Efficient Skyline Community Discovery in Large Networks

by

Mohammad Ali Akber
B.Sc., Khulna University of Engineering & Technology, 2015

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Efficient Skyline Community Discovery in Large Networks

by

Mohammad Ali Akber
B.Sc., Khulna University of Engineering & Technology, 2015

Supervisory Committee

Dr. Alex Thomo, Supervisor
(Department of Computer Science)

Dr. Sean Chester, Co-Supervisor
(Department of Computer Science)

ABSTRACT

Every entity in the real world can be described uniquely by it's attributes. It is possible to rank similar entities based on these attributes, i.e. a professor can be ranked by his/her number of publications, citations etc. A community is formed by a group of connected entities. Individual ranking of an entity plays an important role in the quality of a community. Skyline community in a network represents the highest ranked communities in the network. But how do we define this ranking? Ranking system in some model considers only a single attribute [16], whereas the other [15] [23] considers multiple attributes. Intuitively multiple attributes represent a community better and produce good results. We propose a novel community discovery model, which considers multiple attribute when ranking the community and is efficient in terms of computation time and result size. We use a progressive (can produce results gradually without depending on the future processing) algorithm to calculate the community in an order such that a community is guaranteed not to be dominated by those generated after it. And to verify the dominance relationship between two communities, we came up with a range based comparison where the dominance relationship is decided by the set of nodes each group dominates. If domination list of a group is a subset of another group, we say the second group dominates the first. Because a groups domination list contains it's member along with the nodes they dominate. So in the example, the second group dominates every node of the first group.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My wife (Kazi Tabassum Ferdous) and my family, for encouraging me to pursue this degree and supporting me in my lowest moments.

Dr. Alex Thomo, for providing me this opportunity by hiring me as one of your research students and having faith in me.

Dr. Sean Chester, for mentoring, support, encouragement, and patience. I have learnt a lot working with you.

NSERC, for funding me with a Scholarship.

The important thing in life isn't the destination, it's the journey. The challenges we face along the way, the unexpected twists and turns, the disappointments we overcome. So, it's great to focus on the destination, but the most important thing is to be happy — right here, right now.

If the universe isn't happening to you, it's happening for you!

Chapter 1

Introduction

With the proliferation of social networks over the past two decades, community detection in graphs has become a well-studied problem. Recently, a focus has emerged on trying to detect particularly interesting communities based on multivariate vertex labels [15, 23, 24].

An example of this is presented in the fictitious, toy co-author network in Figure 1.1. Here, nodes represent authors with their h-indexes and the number of SIGMOD papers that they have published, while edges indicate that they have co-authored a paper. While it is interesting to see that everyone in the group B,C, and D have co-authored papers with each other, it is even more interesting that C,D, and E have all co-authored papers, given the prominence of author E. We could say that C,D, E is a more interesting community, based on these vertex weights.

Because the nodes are multivariate, a natural approach to comparison is pareto-optimality, i.e., the skyline operator [2]; however, this is generally defined for points, not sets of points or network communities. Where [15] and [23] disagree is in how to adapt the concept of pareto-optimality for communities: [15] uses a simple aggregation (element-wise MIN or MAX) to cast the problem into one of comparing points as a generalisation of their earlier work with univariate node labels [16]; whereas [23] applies previous research on group-based skylines [19]. Moreover, each work targets specific types of communities, namely k-cores and k-cliques, respectively.

We describe and contrast these models in more detail in Chapter 3, but, at a high-level, they are both very pessimistic, leading to large output sizes. Element-wise maximising the worst values in a community, as per [16, 15], is agnostic to the quality of anything but the worst nodes, leading to groups B,C,D and B,C,E in Figure 1.1 being indiscernible because they share the same MIN values. Requiring nodes of one

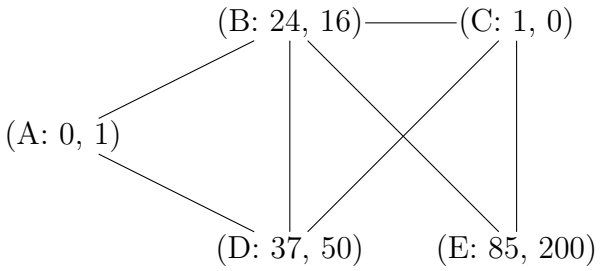


Figure 1.1: An undirected graph with multivariate vertex weights, representing a toy co-authorship network. Each author node is described by the tuple: (id, an h-index, SIGMOD paper count). Existing methods to identify interesting communities (3-cliques/2-cores in this example) fail to discern between BCD and BCE [15] or between ABD and BCE [23].

community to dominate those of another community in one-to-one correspondence, as per [19, 23], leads to huge outputs, as can be inferred by the algorithm's returning all combinations of "unit groups" whose union has size k and, in [23], also form a clique. This leads in Figure 1.1 to A,B,D and B,C,E being indiscernible, since there is no strict dominance relationship between their nearly identical least important nodes.

