominate" many others in another group. If the group dominance of [33] can be viewed as ensuring that you cannot replace *every member* of a group with a better choice, we propose that a group is not dominated if it has *somebody* who is irreplaceable. This model captures the strengths of both earlier models—an ability to compare any two arbitrary groups without sacrificing diversity. We also introduce a novel strategy of composing fine-grained models to elicit specific traits.

In Figure 1, we try to find the best 3-cliques. The *Min* model [29] prefers the clearly worse red group because the green one has a C# developer who is unskilled in NodeJS and vice versa. Meanwhile, our proposed model and the *Permute* model [33] return the better green group, but the *Permute* model only works because we know in advance that the team size should be exactly three.

We compare the model properties more thoroughly in Section 2. Later, in Section 5, we demonstrate with a case study on a real multi-attributed coauthorship network that our proposed model can retrieve the most intuitive, variable-size co-author clusters, while *Min* misses obvious powerhouse research clusters and *Permute* has low selectivity and cannot adapt to this scenario in which one of the top groups has more researchers than the other top group.

Listing small teams The second challenge arises when an application calls for small teams, the number of which can explode combinatorially.

The plot to the right previews Figure 6b from our experiments using the Amazon dataset, a real graph with more than 3 million edges. It compares our model's output size (the dashed line) to the number of five-member teams (the solid line) as the cohesion threshold, k, is relaxed (i.e., increased). With tens to hundreds of millions of communities to inspect, just listing them requires a very efficient algorithm.



Zhang et al. [49] partially resolve this by only permitting teams that are cliques ($k = 0$ in the plot above). One can then use the clique listing algorithm of Danisch et al. [13]. However, requiring teams to be cliques limits this discovery problem to already-connected people; it does not promote new collaborations. The restrictiveness of this is illustrated in our case study on coauthorship networks (Section 6), where only one research cluster returned is a clique. We want to relax the clique constraint while still being able to list candidate communities efficiently.

A natural choice is to identify k-plex teams of size g. Those are close to g-cliques, except that each member is not connected to up to k others [41] (the x-axis in the plot above). For example, in Figure 1, the two 3-cliques are also 0-plexes of size $g = 3$. Also, the green employees and the two upper red employees in Figure 1 form a 2-plex of size 5: everyone has already worked successfully with at least half of the other team members before. Although it substantially increases the search space, we prefer to consider such teams as well.

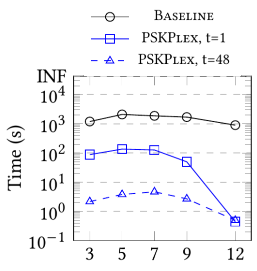
Earlier community search problems, e.g., [11, 32, 42, 47], identify maximal communities and can be discovered efficiently with branch-and-bound style techniques [10, 12, 34, 52], which are adopted by Li et al. [29]. However, they are not optimised for small teams. Our algorithm is based on a counter-intuitive iteration strategy that focuses first on vertices that are *not* neighbours, leading to a better asymptotic worst case time complexity in the bulk synchronous parallel (BSP) model of computation. This furthermore facilitates multi-threading.

Scalability challenge The third major challenge is that calculating skylines over exponentially many small communities is too expensive. Naively listing all k-plexes and then comparing them to each other is impractical. As already noted in the plot above, there are half a dozen orders of magnitude difference between the number of candidate teams and those that should be output.

Prior algorithms can take hours on relatively small datasets, which makes this problem unsuitable to interactive contexts. This problem is exacerbated by the fact that naively applying heuristics used in group skyline computation do not immediately work for graphs, where there are constraints on which nodes can form acceptable groups, as we will show in Section 4. In fact, this unrecognised challenge would be true in any context where group skyline computation is composed with selection.

To improve performance, we interleave skyline calculation into our efficient, parallel k-plex listing algorithm, using the synchronisation ideas from Chester et al. [7] and graph-specific ordering techniques that are robust to the theoretical challenges just noted.

Run sequentially on large datasets, our proposed algorithm, PSKPlex, achieves speedups up to 47.9× over our own optimisations to the state-of-the-art baseline [49], and up to 950.9× when running in parallel on 48 cores. On many problem instances, the baseline runs in several hours while our multi-threaded version runs in a minute or two.



Contributions and outline In this work, we make three orthogonal contributions to improve community search on multi-attributed networks. Namely, we introduce:

- A *General* model of group dominance that can be used to compare any two communities of any size without penalising collaborations among specialists (Section 2).
- A novel method to list fixed-sized connected k-plexes based on a counter-intuitive cousins-first-search strategy, demonstrating an asymptotic improvement in BSP (Section 3).
- A counter-example to show how existing group skyline algorithms can fail when composed with selection predicates. Moreover, the first algorithm to list skyline k-plexes, which achieves up to 950.9× speedup against our parallelisation and optimisation to the state-of-the-art for cliques (Section 4).

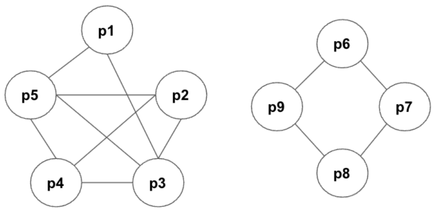
2 PROBLEM MODELLING & BACKGROUND

2.1 Defining teams

We assume a simple, undirected, graph, $G = (V, E)$, where V is the set of vertices and E is the set of edges, with vertex-labelling function $L : V \longrightarrow \mathbb{R}^d$. When $d > 1$, we call this a *multi-attributed network*, because there is more than one attribute assigned to each vertex. Figure 2 illustrates an example with the graph connectivity, E, rendered independently from the vertex labels, L. As described soon, the vertex labels can originate from any domain over which a partial order can be constructed, but our examples will generally consider them to be measurements of quality.

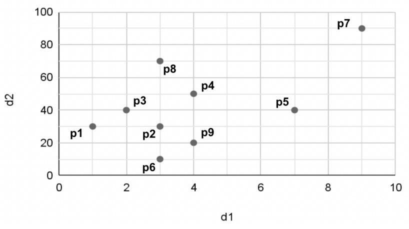
The goal is to find the best teams (a.k.a., sub-communities) within a multi-attributed network. A subset of vertices $V' \subseteq V$ is *a team* if the subgraph induced on them, $H_{V'} = (V', E', L)$, $E' = \{e = (u, v) : e \in E \wedge \{u, v\} \subseteq V'\}$, meets a predefined cohesion threshold that is specified in terms of the subgraph structure. In many cases, and throughout this paper, this is defined in terms of the degree of each node v, δ_v, though one could incorporate other notions of cohesion.

Zhang et al. [49] search for the best *cliques* whose size is a small, user-specified constant, g. In other words, teams are subgraphs for which $|V'| = g$ and $\forall v \in V', \delta_v = g - 1$. The clique constraint is too restrictive for most applications; so, *quasi-cliques* relax



(a) Simple, undirected graph, $G = (V, E)$

point	d1	d2
p1	1	30
p2	3	30
p3	2	40
p4	4	50
p5	7	40
p6	3	10
p7	9	90
p8	3	70
p9	4	20



(b) Vertex-labelling function, L

Figure 2: An example multi-attributed network, $G = (V, E, L)$

how many edges must exist for a graph to be considered "clique-like." We focus on a natural generalisation, called a k-plex [41], in which each vertex of the quasi-clique can lack up to k edges. Note that our graphs are simple: i.e., self-loops are not permitted and not counted as potential connections. Thus, a clique is a 0-plex.

Definition 2.1 (k-plex [41]). A k-plex is a graph $G = (V, E)$ in which $\forall v \in V, \delta_v \geq |V| - k - 1$.

Following most research on maximal k-plexes, we exploit the property that if $g \geq 2k + 1$, then every k-plex is connected [41]. Intuitively, a k-plex with disjoint components is not really a "team." By assuming in this work that $g \geq 2k + 1$, we thereby assume that each vertex is linked to at least half of the others.

In Figure 2, the teams $\{p_2, p_3, p_4\}$ and $\{p_2, p_3, p_4, p_5\}$ are 0-plexes of size 3 and 4, respectively (i.e., a 3-clique and a 4-clique). The teams $\{p_1, p_3, p_4, p_5\}$ and $\{p_6, p_7, p_8, p_9\}$ are 1-plexes with $g = 4$ and the team $\{p_1, p_2, p_3, p_4, p_5\}$ is a 2-plex with $g = 5$. The 0- and 1-plexes are also 2-plexes. We want the ability to compare all of these teams, rather than restricting ourselves to cliques with fixed size g,[1] leading us to the new model described below.

2.2 Comparing teams

To compare cohesive teams, we use the vertex labels, as in [26, 29, 30, 49, 50]. If the labels are non-numeric, we assume they have been mapped to an ordinal domain (i.e., partitioned into equivalence classes). For example, conferences can be mapped onto the set {0,1,2} based on whether they are low-, medium-, or high-impact; and the CVs of job applicants can be mapped to a discrete multi-point scale for each criterion based on how competitive they are. For cardinal attributes, like paper citations, a mapping is not required.

It is easy to decide whether a vertex is better than another w.r.t. a single attribute. However, comparing teams with multiple members and multiple attributes is not straightforward. This difficulty is compounded if any of the attributes are not cardinal and therefore cannot be meaningfully ranked by linear functions.

In such scenarios, it is natural to search the pareto frontier, i.e., use the skyline operator, briefly reviewed below. This addresses the multi-variate and ordinal domain challenges, but it is not clear how to compare entire teams, especially those with different sizes.

To recall, the skyline is the set of non-dominated points.

Definition 2.2 (Point Dominance [2, 24]). Given two points that are not coincident, u and v, in $\mathbb{R}^d$, where u^i denotes the value of the i'th attribute of u, we say u dominates v, denoted by $u \succ v$, iff $u^i \geq v^i$ for all $i \in [1, d]$, i.e., there is no attribute on which v is better.

[1]For the algorithmic development later, we will assume a fixed size g to facilitate efficient listing on large graphs, but we want a model that is not constrained in this way and that can be used in the flexible way that we demonstrate in our case study.

In the cases that all attributes are total orders (called *distinct value condition*) or the input points are a set, the distinctness check (that $u \neq v$) is not required [7]. While Zhang et al. [49] implicitly make this assumption, repeated values and coincident points in our case study compel us to handle the general case, which will introduce more challenges to our algorithms later. Thus, we also use $u \preceq v$ to denote that either u dominates v or they are coincident.

Definition 2.3 (Skyline [2]). Given a multiset of points P, the skyline multiset of P is $SKY(P) = \{u : u \in P \wedge \nexists v \in P, v \succ u\}$.

Example 2.4. In Figure 2, $SKY(P) = \{p_7\}$. Every other point is dominated by p_7 as it is not coincident to p_7 and has neither a larger x-value nor y-value.

If we construct P' by mirroring P about the line $y = -x$, say by subtracting every attribute value from 100, then $SKY(P') = \{p_1, p_6\}$. Every other point is dominated by p_1, p_6, or both.

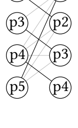
Considering teams, it is not obvious how to generalise these definitions from multisets of points to multi-attributed graphs. We recap existing definitions for general groups of points. The first model, *Permute*, applies dominance recursively.

Definition 2.5. (Group-based skyline [33]) For groups G and G' of g points in $\mathbb{R}^d$, we say that G *dominates* G', denoted $G \succeq_{Permute} G'$, iff there exists a permutation $\hat{G} = \langle \hat{G}[1], \ldots, \hat{G}[g] \rangle$ of G such that:

$$\bigwedge_{i=1}^{g} \hat{G}[i] \succeq_{Permute} G'[i] \text{ and } \bigvee_{i=1}^{g} \hat{G}[i] \neq G'[i].$$

Stated simply, points u in G and points v in G' are mapped one-to-one such that $u \succeq v$; also, distinctness must be checked. The model clearly requires groups to be the same size. Moreover, it has been criticised by Zhu et al. [53] for being prohibitively unselective.

Example 2.6. In Figure 2, $\{p_2, p_3, p_4, p_5\} \succ \{p_1, p_2, p_3, p_4\}$. We can construct a one-to-one mapping from elements of the first group onto elements of the second group $\{(p_2, p_2), (p_3, p_3), (p_4, p_4), (p_5, p_1)\}$. For all pairs, the left element is either coincident to or dominates the right element; moreover, the last pair is not coincident. Neither group can be compared to $\{p_2, p_3, p_4\}$ nor $\{p_1, p_2, p_3, p_4, p_5\}$.



An alternative approach is to aggregate each attribute independently, then apply Definition 2.3 to the aggregate points.

Definition 2.7 (Skyline Groups [22, 25]). Consider two groups G and G' of points in $\mathbb{R}^d$ and function $f : \mathbb{R}^d \to \mathbb{R}$. Let $G[i]$ be

Model	Ordinal	VariableSize	Diverse	Complexity
Avg [22, 25]		✓	✓	$O(dg)$
Permute [49]	✓		✓	$O((\sqrt{g}+d)g^2)$ [49]
Min [29]	✓	✓		$O(dg)$
General	✓	✓	✓	$O(dg^2)$

Table 1: Comparison of group dominance models by key properties and asymptotic complexity

the i'th point in the group and $G[i]^j$ be the j'th attribute of that point. We say that G *dominates* G' denoted $G \succ_{f_i} G'$, iff:

$$\left(f_{i=1}^{|G|}G[i]^1, \dots, f_{i=1}^{|G|}G[i]^d\right) \succ_{f_i} \left(f_{i=1}^{|G'|}G'[i]^1, \dots, f_{i=1}^{|G'|}G'[i]^d\right).$$

Any functions $f_i : \mathbb{R}^d \to \mathbb{R}$ can presumably be used. We differentiate *Avg* and *Min* as distinct models, particularly since *Min* has been used for community discovery in multi-attributed networks [29] and has different trade-offs that we compare soon.

Example 2.8. In Figure 2, group $\{p_2, p_4, p_5\} \succ \{p_6, p_7, p_8, p_9\}$ under *Min*. Each group is mapped to its worst value in each dimension; so, we have $(3, 30) \succ (3, 10)$. The excellent point p_7 does not make any contribution to the second group's score.

Under *Avg*, these groups are instead mapped to $(14/3, 120/3)$ and $(19/4, 190/4)$; thus, the dominance relationship is inverted. The first group is discarded by just $1/12$, however, which is a margin that is smaller than the granularity of the domain of the x attribute.

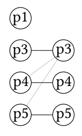
None of these models particularly suit comparing teams in social networks. So, we introduce *General*, which relaxes *Permute* from a one-to-one mapping to a one-to-many mapping.

Definition 2.9 (Team dominance). For groups G and G' in $\mathbb{R}^d$, $G \neq G'$, we say that G *dominates* G', denoted $G \succ_{General} G'$, iff:

$$\forall v \in G' \setminus G, \exists u \in G \setminus G', u \succ v.$$

The multiset difference operator gracefully handles coincident points by removing them in pairs from comparison. Like the prior definitions, this relation is transitive, asymmetric, and irreflexive.

Example 2.10. In Figure 2, group $\{p_1, p_3, p_4, p_5\} \succ \{p_3, p_4, p_5\}$ under the *General* model. After removing common elements, the second group is empty. The model naturally selects the first group. As another example, $g_0 = \{p_6, p_8, p_9\}$ and $g_1 = \{p_1, p_2, p_3, p_4, p_5\}$ are incomparable: no member of the first group is better on the x-axis than p_5 and vice versa with respect to the y-axis and p_8. However, by expanding g_0 with p_7, we see that $g_0 \cup \{p_7\} \succ g_1$.



2.3 Comparison of models

Here we compare the models in terms of three properties that are desirable for comparing teams in multi-attributed networks.

- **Ordinal** Many graphs contain categorical attributes that are difficult to map onto cardinal values. Even numeric attributes like price or rating might be better compared in terms of equivalence classes than precise arithmetic as many users are equally satisfied with a 4.7- and a 4.8-rated hotel. *Only Avg cannot support this data type effectively.*
- **VariableSize** For comparability to earlier work [49], we focus algorithmically on fixed-size k-plexes. However, a strong model should support a spectrum of definitions of cohesion, such as comparing different maximal k-plexes. *Only Permute cannot support variable team sizes.*
- **Diverse** Constructing diverse teams means bringing together people with different strengths. A strong model should not be systematically biased against groups whose team members have specialised skills. *Only Min excludes teams based on each member's weakest skill, irrespective of how well it is compensated by the rest of the team.*

As Table 1 summarises, only *General* provides all of these properties. Finally, we note that the aggregation models are the fastest to evaluate. While *Permute* and *General* have comparable worst-case complexities, the latter is a simple, vectorisable nested loop whereas the former requires setting up and solving a bipartite matching.

In Section 6, we perform a larger-scale comparison on a real coauthorship network. Li et al. [25] also compare *Min* and *Avg* on NBA and stock analysis datasets, though neither of these is graph-based and *Permute* and *General* are not included in that comparison.

2.3.1 Blending Models. Observe that three of the models have complementary characteristics that one can blend to create custom, ensemble-like models. *Avg* favours teams with a high average proficiency; *Min* favours teams with high minimum quality; and *General* favours teams with at least one high-performer.

One can compose these models to target those characteristics. For example, by requiring that a group G is not dominated by another group G' on *either Avg or General*, one constructs a more selective output in which groups have average high proficiency *and* at least one high-performer. On the other hand, by requiring that a group G is dominated by *both Min and General*, one less selectively constructs groups with high performers and/or minimum qualities. *Permute* does not blend well because it already returns huge supersets of what *Avg* and *General* return.

2.4 Team discovery problem statement

We can now state the succinct formal problem definition for the algorithms in this paper. We want to find k-plexes of (small) size g that lie on the pareto frontier with respect to *team dominance*.

Problem Definition. Given graph $G = (V, E), L$, integers k and g, $g \geq 2k + 1$, output the non-dominated k-plex teams of size g:

$$\begin{aligned} P_{k,g}(G) &= \{V' \subseteq V : |V'| = g \wedge \min_{H_{V'}} \delta(v) \geq g - k - 1\} \\ SKY(G, k, g) &= \{P \in P_{k,g}(G) \mid \nexists P' \in P_{k,g}(G),\ P' \succ P\}. \end{aligned}$$

Example 2.11. In Figure 2, the size-4 1-plexes are $P_{1,4}(G) = \{\{p_1, p_3, p_4, p_5\}, \{p_1, p_2, p_3, p_5\}, \{p_2, p_3, p_4, p_5\}, \{p_6, p_7, p_8, p_9\}\}$. As the last one dominates others, $SKY(G, 1, 4) = \{\{p_6, p_7, p_8, p_9\}\}$.

3 PARALLEL LISTING OF SMALL k-PLEXES

Listing small k-plexes is a challenging, NP-hard, combinatorial problem. If the input is the complete graph, the size of each k-plex is fixed to g, and $n = |V|$, then there are nCg k-plexes, though we expect far fewer in sparse graphs, especially as g grows.

Great progress has been made on listing maximal k-plexes [4, 5, 10], but simply halting recursion early in those algorithms is naive. Listing small g-cliques can be done quite efficiently [13], but such algorithms rely on the property that every vertex in a clique is a neighbour of every other vertex. This is the precise property relaxed by quasi-cliques and so is not available algorithmically.

We give a novel algorithm, ListKPlexes (Algorithm 1), based on a counter-intuitive cousins-first iteration strategy to reduce asymptotic complexity and an iterative k-core repeeling heuristic.

3.1 Degeneracy-ordered, cousins-first search

Before presenting our algorithm, we first recount some well-known results on clique and quasi-clique listing:

(1) The diameter of a k-plex is ≤ 2 if $g \geq 2k+1$ [41].
(2) If all edges in undirected graph G are oriented from the lower to the higher degeneracy vertex (one in the k-core for higher k), then the maximum out-degree, Δ, of the oriented graph is equal to the degeneracy, β, of G [9]. Usually, $\beta << \Delta$ [17].
(3) Listing g-cliques can be done in $O(gm\beta^{g-2} + m)$ time by recursively expanding cliques and repeatedly intersecting the neighbour lists of all vertices during that expansion. The bound comes from the degeneracy ordering [13].