In this work, we generalise group-based dominance to permit a many-to-one correspondence between non-shared nodes. In direct contrast to the prior works, this gives more emphasis to strong nodes and enables discarding a greater number of communities that are not very different from others that are more interesting. For example, B,C,E will dominate A,B,D because E dominates everything in A,B,D. Moreover, B,C,E will dominate B,C,D because we will consider nodes E and D. Indeed, B,C,E is the only community that we report as interesting in this example, illustrating the greater selectivity.

This generalisation introduces some algorithmic challenges, however. In particular, the test of dominance between two communities must search a much larger space than when only one-to-one correspondences are permitted. We introduce two novel alternatives to accelerate this procedure with on-the-fly indexing techniques, one based on tries and another based on quad-trees.

As a result, we are able to identify a better tuned set of more interesting communities without suffering performance degradation.

1.1 Contributions and Outline

In this paper, we investigate the problem of identifying communities in networks with multi-variate vertex labels. After this introduction and a review of related works (Chapter 3), a review of related algorithms (Chapter 4) and prior to concluding (Chapter 7), we make the following contributions:

- We introduce a more general model of group dominance that reduces the very large output size of existing models (Chapter 2)
- We introduce a novel spatial-indexing-based algorithm to efficiently identify non-dominated skyline cliques (Chapter 5)
- We conduct a thorough set of experiments, including a case study, that demonstrates some of the limitations of prior approaches (Chapter 6)

Chapter 2

PRELIMINARIES

The problem of detecting pareto-optimal cohesive sub-communities in networks is one that leverages background from both graph analytics and skyline computation. This section provides the technical background necessary to understand both these fields. We first describe relevant graph concepts (Section 2.1), then review the skyline operator (Section 2.2), and finally combine the two concepts in presenting our problem definition (Section 2.3). We will compare that problem definition to existing models in Section 3.

2.1 Cohesive sub-communities

In this work, we use simple, undirected graphs with multivariate vertex labels. Let $G = (V, E)$ be one such simple, undirected graph and let function $w : V \longrightarrow \mathbb{R}^d$ map vertices to vectors of d real-valued labels. We use classic array notation to refer to the values of particular dimensions of label vectors, e.g., $w(v)[i]$ refers to the i'th dimension of the label vector of vertex v.

Given a subset $C \subseteq V$, we denote by $G_C = (C, E_C)$ the graph induced by vertex subset C, i.e., $E_C = \{(u, v) : u \in C, v \in C, (u, v) \in E\}$. Moreover, for each vertex $v \in V$, let $\deg_{G_C}(v) = |\{u : (u, v) \in E_C\}|$ denote the degree of v in the induced subgraph, i.e., the number of edges of E_C that contain v.

Cohesive subgraphs are a class of induced subgraph that have a specific highly-connected structure. These generally coincide with communities, due to the propensity for friends of friends ("weak ties") to become friends ("strong ties") [9]. Two related structures that are investigated in related literature are k-cores and k-cliques.

A k-core is an induced subgraph in which everyone has at least k other connections, as formalised below:

Definition 1 (k-Core Community)**.** *A vertex subset $C \subseteq V$ forms a* k-core community *of a graph $G = (V, E)$ iff every vertex in the induced subgraph G_C has a degree of at least k. We denote with the predicate $K_G(C)$ that C is a k-core community:*

$$K_G(C) \equiv \min_{v \in C} \left(\deg_{G_C}(v)\right) \geq k$$

A more restricted subclass of cohesive subgraphs is a k-clique, which is a $(k-1)$-core with exactly k vertices, i.e., a group of k vertices in which every vertex is connected to every other vertex. We formalise this below:

Definition 2 (k-Clique)**.** *A vertex set $C \subseteq V$ is a* k-clique *of a graph G iff $u, v \in C \implies (u, v) \in E$. We use a predicate $\kappa_G(C)$ to indicate whether vertex set C is a k-clique in G.*

We often drop k from the term k-clique, since it is implied by the number of vertices. A clique represents a very strongly connected community in a graph and each graph G may have many cliques. We denote by $\mathcal{C}_k(G) = \{C : |C| = k, C \subseteq V, \kappa_G(C)\}$ the set of all *k-clique communities of size k.*

This very tightly-knit form of community will be the focus of our work. Listing all k-cliques of a graph is a well-studied problem with an efficient solution [6]; however, simply listing k-cliques ignores vertex labels. To incorporate multivariate vertex labels effectively, we introduce the skyline operator next.

2.2 The Skyline Operator

Given univariate, i.e., one-dimensional, data, it is trivial to find the k "best" points: one simply sorts according to a preference for larger or smaller values. Already in two dimensions, this is not possible, because there is no inherent way to sort two-dimensional data; the closest analogue is to project points in the plane onto a one-dimensional subspace and sort them with respect to that. This technique is referred to as a top-k query, well handled by Fagan's Threshold Algorithm [8]. However, it requires specifying precisely a one-dimensional subspace that unequivocally indicates "bestness."