The following obvious but useful corollaries follow from (1):

Corollary 3.1. *For a k-plex P of size g, if $g \geq 2k+1$, then all vertices $v \in P$ have at most k cousins in P; moreover, they are incident to any vertex $u \in P$ for which u is not a cousin.*

Corollary 3.2. *Given fixed size g, every k-plex P is a subset of the 2-hop neighbourhood of every vertex $v \in P$ if $g \geq 2k+1$.*

Using Corollary 3.2 is not as immediately helpful as it may seem: even with degeneracy ordering, the 2-hop neighbourhood of a vertex is bounded only by $\min(n-1, \Delta\beta)$, not β^2 as one may hope. This is because one cannot obtain the 2-hop neighbourhood just by following outgoing edges. Instead, maximal k-plex listing algorithms will keep track of potentially connected vertices. Thus, halting a maximal k-plex algorithm like that of Conte et al [10] at recursion depth $g-2$ explores $O(m\Delta^{g-2}\beta^{g-2})$ combinations. It also costs $O(g \log \Delta)$ to confirm that each combination is actually a k-plex. This is much worse than the complexity in result (3) above for cliques where one need never visit in-neighbours.

However, we can use Corollaries 3.1–3.2 to make this problem more "clique-like." If we expand groups with up to k cousins first, then we know that the remaining vertices in the k-plex must be neighbours of v, and we can thereafter only follow oriented out-edges with a maximum degree of β. This reduces the search space from $\Delta\beta$ to β for each slot and overall by a multiplicative factor of $O(\Delta^{g-k-1})$.

Cousins-first search is counter-intuitive, because in k-clique listing, one iterates over neighbours in order to constrain search early. But in fact, the principle is the same. If one begins by obviously filtering to a 2-hop neighbourhood, then expanding a k-plex with a neighbour does not add any constraints, because we cannot intersect the neighbour lists and we already knew the cousins. Expanding with a cousin, however, constrains search space by reducing the number of additional cousins that can be added. Thus, it reduces the search space faster.

Given the fact that it is the number of cousins that are limited in a k-plex, the cousins-first search optimises for small groups by reducing the search space as early in the recursion as possible, since any k-plex-discovery algorithm must search cousins.

Algorithm 1 ListKPlexes($G = (V, E)$, k, g)

```
    Precondition: nodes ordered by ↑ coreness
    Precondition: neighbour lists $E[i]$ ordered by ↓ node id
 1: $V \leftarrow$ CoreDecomp(G, $g-k-1$)  ▷ Save bucket queue state
 2: for $v \in V$ in order do
 3:     $C, N \leftarrow$ cousins (or [] if $k$-clique), neighbours of $v$
 4:     if $|C| + |N| \geq g$ then
 5:         PrefixKPlexes([$v$], [0], $C + N, E, k, g$)
 6:     $V \leftarrow$ CoreDecomp($G' = (V \setminus \{v\}, E), g-k-1$)
 7: function PrefixKPlexes($C, M, V, E, k, g$)
 8:     for $v \in V$ in order do
 9:         $N \leftarrow C \cap E[v]$          ▷ Neighbours of $v$ in group $C$
10:         if $|C| - |N| \leq k$ then
11:             if $|C| + 1 = g$ then emit $C + [v]$; continue
12:             $V', M' \leftarrow$ Trim($C \setminus N, V, E, k$)
13:             if $|C| + |V'| + 1 \geq g$ then
14:                 PrefixKPlexes($C + [v], M', V', E, k, g$)
15: function Trim($M, V, E, k$)
16:     for $v \in$ copy($V$) do           ▷ Preserve iterator validity
17:         Increment missed connections count for $v$ in $M$
18:         if M[v] $= k$ then $V \leftarrow V \cap E[v]$
        return $V, M$
```

3.2 Listing k-plexes efficiently

Algorithm 1 integrates cousins-first search with the iterative k-core repeeling heuristic introduced independently in [18] and [4, 5].[2] We first shrink G to the (g-k-1)-core and save the state of the bucket queue BQ for reuse (Line 1). Then, for each vertex $v \in V$ in ascending coreness (Line 2), we recursively enumerate k-plexes in which v is the vertex with lowest id, i.e., lowest coreness (Lines 3–5). Finally, peeling resumes by removing v from BQ and decrementing the degrees of its neighbours.

The search for k-plexes involving v first reduces the input to the remaining 2-hop neighbourhood of v in the (g-k-1)-core, then performs cousins-first search. Consequently, we search at most $O(n\Delta^k\beta^{g-k-1})$ combinations, leading to the complexity below.

Theorem 3.3. *With degeneracy-oriented edges and $g \geq 2k+1$, $P_{k,g}(G)$ can be computed in $O(ng\Delta^{k+1}\beta^{g-1} + m)$ time.*

Proof. For each of the n vertices in V, there are $O(\Delta^k\beta^{g-1})$ combinations of vertices that involve at most k cousins. For each of these combinations, there is $O(g\Delta + g^2 + (g-k-1)\Delta + g)$ work for intersecting sorted lists of groups and non-oriented neighbours on Lines 9 and 12, for intersecting sorted lists of active vertex lists and non-oriented neighbours until the k-plex constraint ensures no other vertices can miss k connections on Line 18, and emitting a k-plex on Line 11. Thus, the main term is $O(ng\Delta^{k+1}\beta^{g-1})$.

Orienting the edges by coreness requires $O(m+n)$ time for peeling and relabelling. By saving the bucket queue state for peeling the max (g-k-1)-core, we have a total cost of $O(m)$ to shrink the graph over all iterations. With lazy updates to remove

[2]This heuristic observes that all k-plexes are contained within the (g-k-1)-core of graph G. One can immediately reduce the input graph G to its (g-k-1)-core before listing k-plexes. Furthermore, one can save the bucket queue state of the peeling algorithm and resume the peeling from that state after each iteration of the listing algorithm. See any of those references or our source code for details.

vertices and reverse sorted neighbour lists, we can remove edges on Line 6 by popping them from the back of a vector when their source vertex has just been processed. These operations contribute $O(m)$ cost.

All k-plexes are listed because Line 6 only removes vertices that are not part of the maximum (g-k-1)-core or for which we have already listed any k-cores that include them per the sort order on Line 2. No k-plexes are listed twice because every call to function PrefixKCores has a unique prefix per Lines 5 and 14. □

When $k = 0$, Algorithm 1 reduces to the $O(gm\beta^{g-2} + m)$ g-clique listing algorithm of Danisch et al. [13], except it also rapidly shrinks the input graph with iterative k-core repeeling. The algorithm exposes coarse-grained parallelism, described next.

3.3 Bulk synchronous parallelisation

We expose coarse-grained parallelism at Line 2 of Algorithm 1, as described in the code snippet below. $u < v$ denotes that u precedes v in the (degeneracy) order that a sequential algorithm visits them:

Code Snippet 1: Parallelisation Strategy

```
for each batch V' of V in order do
    for each vertex v in V' in parallel do
       process v (Lines 3-5), ignoring all u < v
    # thread barrier
    V  CoreDecomp( G' = ( V \ V' ), E ), g - k - 1)
    # thread barrier
```

Each vertex is a parallel task and the PrefixKPlexes kernel is entirely thread-local. The processing order within a batch is preserved logically (rather than physically) with the check that any vertex u processed in task v is ignored if $u < v$. Lemma 3.4 below implies that the batch pruning of the input graph to an ever smaller (g-k-1)-core is equivalent to having peeled it one-at-a-time, as in [4, 5].

Lemma 3.4. *Let u, v be vertices in a graph G and let $G \setminus \{v\}$ be shorthand for inducing a subgraph of G on $V \setminus \{v\}$. Then, the following are all the same graph: (a) the k-core of $G \setminus \{u, v\}$; (b) the k-core of $H \setminus \{u\}$, where H is the k-core of $G \setminus \{v\}$; and (c) the k-core of $H \setminus \{v\}$, where H is the k-core of $G \setminus \{u\}$.*

Proof. Assume that there exists an order for removing these vertices in which the degree of some vertex that would have been peeled never falls below k. Then the standard peeling algorithm produces a smaller k-core than this new order, contradicting the algorithm's correctness.

Assume that these exists an order for removing a vertex i with degree δ_i prematurely from the k-core solution. Then it must have $\geq \delta_i - k$ links outside the k-core and $< k$ links to vertices therein, again contradicting the correctness of the standard peeling algorithm. □

The barrier synchronisation is helpful, because the work per task is increased (non-asymptotically) relative to single-core execution. Namely, the (g-k-1)-core is only repeeled once per batch rather than once per vertex. On the other hand, parallel k-core decomposition algorithms [14, 16, 23, 35, 36] can be used. As these algorithms can struggle to expose sufficient task parallelism, batch repeeling could accelerate them significantly. This produces the following result:

Theorem 3.5. *With p processors, Algorithm 1 lists all size-g k-plexes in $O(\frac{n}{p} g\Delta^{k+1}\beta^{g-1} + m)$ bulk synchronous parallel time.*

Proof. The algorithm requires n/p supersteps (batches), each performing no more work asymptotically than does the first vertex during sequential execution. Communication between processors and the cost of synchronisation is limited to updates to the (g-k-1)-core, of which there can be at most m across all supersteps. □

4 PARALLEL LISTING OF SKYLINE k-PLEXES

In this section, we study how to rapidly discover the *best* k-plex teams of size $g \geq 2k + 1$ under *team dominance* (Definition 2.9).

A straight-forward solution would be to invoke Algorithm 1 to identify all size-g k-plexes, then run a skyline algorithm over that result. However, this involves materialising a very large intermediate result that, per our experiment results in Figure 6b, can be many orders of magnitude larger than the final result. Moreover, the skyline operator is very expensive, super-linear in the input size. Thus, we must push the skyline operator into the listing algorithm.

4.1 Challenges blending skyline & k-plex listing

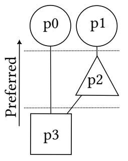
To blend skyline computation into our k-plex listing algorithm, we first recount some key, useful, prior results on skyline computation.

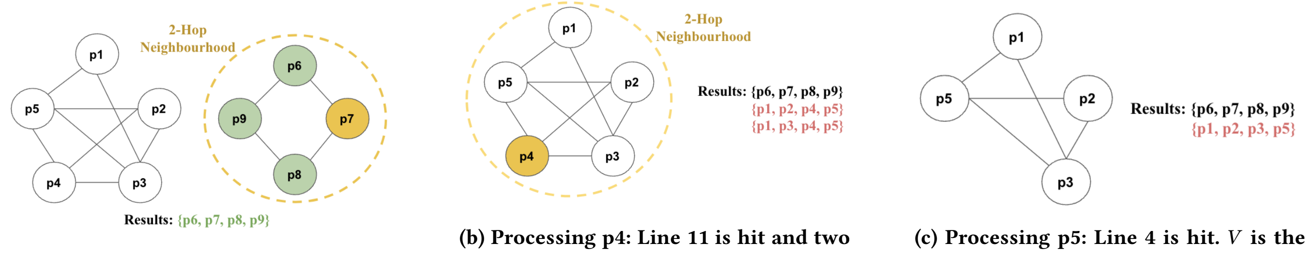
(1) As naive skyline computation is quadratic, one should first sort all points by a monotonic criterion (e.g., L_1 norm) so that the best points are encountered early and quickly dominate most other points which are compared to them first [8].
(2) For the group-based skyline, skyline layers (defined below) are an effective monotonic sort criterion [33].
(3) The order in (1) can be maintained and exploited by parallel algorithms if they process points in batches [7, 39], even for group-based skylines [54].
(4) If some skyline group's Min representative point is better than the Max representative point of an entire skyline layer, that skyline layer can be skipped completely [49].

However, we note a few challenges and incongruences. First, re: result (1), ListKPlexes already defines an iteration order based on degeneracy. As this is orthogonal to vertex labels, it clearly is not monotonic with respect to skyline dominance. We can still iterate points in a newly defined order, but the (g-k-1)-core may not shrink as quickly and parallel task workload may not be as well balanced.

Second, result (2) *does not apply to graphs* unless one assumes that labels are distinct. We prove this below by counter-example.

Example 4.1. Consider extracting skyline k-plexes from the graph to the left with $k = 0$ and $g = 2$, i.e., edges. p_0 and p_1 are coincident, but distinct points represented by the same shape. Vertices are ordered vertically by equivalence class (specifically, "skyline layer") such that $p_0, p_1 \succ p2 \succ p_3$. Observe that group $g_{12} = \{p_1, p_2\} \succ g_{03} = \{p_0, p_3\}$ under both *Permute* and *General*. Yet, there is nothing at all to differentiate p_0 from p_1 that could define a per-vertex sort that ensures g_{03} is encountered prior to g_{12}. This challenge is unique to the graph setting in which





(a) Processing p7: Line 11 is hit and one candidate is generated (line 12). No other group exists to dominate this group.

(b) Processing p4: Line 11 is hit and two candidates are generated (line 13). Both are discarded as they are dominated by the previously-generated group.

(c) Processing p5: Line 4 is hit. V is the only candidate left. It is dominated by the existing group and discarded. The algorithm terminates after this iteration.

Figure 3: Illustration of Example 4.3 showing node processing (of dataset shown in Figure 2) in algorithm 2 to list k-Plexes and identify skyline ones. The green and red colors indicate new groups and dominated candidates respectively.

Algorithm 2 PSKPlex($G = (V, E, L)$, k, g)

```
    Precondition: nodes ordered by ↑ skyline layer, ↓ labels
    Precondition: neighbour lists $E[i]$ ordered by ↓ node id
 1: $\mathscr{S} \leftarrow \{\}$, terminate ← false
 2: $V \leftarrow$ MaxKCore($(V, E)$, $g$-$k$-1)        ▷ Save bucket queue state
 3: while not terminate do
 4:     if $|V| = g$ and $\nexists S \in \mathscr{S}, S \succ V$ then add $V$ to $\mathscr{S}$
 5:     if $|V| \leq g$ then return $\mathscr{S}$
 6:     assign first $t$ nodes in $V$, $u_1, \ldots, u_t$, to $t$ threads
 7:     if $\exists\, S \in \mathscr{S}$ such that $S^+ \prec L_i^-$ ($L_i$ layer for $u_t$) then
 8:         terminate ← true; break
 9:     // begin thread-local parallel region, $i \in [1, t]$
10:     $N \leftarrow E[u_i]$ and if not $k$-clique also 2hops($V, u_i$)
11:     if $|N| \geq g - 1$ then
12:         if $|N| = g - 1$ then $C_i \leftarrow N \cup \{u_i\}$
13:         else $C_i \leftarrow$ PrefixKPlexes($[u_i]$, $[0]$, $N, E, k, g$)
14:         $C_i \leftarrow \{P \in C_i : \nexists S \in \mathscr{S}, S \succ P\}$
15:     // synchronisation step
16:     $C'_i \leftarrow \{P \in C_i : \forall C_{j<i}, \nexists S \in C_j, S \succ P\}$
17:     // end of parallel region
18:     Append each $C'_i$ to $\mathscr{S}$ in order
19:     $V \leftarrow$ MaxKCore($G' = (V[t:], E)$, $g$-$k$-1)
20: return postprocess($\mathscr{S}$)
```

$g_{01} = \{p_0, p_1\}$ is not a k-plex; for group-based skyline queries without selection predicates (as in [33, 54]) group $g_{01} \succ g_{12}$ and $g_{01} \succ g_{03}$.

4.2 The PSKPlex Algorithm

The broad structure of our algorithm, described in Algorithm 2, is to blend our batch-parallel algorithm for k-plex listing with a batch-parallel algorithm for listing skyline groups [54]. In addition to techniques used in listing (Algorithm 1), we apply prior result (4) to terminate early (Lines 7–8) and compare k-plexes to those previously discovered to be skyline teams (Lines 13, 16, and 18).

The main challenge is to identify an effective access order that ensures correctness. To start, we recall the approach of result (2):

Definition 4.2 (Sky Layers [33]). Given points P in $\mathbb{R}^d$, the first skyline layer is the skyline, $L_0 = SKY(P)$. The i'th skyline layer is the skyline of all points not in the first $i - 1$ layers: $L_i = SKY(P \setminus \bigcup_{j=0}^{i-1} L_j)$.

Clearly a group g with a vertex in layer L_i cannot be dominated under *Permute* or *General* by a group g' whose vertices are all in layers $L_{>i}$. While, accessing points in descending skyline order finds strong groups fast, it is not monotonic per Example 4.1.

We do not believe that a monotonic order for groups can be defined over single vertices when the existence of groups is determined by team cohesion thresholds. Instead, we generate a quasi-monotonic order in which potential false positives are adjacent in the solution. In Example 4.1, we want $\{p_0, p_3\}$ and $\{p_1, p_2\}$ to be adjacent in the output, irrespective of how large the graph may be.

For this, we sort points first by skyline layer, then with a full sort on their labels. Note that the skyline layer is determined by the labels; so, all coincident points (such as p_0 and p_1) will be adjacent. We then orient edges per this ordering. For all false positives, the smallest vertex is coincident to the smallest vertex of the group that dominates it. All groups beginning with coincident vertices will be adjacent in the output and are thus easy to locate.

We conclude with an example to illustrate Algorithm 2.

Example 4.3. We use Algorithm 2 to find skyline 1-plexes of size 4 with one thread, using the input from Figure 2. Figure 3 illustrates the steps. Initially, the (g-k-1)-core, G_2, has all nodes as the minimum degree in G is two. We iterate according to the skyline layers in Figure 2.

First (a), p_7 is accessed and $g_0 = \{p_6, p_7, p_8, p_9\}$ is generated from its 2-hop neighbourhood. This group is not discarded, as no group exists to dominate it. Next, p_7 is removed from G_2, popping it off the back of the neighbour lists of its neighbours. The peeling of G_2 is restarted. The degrees of p_6, p_8, and p_9 all fall below 2; so, they are lazily removed from G_2. Next (b), p_4 is accessed and two groups are generated from its 2-hop neighbourhood: $g_1 = \{p_1, p_2, p_4, p_5\}$ and $g_2 = \{p_1, p_3, p_4, p_5\}$. They are compared to g_0 and discarded as $g_0 \succ g_1$ and $g_0 \succ g_2$. Point p_4 is removed from G_2. Peeling restarts, but is immediately paused again since $\delta_2, \delta_3, \delta_5 \geq 2$. Now (c), G_2 includes only 4 nodes, the group $g_3 = \{p_1, p_2, p_3 p_5\}$. It is discarded because $g_0 \succ g_3$. The algorithm terminates, as no new group can be generated.

5 EMPIRICAL VALIDATION

5.1 Baseline Adaptation

We evaluate the computational performance of PSKPlex (Algorithm 2) relative to Baseline, an improvement to the state-of-the-artskyline g-clique algorithm of Zhang et al. [49]. However, [49]

Algorithm 3 BASELINE (a streamlined, parallelised version of [49])

```
    Input: skyline layers of graph $G = (V, E, L)$, size $g$
    Output: set of skyline $g$-cliques
 1: $\mathscr{S} \leftarrow \emptyset$                    ▷ Initialise empty skyline
 2: semaphore ⟵ 0
 3: for all layers $L$ do
 4:     if $\exists S \in \mathscr{S}$ such that $S^- \succ L^+$ then return $\mathscr{S}$
 5:     for all node $u \in L$ in parallel do
 6:         if $|\Delta(u)| \geq g - 1$ then
 7:             $M \leftarrow$ induced subgraph on $\Delta(u)$
 8:             cliques_with_u ⟵ $Cliques(M, g-1, \{u\})$
 9:             wait until semaphore = u
10:             for all $C \in$ cliques_with_u in order do
11:                 if $\exists S \in \mathscr{S} \cup$ cliques_with_u, DOM(S, C) then
12:                     remove $C$ from cliques_with_u
13:                 else
14:                     $\mathscr{S} \leftarrow \mathscr{S} \cup C$
15:             memory fence
16:         wait until semaphore = u
17:         semaphore ⟵ semaphore + 1
18: return postprocess($\mathscr{S}$)
19: function DOM($C'$, $C$)
20:     if $C'^- \succ C^+$ then return true
21:     return $C' \succ C$
```

has been designed instead for the *Permute* dominance model of [33], cannot handle k-plexes, assumes non-coincident points, and lacks parallelism. We adapt it in Algorithm 3 to be more competitive in this evaluation, though it still inherently cannot support k-plexes.

Following the notation of [49], $\Delta(u)$ denotes the neighbours of node u that have not been visited earlier in the algorithm. Method *Cliques* on line 8 retrieves all cliques of a certain size from a given graph with the method of Danisch et al. [13]. The DOM method (Lines 19-21) performs dominance tests per [49], excluding pruning rules that are not constructive in the *General* dominance model.

Dominance checking in *Permute* is expensive and [49] is heavily optimised to reduce those tests. Their work improves the complexity of dominance tests (c.f., Table 1), but it still involves setting up and solving a maximum bipartite matching problem. By contrast, *Avg* and *Min* traverse elements in groups linearly to compute representatives and *General* involves a short and simple quadratic loop to compare the g elements of each group. So, the original algorithm of [49] introduces optimisations to avoid dominance checks. For example, they store the aggregate maximum and minimum of each group in an R-tree to permit early termination of dominance tests (Line 15). These are not very effective (we observe them to fail more than 99% of the time), but we keep them because their overhead is minimal. BASELINE also stores the aggregate maximum of each skyline layer which can help terminate the algorithm early on Line 4.

Other optimisations in [49] introduce overhead. For example, on Line 6, they calculate maximal cliques on each iteration. This only pays off when using expensive *Permute* dominance tests; we streamline it with a fast-to-compute induced subgraph. We use the same order and postprocessing as Algorithm 2 so that BASELINE can handle coincident points. We observe BASELINE to execute faster than reported in [49], demonstrating a good

Dataset	$\|V\| = n$	$\|E\| = m$	δ_{avg}	Δ	Density
Eucore (EU)	986	16064	33	345	.033047
WikiVote (WV)	7115	100762	18	1065	.003981
Enron (EN)	36692	183831	10	1383	.000273
DBLP (DB)	317,080	1,049,866	7	344	.000021
Amazon (AM)	735,323	3,523,472	10	1,077	.000013
YouTube (YT)	1,134,890	2,987,624	5	28,754	.000005
WikiTalk (WT)	2,394,385	4,659,565	4	100,029	.000002
CitPatent (CP)	3,774,768	16,517,947	9	793	.000002
LiveJournal (LJ)	4,846,609	42,851,237	18	20,333	.000004

Table 2: Statistics of datasets used in Section 5

balance between including and excluding optimisations for the *General* model.

For parallelisation, we introduce a counting semaphore to ensure that a thread cannot enter the critical region involving the shared, mutable global solution, $\mathscr{S}$, until $\mathscr{S}$ has been updated with the results of all preceding nodes. (This is required for correctness.) The cost of this critical region is low relative to clique listing, because, as the following experiments will show, $|\mathscr{S}|$ is typically small.

5.2 Experiment Setup

Datasets and parameters We use nine real datasets from SNAP[3] that are commonly used for evaluating community search algorithms and that range in size from m = 16K edges (EU) to m = 42M edges (LJ) and density from ρ =2E-6 (CP) to ρ =3.3E-2 (EU). For each graph, we replace directed edges with de-duplicated, undirected edges and delete self-loops. Table 2 shows describes their properties. YT is the largest dataset in the experiments of [49].

We generate d-dimensional vertex labels in [0,100) with correlated, independent, and anticorrelated distributions per [2]. Experiments vary the number of vertex attributes (d = 2 as default); the correlation between attribute values (attribute distribution, indep as default); the community size (g = 3 as default); cohesiveness (k = 0 as default) and concurrency (t = 48 as default). As preprocessing is common between BASELINE and PSKPLEX, we measure time from the moment the graph edges are re-oriented until the final set of skyline communities is returned as arrays of vertex ids.

Environment Experiments are conducted on a cloud that provisions *2× Intel Platinum 8160F Skylake @ 2.1GHz* and *2× Intel Platinum 8260 Cascade Lake @ 2.4GHz* whole nodes with 187GB RAM and running CentOS 7. All algorithms are implemented in C++; OpenMP is used for multithreading. If the application is out of memory or it cannot finish in 12 hours, then processing time is reported as INF (numerically 42000 seconds in the plots).

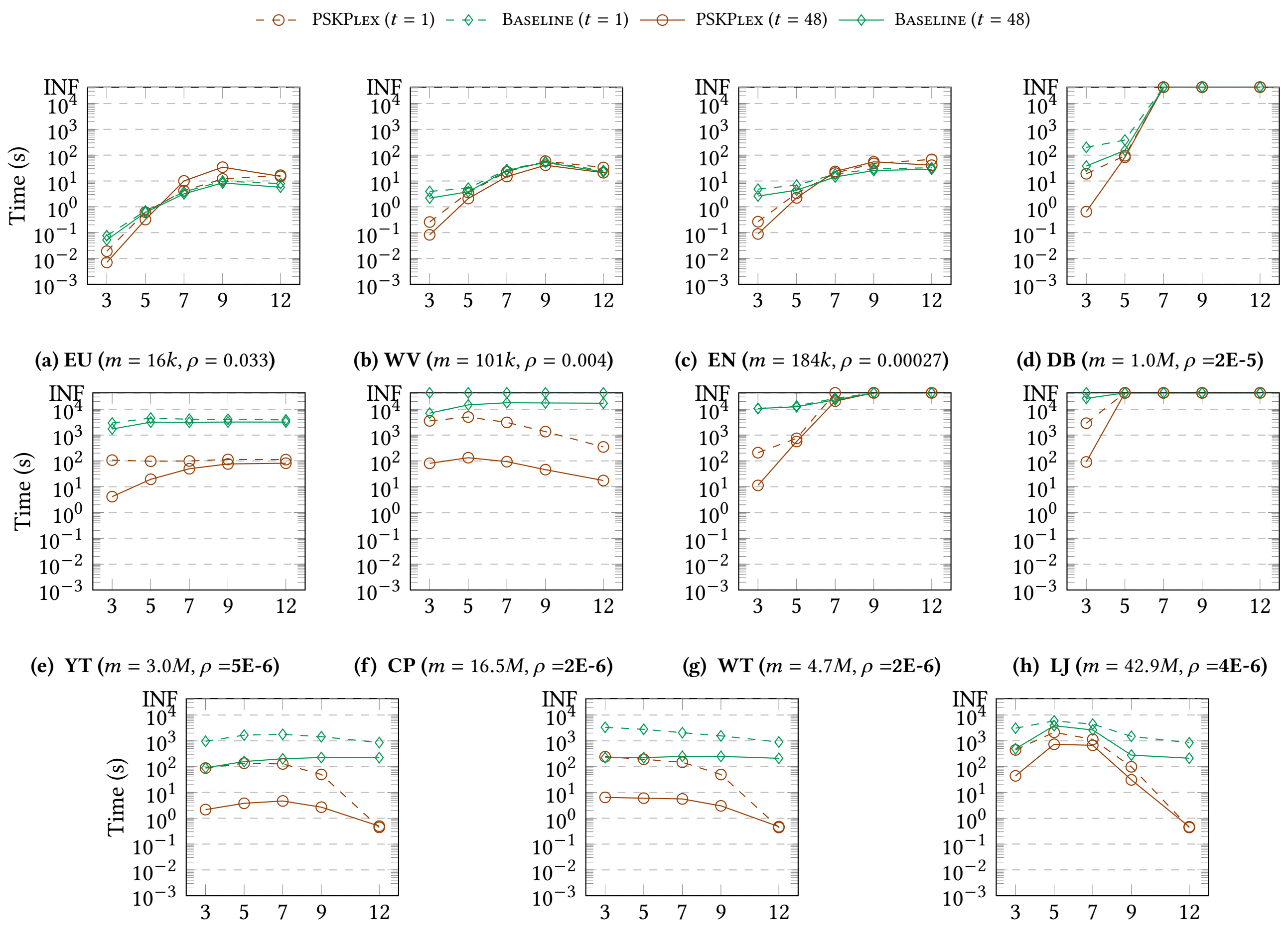
The source code for this work is available at https://github.com/parisaes/skyline-kplex.

5.3 Comparisons to State-of-the-Art (Cliques)

In these initial experiments, we set k = 0 (i.e., search for skyline g-cliques rather than k-plexes) to enable comparisons to BASELINE.

Group Size (g) To begin, Figure 4 studies how team (or clique) size affects the algorithms on all datasets. On the small, dense

[3]http://snap.stanford.edu/



(a) EU (m = 16k, ρ = 0.033) **(b) WV** (m = 101k, ρ = 0.004) **(c) EN** (m = 184k, ρ = 0.00027) **(d) DB** (m = 1.0M, ρ =**2E-5**)

(e) YT (m = 3.0M, ρ =**5E-6**) **(f) CP** (m = 16.5M, ρ =**2E-6**) **(g) WT** (m = 4.7M, ρ =**2E-6**) **(h) LJ** (m = 42.9M, ρ =**4E-6**)

(i) AM (m = 3.5M, ρ =**1E-5**) **(j) AM (correlated)** **(k) AM (anticorrelated)**

Figure 4: Group size effect (varying g on x-axis)

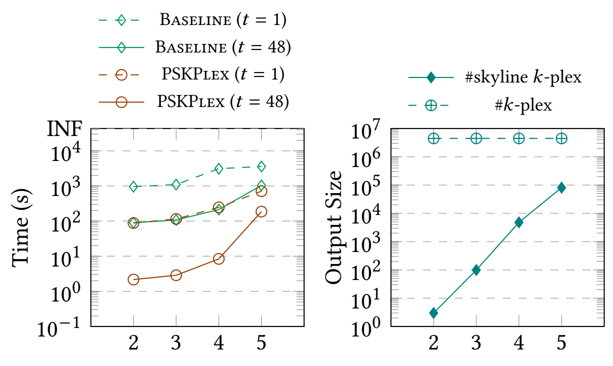
datasets, the difference between PSKPlex and Baseline is minimal; except, of course, that PSKPlex can handle more general cases. Both algorithms complete within three minutes, irrespective of group size. Execution slows up to g = 9 as the search space for groups expands combinatorially, before accelerating as fewer combinations of vertices meet the clique cohesiveness threshold.

For the mid-size graphs (DB, AM, YT, WT), PSKPlex enjoys a marked improvement of 1.5–3+ orders of magnitude, corresponding to fewer hours of waiting, except when it runs out of memory on WT at $g \geq 7$. This is very impressive, considering that Baseline is specialised only to this case of g-cliques. On large graphs (CP, LJ), we observe the same performance gap, except that memory consumption catches up to PSKPlex sooner on LJ.

We study parallelism in more depth in Section 5.4, but a few points can be observed already. The baseline obtains a marginal improvement from parallelism, but PSKPlex can see gains up to 44.1× on 48 cores as the graph grows (and becomes sparser), despite having already been substantially faster on one thread. Also, as g increases, single-threaded and parallel execution converge. This is not related to listing cliques (which we show later improves with increasing g), but to the synchronisation needed for skyline calculation as thread workload becomes more imbalanced. Still, even in the pathological case of g = 12, multithreading offers improvements.

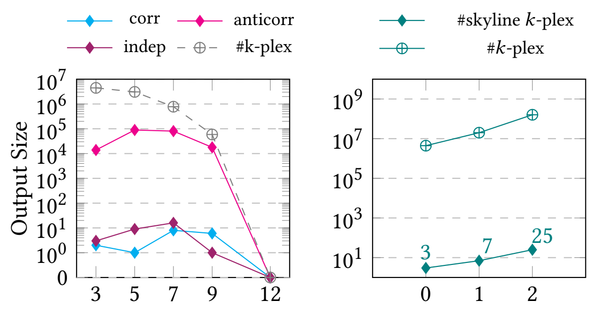
Attribute Distribution Figures 4(j) and 4(k) vary the correlation among the attributes as this is well known to affect skyline performance [2]. We use the AM dataset, where Baseline sees the best parallel acceleration. Although anticorrelated distributions are slower, we see the same broad-based trends: sequentially, PSKPlex is at least 2.9× faster than Baseline, and multicore parallelism increases that gap to at least 6.0×. Execution slows to a peak g before accelerating again. There are zero cliques at g = 12, so PSKPlex terminates very quickly and does not need multithreading. Our listing algorithm detects this case quickly, thanks to iterative k-core repeeling, but Baseline cannot, leading to the exaggerated 2026×.

Dimension (d) Figure 5 repeats the last remaining experiment from [49], in which we vary d. As in all cases prior, we observe an order of magnitude speedup for sequential PSKPlex relative to Baseline, and up to an additional 40.6× from multi-threading. As is usually the case with skylines, an increase in the number of dimensions leads to an increase in the output size. As the number of comparisons per group grows, the work per thread and also the execution time go up as well. This affects PSKPlex at t = 48 more as it increases the time between synchronization barriers: a slow thread takes proportionately longer than it would with



(a) Processing Time (b) Output Size

Figure 5: Dimension effect on AM (varying d on x-axis)



(a) Varying g on the x-axis **(b) Varying k on the x-axis**

Figure 6: Output size, AM

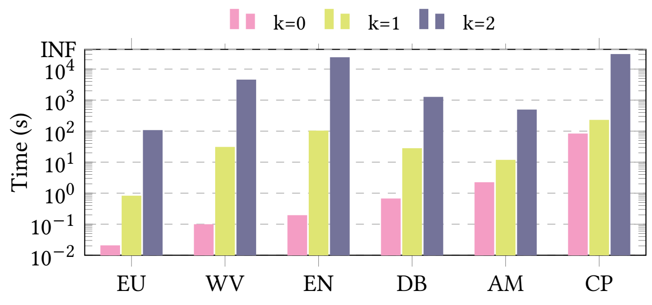


Figure 7: Cohesiveness (PSKPLEX, $g - k = 3$, $d = 2$, $t = 48$).

Dataset	PSKPLEX				BASELINE			
	t=4	t=12	t=24	t=48	t=4	t=12	t=24	t=48
EU	1.33	2.22	2.5	1	1.21	1.46	1.29	1.46
WV	1.28	1.91	2.41	2.89	1.12	1.42	1.65	1.82
EN	1.67	2.3	2.85	1.47	1.18	1.41	1.81	1.86
DB	3.58	9.79	17.94	29.88	1.65	3.04	4.02	5.31
AM	3.78	11.1	21.63	40.61	2.24	5.03	7.44	10.64
YT	3.43	9.05	15.99	25.98	1.2	1.4	1.55	1.71
WT	3.26	8.02	12.75	18.62	1.03	0.93	0.97	1
CP	3.84	11.34	22.42	44.12	-	-	-	-
LJ	3.49	9.35	16.3	26.55	-	-	-	-

Table 3: Parallel speed-up relative to $t = 1$ ($k = 0$, $d = 2$, $g = 3$)

smaller d. The curves for $t = 1$ and $t = 48$ are thus converging in the limit, though the gap to the baseline is maintained.

At $d = 5$, 1.8% of the k-plexes in the graph are skyline groups. Thus, it is advisable to perform skyline queries on the projection of the few most important attributes. Still with 48 cores, we observe a speed-up of 5.6× over BASELINE even for this more severe case.

5.4 Expanded Analysis

The next experiments continue the scalability analysis, but with respect to parameters that could not be studied in [49].

Parallel scalability (t) Our parallelised PSKPLEX makes a lot of problem instances practical that otherwise would not be. Table 3 reports the ratio of running PSKPLEX with 1 thread relative to t threads for all datasets. On 48 cores, we achieve between 18.6× (WT) and 44.1× (CP) parallel speed-up for mid- and large-size graphs. In the case of LJ (the longest-running dataset at 48 cores), our multithreading reduces time by 26.6× from 42 minutes on one core to just 1.5 minutes.

This strong parallel scalability comes from how we expose coarse-grained parallelism. As we progress through the list of nodes, the processing time per iteration decreases somewhat evenly as the iterative k-core repeeling dramatically shrinks the input (c.f., Figure 8). Moreover, the repeeling could be done with parallel k-core decomposition algorithms using all threads. This contrasts to BASELINE, which generally does not have a mechanism to globally shrink the data. Instead, each thread independently computes an induced subgraph, leading to large workload imbalances, L2 cache pollution, and heavy contention on the memory bus.

Clique relaxation (k) This work makes it possible to relax cliques to k-plexes ($k > 0$) and thereby discover new types of skyline communities. Figure 6 shows output sizes and Figure 7 shows PSKPLEX execution times grouped by dataset for 0-, 1-, and 2-plexes, respectively.

Naturally, increasing k requires more time, irrespective of dataset. We expect this because all k-plexes are (k+1)-plexes; so, the search space for skyline communities increases monotonically with k. We expect a pronounced degradation from $k = 0$ to $k = 1$ as we can no longer simply intersect neighbour lists, i.e., when the problem changes from finding cliques to finding k-plexes.

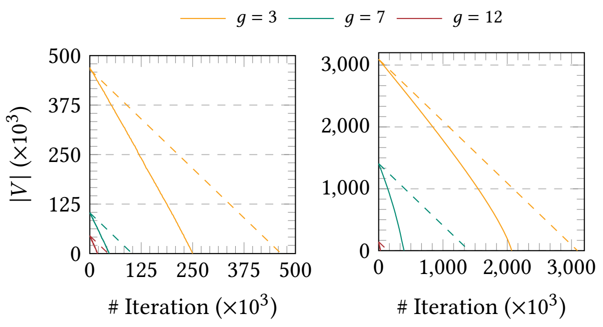
Interestingly, the impact of k is non-uniform. For example, execution time for CP and AM is only ≈ 3× and ≈ 5× slower at $k = 1$ than $k = 0$, respectively, whereas the time for DB increases from 80ms to 12s. However, on CP, there is a big jump from $k = 1$ to $k = 2$. Likely, these results relate to the size of the 2-hop neighbourhoods of vertices. Datasets that ran out of memory at $k = 2$ are not shown.

As k increases from $k = 0$ to $k = 2$, the number of candidate k-plexes rises dramatically from about 10^7 to about 10^9. However, the result size is restricted to just 25 groups, even at $k = 2$. This reflects the better utility of the skyline operator on graphs relative to arbitrary combinations of tuples.

Skyline overhead Table 4 reports execution time for PSKPLEX versus just LISTKPLEXES to determine the most expensive part of the algorithm. We first observe that in both datasets, listing time improves dramatically as g increases. The most interesting data is with respect to YT where at $g = 3$, skylines are 10× faster than just listing, but at $g = 12$, they are 3× slower. When PSKPLEX

	YT		CP	
g	PSKPlex	ListKPlexes	PSKPlex	ListKPlexes
3	107.219	1589.689	3550.084	30314.246
5	97.054	393.198	4995.834	2352.071
7	99.294	132.489	3102.361	823.511
9	113.828	56.369	1361.930	256.636
12	112.119	41.704	347.379	69.431

Table 4: Comparing skyline to listing all k-plexes (t = 1)



(a) YT **(b) CP**

Figure 8: Input size reduction due to iterative repeeling

is faster than ListKPlexes, it is clear that pushing the skyline operator into the listing provides tremendous advantage over generating tens of millions of k-plexes and then piping that to a skyline algorithm.

Iterative k-core repeeling efficacy Figure 8 evaluates the rate at which iterative k-core repeeling shrinks the input graph for different group sizes. The y-axis shows the number of vertices that have not yet been peeled as a function of the number of iterations that have been completed. The dashed lines following $y = |V| - x$ illustrate the rate at which the graph size shrinks just by processing vertices. The solid lines, by contrast, show the additional effect of k-core repeeling. We see that, for example, half the iterations on Youtube and one third of the iterations on CitPatent are eliminated. Moreover, for any given iteration, the input size is dramatically reduced.

Summary The proposed algorithm typically provides 1–2 orders of magnitude improvement over the state of the art and then the multicore parallelism provides 20–40× improvement on top of that. The major improvement comes in the k-plex listing, where the parallelism is exposed and where the iterative k-core repeeling can reduce half of the iterations of the algorithm.

6 CASE STUDY

In this experiment, we compare models on arnetminer [43].

Task Description We want to discover the top cohesive research clusters based on prior publications. We label authors by their paper counts at PVLDB and SIGMOD, dropping vertices with all-zero vectors. Edges pair researchers who have co-authored at least five papers together at any venue. This produces a graph that contains 5856 vertices with 2d labels and 4474 undirected, unlabelled edges.

Analysis Case study results are shown in Figure 10 for $g = 6$ researchers, a clique relaxation of $k = 1$, and preferring higher

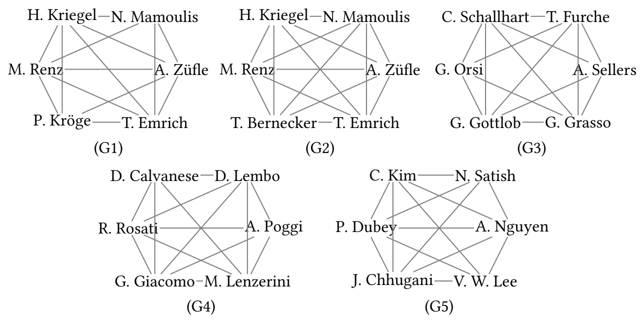
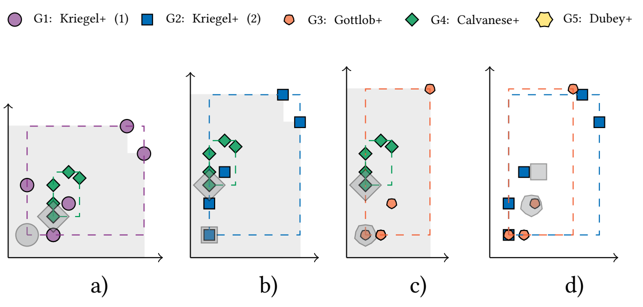
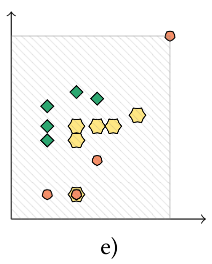


Figure 9: 4-plexes of size 6 in arnetminer





Model	g = 6	$g \in [6, 7]$
Min [29]	G4,G5	G4,G5
Avg	G1,G2	G1,G2
Permute [49]	G1,G2,G3, G4,G5	N/A
General	G1,G2,G3	G1∪G2,G3

f)

Figure 10: Case study contrasting dominance models for 1-plex co-author communities using arnetminer data with VLDB and SIGMOD paper counts as measures of skill.

paper counts as a proxy for research skill. *Avg*, *Min*, *Permute*, and *General* produce 9, 9, 12, and 10 skyline communities, respectively. Seven 1-plexes are common to all models; the middle column of subtable (f) reports how the models differ with respect to the other 1-plexes. Subfigures (a)-(e) illustrate dominance relationships to explain the results. All contain $g = 6$ (possibly coincident) points. Figure 9 shows how researchers in groups are connected.

Min and *General* present the clearest contrast, as the *Min* model excludes groups with members who have specialisations, since it focuses only on representing group strength per individual members' weaknesses. Consider sub-figure (a) which shows the purple group (G1), selected by *General*, and the green group (G4) selected by *Min*. The shaded gray area shows the region dominated by purple points per the *General* model and it is clear that this region contains all of G4. In contrast, the gray markers show dominance in the *Min* model by aggregating entire groups into the weakest value for each dimension. G1 has one member with several VLDB papers but none at SIGMOD and another member has several at SIGMOD but none at VLDB; therefore, the entire group is dominated by G4, where everyone has a few papers at both. Figures (b) and (c) are similar. The *Min* model excludes strong groups if some members specialise on axes. It actively penalises diverse groups.

Avg has similarities to *General* but omits the orange group (G3), illustrated in sub-figure (d), though it contains the top SIGMOD researcher across all k-plex communities. The reductive aggregation of the *Avg* model is unable to capture this type of group. The *Permute* model, lacking much selectivity, returns the union of all other models; sub-figure (e) illustrates that the dominance region of the orange group (G3) contains all members of G4 and G5, as we had already seen for the purple group, G1, in sub-figure (a).

Variable Group Sizes Next, the right column of (f) investigates how the models could perform if we could relax the constraint that g were fixed. As before, we search for 1-plexes, but now g can vary from $g = 6$ to $g = 7$. We omit the *Permute* model because it is limited to comparing groups of the same size and thus cannot be used in this context. The *Avg* and *Min* models return the same groups as before: for *Avg* model, adding more points to strong groups typically lowers the average; for *Min* model, additional points can only decrease the score of a group. The *General* model, by contrast, reports G1 ∪ G2 as well as G3, capturing larger co-authorship communities–if they exist–when given a wider search space.

Summary This case study shows that we can find a small number of dominant communities on real graphs. The previously studied models have clear limitations: *Min* model penalises diversity and *Permute* model requires fixed group sizes. By contrast, our *General* model resolves both weaknesses and finds communities missed by *Avg* model.

7 RELATED WORK

Problem modelling The skyline operator [2] adapts the max vector problem [24] for relational databases. Since this operator is very expensive, selective operators, e.g., selection (σ) [6, 31, 37, 38] and projection (π) [44], or both [15] are often pushed through.

Notably, Papadias et al. [38] compose the skyline and group by (γ) operators, which they call *group-by skyline*. Thereafter, Li et al. [25] and Im and Park [22] proposed *skyline groups*, in which groups of tuples no longer represent an equivalence class. In both cases, skyline occurs logically after grouping. Finally, Liu et al. [33] introduce *g-skylines* to compare groups directly, rather than aggregating them. This is a major shift, because it does not integrate with γ, as it requires groups to the same size, and produces massive output [53]. None of these models were designed with graph databases in mind. Table 1 and Figure 10 compare them extensively to our proposal.

Quasi-clique listing Listing all k-cliques can be done in parallel with $O(km\beta^{k-2} + m)$ work [13] with degeneracy-oriented edges by recursively expanding an i-clique to an (i+1)-clique from those vertices in the intersection of the neighbour lists of *all* of the first i vertices. Pre-filtering a graph to a maximal k-clique [13] or k-core [48] can reduce search space, though the former does not apply to quasi-cliques. [18] and [4, 5] proposed a repeeling technique for listing k-plexes where the input graph is first reduced to a k'-core including all potential k-plexes. The state of the peeling algorithm is preserved allowing it to be resumed after processing each node. [21] propose an index to find the subgraph of graph G that contains the highest concentration of cliques. [27] proposed orienting edges with a greedy colouring instead of degeneracy and [45] introduced an edge-centric branch-and-bound strategy. [48] accelerated set intersection for k-clique listing with vectorisation.

Listing k-plexes is more difficult because some vertices may not be incident and thus intersecting neighbour lists will miss results. [41] introduces the small-diameter property when $g \geq 2k + 1$, which [10] uses to constrain search for k-plexes to the 2-hop neighbourhood of each vertex [10]. More recently, branching techniques with heuristic pivots [52], branch-and-bound computations [12] and merging nodes [34] have brought time complexity on scale-free graphs down to an "almost linear scalability" with respect to n. These results are for discovering maximal k-plexes, not listing k-plexes of a specific size, and so miss out on optimisations provided by our corollaries that relate subgraph size to the k-plex property.

Identifying skyline communities Li et al. [30] initiate this line of work by studying the univariate case of which communities have the greatest "influence" using *Min* model. This was extended by Zhou et al. [51] to heterogeneous information networks (HINs). Peng et al. [40] consider a top-k version. Later, Li et al. extended this model to multi-attributed networks [29]. Zhang et al. [49] improved the dominance model to *Permute*, but then focused on fixed-size k-cliques, because of the model's limitation. The special case of bipartite graphs with *Min* [50] has been considered. Li et al. [26] compute skyline groups in road networks based on coreness properties, but there are no attribute values. For a survey of community search techniques, refer to the survey by Fang et al. [19]. Our work is the first to generalise [49] to the difficult case of k-plexes.

8 CONCLUSION

In this work, we introduced a generalised model for comparing attributed sub-communities in social networks and demonstrated with a case study that it is more effective at assembling variable-sized teams of specialists with diverse skills. We also introduced an algorithm for listing fixed-size k-plex communities in a graph, based on cousins-first search, and integrated it with team dominance to produce the first algorithm that discovers dominant quasi-cliques in large graphs. It is also up to 950.9× faster at cliques than our parallelisation of the prior art.

There are plenty of opportunities to extend this work. We applied our flexible model to fixed-size groups, but it could be used for any types of communities. We listed (skyline) k-plexes on large graphs on a multi-core machine, but one could adapt our BSP algorithm to a distributed context to process larger graphs or larger values of k. GPU acceleration has been successfully applied to the skyline operator [3] and may be applicable to skyline groups. Finally, one could explore the blending of dominance models for greater individualisation of results. Code for this work is available online: https://github.com/parisaes/skyline-kplex.

ACKNOWLEDGMENTS

We thank M. Ali Akber for early contributions to this work. This research was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada through the Discovery Grants program, grant #RGPIN-2020-06639. Compute resources were provided by the Digital Research Alliance of Canada.

REFERENCES

[1] Mei Bai, Yuting Tan, Xite Wang, Bin Zhu, and Guanyu Li. 2021. Optimized Algorithm for Skyline Community Discovery in Multi-Valued Networks. *IEEE Access* 9 (2021), 37574–37589. https://doi.org/10.1109/ACCESS.2021.3063317

[2] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline operator. In *The 17th International Conference on Data Engineering*. 421–430.

[3] Kenneth S Bøgh, Sean Chester, and Ira Assent. 2015. Work-Efficient Parallel Skyline Computation for the GPU. *PVLDB* 8, 9 (2015), 962–973.

[4] Lijun Chang, Mouyi Xu, and Darren Strash. 2022. Efficient maximum k-plex computation over large sparse graphs. *Proc. VLDB Endow.* 16, 2 (Oct. 2022), 127139. https://doi.org/10.14778/3565816.3565817

[5] Lijun Chang and Kai Yao. 2024. Maximum k-Plex Computation: Theory and Practice. *Proceedings of the ACM on Management of Data* 2 (03 2024), 1–26. https://doi.org/10.1145/3639318

[6] Sean Chester, Michael Lind Mortensen, and Ira Assent. 2014. On the Suitability of Skyline Queries for Data Exploration. In *Proc. ExploreDB Workshop*. 161–166.

[7] Sean Chester, Darius Šidlauskas, Ira Assent, and Kenneth Sejdenfaden Bøgh. 2015. Scalable parallelization of skyline computation for multi-core processors. In *IEEE 31st International Conference on Data Engineering*. 1083–1094.

[8] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with Presorting. In *19th International Conference on Data Engineering*. 717–719.

[9] Marek Chrobak and David Eppstein. 1991. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science* 86, 2 (1991), 243–266.

[10] Alessio Conte, Tiziano De Matteis, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 2018. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1272–1281.

[11] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large grahps. In *ACM SIGMOD International Conference on Management of Data*. 991–1002.

[12] Qiangqiang Dai, Rong-Hua Li, Hongchao Qin, Meihao Liao, and Guoren Wang. 2022. Scaling Up Maximal k-plex Enumeration. In *CIKM*. 345–354.

[13] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing K-Cliques in Sparse Real-World Graphs*. In *Proceedings of the 2018 World Wide Web Conference*. 589–598.

[14] Naga Shailaja Dasari, Ranjan Desh, and M. Zubair. 2014. ParK: An efficient algorithm for k-core decomposition on multicore processors. In *IEEE International Conference on Big Data*. 9–16.

[15] Evangelos Dellis, Akrivi Vlachou, Ilya Vladimirskiy, Bernhard Seeger, and Yannis Theodoridis. 2006. Constrained Subspace Skyline Computation. , 415–424 pages.

[16] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. 2021. Theoretically Efficient Parallel Graph Algorithms Can Be Fast and Scalable. *ACM Trans. Parallel Comput.* 8, 1, Article 4 (2021), 70 pages.

[17] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing All Maximal Cliques in Large Sparse Real-World Graphs. *ACM J. Exp. Algorithmics* 18, Article 3.1 (nov 2013), 21 pages. https://doi.org/10.1145/2543629

[18] Parisa Esmaeilian Ghahroudi. 2023. *Parallel Discovery of Fixed-sized Connected k-Core Skyline Communities*. Master's thesis. University of Victoria.

[19] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *VLDB J.* 29, 1 (2020), 353–392. https://doi.org/10.1007/S00778-019-00556-X

[20] Fei Hao, Jie Gao, Jianrui Chen, Aziz Nasridinov, and Geyong Min. 2022. Skyline (λ, k)-Cliques Identification From Fuzzy Attributed Social Networks. *IEEE Transactions on Computational Social Systems* 9, 4 (2022), 1075–1086. https://doi.org/10.1109/TCSS.2021.3101152

[21] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2023. Scaling Up k-Clique Densest Subgraph Detection. *Proc. ACM Manag. Data* 1, 1, Article 69 (May 2023), 26 pages. https://doi.org/10.1145/3588923

[22] Hyeonseung Im and Sungwoo Park. 2012. Group skyline computation. *Information Sciences* 188 (2012), 151–169.

[23] Humayun Kabir and Kamesh Madduri. 2017. Parallel k-Core Decomposition on Multicore Platforms. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1482–1491.

[24] H T Kung, Fabrizio L Luccio, and Franco P Preparata. 1975. On Finding the Maxima of a Set of Vectors. *Journal of the ACM* 22, 4 (1975), 469–476.

[25] Chengkai Li, Nan Zhang, Naeemul Hassan, Sundaresan Rajasekaran, and Gautam Das. 2012. On skyline groups. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 2119–2123.

[26] Qiyan Li, Yuanyuan Zhu, and Jeffrey Xu Yu. 2020. Skyline Cohesive Group Queries in Large Road-social Networks. In *IEEE 36th International Conference on Data Engineering (ICDE)*. 397–408.

[27] Rong-Hua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering heuristics for k-clique listing. *Proc. VLDB Endow.* 13, 12 (jul 2020), 25362548. https://doi.org/10.14778/3407790.3407843

[28] Rong-Hua Li, Lu Qin, Fanghua Ye, Guoren Wang, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2020. Finding skyline communities in multi-valued networks. *The VLDB Journal* 29, 6 (01 Nov 2020), 1407–1432.

[29] Rong-Hua Li, Lu Qin, Fanghua Ye, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. 2018. Skyline Community Search in Multi-Valued Networks. In *Proceedings of the 2018 International Conference on Management of Data*. 457–472.

[30] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential Community Search in Large Networks. *PVLDB* 8, 5 (2015), 509–520.

[31] Ming-Yen Lin, Yueh-Lin Lin, and Sue-Chen Hsueh. 2017. Discovering Group Skylines with Constraints by Early Candidate Pruning. In *International Conference on Advanced Data Mining and Applications*. 49–62.

[32] Boge Liu, Fan Zhang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Efficient Community Search with Size Constraint. In *Proc. ICDE*. 97–108.

[33] Jinfei Liu, Li Xiong, Jian Pei, Jun Luo, and Haoyu Zhang. 2015. Finding Pareto Optimal Groups: Group-Based Skyline. *PVLDB* 8, 13 (2015), 2086–2097.

[34] Shohei Matsugu, Yasuhiro Fujiwara, and Hiroaki Shiokawa. 2023. Uncovering the Largest Community in Social Networks at Scale. In *IJCAI*. 2251–2260.

[35] Amir Mehrafsa, Sean Chester, and Alex Thomo. 2020. Vectorising k-Core Decomposition for GPU Acceleration. In *32nd International Conference on Scientific and Statistical Database Management*. 4.

[36] Alberto Montresor, Francesco De Pellegrini, and Daniele Miorandi. 2013. Distributed k-Core Decomposition. *IEEE Transactions on Parallel and Distributed Systems* 24, 2 (2013), 288–300.

[37] Michael Lind Mortensen, Sean Chester, Ira Assent, and Matteo Magnani. 2015. Efficient caching for constrained skyline queries. In *18th International Conference on Extending Database Technology (EDBT)*.

[38] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progessive Skyline Computation in Database Systems. *ACM Transactions on Database Systems* 30, 1 (2005), 41–82.

[39] Sungwoo Park, Taekyung Kim, Jonghyun Park, Jinha Kim, and Hyeonseung Im. 2009. Parallel Skyline Computation on Multicore Architectures. In *IEEE 25th International Conference on Data Engineering (ICDE)*. 760–771.

[40] You Peng, Song Bian, Rui Li, Sibo Wang, and Jeffrey Xu Yu. 2022. Finding Top-r Influential Communities under Aggregation Functions. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1941–1954. https://doi.org/10.1109/ICDE53745.2022.00191

[41] Stephen B Seidman and Brian L Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology* 6, 1 (1978), 139–154.

[42] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *ACM, SIGKDD International Conference on Knowledge Discovery and Data Mining*. 939–948.

[43] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.

[44] Yufei Tao, Xiaokui Xiao, and Jian Pei. 2006. SUBSKY: Efficient Computation of Skylines in Subspaces. In *22nd International Conference on Data Engineering (ICDE)*.

[45] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2024. Efficient k-Clique Listing: An Edge-Oriented Branching Strategy. *Proc. ACM Manag. Data* 2, 1, Article 7 (mar 2024), 26 pages. https://doi.org/10.1145/3639262

[46] Xiaoqin Xie, Chiming Liu, Jiaming Zhang, and Jiahui Li. 2019. Influential Attribute Community Search. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2630–2636. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00369

[47] Kai Yao and Lijun Chang. 2021. Efficient Size-Bounded Community Search over Large Networks. *PVLDB* 14, 8 (2021), 1441–1453.

[48] Zhirong Yuan, You Peng, Peng Cheng, Li Han, Xuemin Lin, Lei Chen, and Wenjie Zhang. 2022. Efficient $k-$ clique Listing with Set Intersection Speedup. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 1955–1968. https://doi.org/10.1109/ICDE53745.2022.00192

[49] Chen Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, Fan Zhang, and Xuemin Lin. 2019. Selecting the Optimal Groups: Efficiently Computing Skyline k-Cliques. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1211–1220.

[50] Yuting Zhang, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Pareto-optimal Community Search on Large Bipartite Graphs. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*. 2647–2656.

[51] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 16, 8 (apr 2023), 20472060. https://doi.org/10.14778/3594512.3594532

[52] Yi Zhou, Zhou Xu, Zhenyu Guo, Mingyu Xiao, and Yan Jin. 2020. Enumerating Maximal k-Plexes with Worst-Case Time Guarantee. In *AAAI*. 2442–2449.

[53] Haoyang Zhu, Xiaoyong Li, Qiang Liu, and Hao Zhu. 2019. Computing Skyline Groups: An Experimental Evaluation. *Tsinghua Science and Technology* 24, 2 (2019), 171–182.

[54] Haoyang Zhu, Peidong Zhu, Xiaoyong Li, Qiang Liu, and Peng Xun. 2017. Parallelization of group-based skyline computation for multi-core processors. *Concurrency and Computation: Practice and Experience* 29, 18 (2017), e4195.