The broad class of alternative methods is based on Vilfredo Pareto's game theoretic notion of pareto optimality, where one strategy is pareto-optimal if there is no other strategy that is superior in at least one facet and not inferior in any. This concept was first studied as a main-memory Computer Science problem in 1975 [13] and then transformed to a secondary storage, database context and renamed as the *skyline operator* in 2001 [2]. Rather than specify a subspace for sorting, one can return all single points that are optimal with respect to *some* subspace. Clearly, irrespective of which subspace most accurately captures "bestness," the optimal solution will be in the skyline. We formalise this concept below:

Definition 3 (Dominance [2])**.** *Given two d-dimensional points $p, q \in \mathbb{R}^d$, in which $p[i]$ denotes the i'th attribute of p, we say that p* dominates *q, denoted $p \succ q$, iff:*

$$p \neq q \wedge \bigwedge_{i=1}^{d} p[i] \geq q[i].$$

For example, given points $p_0 = (1, 2, 3)$, $p_1 = (3, 2, 1)$, $p_2 = (2, 2, 3)$, and $p_3 = (3, 2, 2)$, we observe that $p_2 \succ p_0$ and $p_3 \succ p_1$ but no other ordered pair exhibits a dominance relationship. If for two points p_i and p_j, neither $p_i \succ p_j$ nor $p_j \succ p_i$, p_i and p_j are said to be *incomparable*, denoted commutatively by $p_i \prec\succ p_j$.

Note that we have a well-known property of *transitivity* defined over dominance, as given in Property 1 below:

Property 1 (Transitivity of Dominance)**.** *Let p, q, r be points, $p \succ q$, and $q \succ r$. Then $p \succ r$.*

Finally, the skyline operator returns from a dataset all points that are not dominated by other points in the same dataset. We formalise this below:

Definition 4 (Skyline [2])**.** *Given a set P of points, the* skyline *of P, denoted* $\mathrm{SKY}(P)$*, is the set of non-dominated points:*

$$\{p : p \in P \wedge \nexists\, q \in P, q \succ p\}.$$

There are several methods for generalising Definition 4 from a set of points to a set of sets of points, giving rise to the notion of *group dominance*. We review other models in Section 3, but the novel model that we propose is *generalised group dominance*. Essentially, one group U is dominated by another group U' if all the points of U' not in U are dominated by *some* point in U. We formalise this below:

Definition 5 (Generalised group dominance)**.** *Let $U, U' \subseteq P$ be subsets of a set of points P. We say that U* group-dominates *U' iff: $U \neq U'$ and every point $u' \in U'$ is equal to or dominated by a point $u \in U$. That is to say, $U \succ U'$ iff:*

$$\{u' \in U' \setminus U | \nexists\, u \in U \setminus U', u \succ u'\} = \emptyset \text{ and } U' \setminus U \neq \emptyset.$$

Combining this notion of *generalised group dominance* with k-clique listing on graphs with multivariate vertex labels leads to our problem definition.

2.3 Problem Definition

In this manuscript, like prior work [23, 15], we would like to find all those tightly-knit, cohesive sub-communities that are pareto-optimal with respect to their vertex labels. Unlike those works, we will use *generalised group dominance*, as formalised below:

Problem Definition (Generalised k-Clique Skyline Problem).
Given a graph $G = (V, E)$ with a vertex-labelling function w and an integer k, return the set of non-dominated k-clique sub-communities:

$$k\text{GCSP}(G, w_d, k) \equiv \{U \in \mathcal{C}_k(G) : \nexists\, U' \in \mathcal{C}_k(G), U' \succ U\},$$

where a vertex or set of vertices dominating another vertex or set of vertices is taken to mean dominance with respect to their vertex label vectors.

Chapter 3

A Review and Comparison of Related Models

In this section, we review a chronology of key skyline models that lead to this work. Recognising that the discovery of influential cohesive sub-communities is, in fact, a form of constrained, group-based skyline query, we review first studies of *constrained skylines* (Section 3.1), then *skylines involving groups* (Section 3.2), and finally their combination in pareto-optimal sub-community retrieval (Section 3.3). We conclude in Section 3.4 with an in-depth analysis of the differences of the models presented in Section 3.3.

3.1 Skylines with constraints

When originally adopted for a database context by Börzsönyi et al. [2], pareto optimality was proposed as an extension to SQL queries as a skyline operator, such as:

```
SELECT *
FROM ‘Authors‘
SKYLINE OF ‘HIndex‘ MAX, ‘PaperCount‘ MAX;
```

Naturally, this could be combined with any other query operators—such as selection, projection, and grouping—and takes semantic precedence only before the sorting and limit operators.

Papadias et al. [21] initiated much study into composing the skyline operator

with other standard operators. They named the composition of selection and skyline as a *constrained skyline* and focused on algorithmic techniques for supporting ranges. Mortensen et al. [20, 3] provide a thorough analysis of constrained skylines. Combining skylines with projections has been called a *subspace skyline* [22] and the composition of all three operators has been called a *constrained subspace skyline* [7]. Composition is attractive, because pushing operators like selection and projection through the expensive skyline operator should reduce its working set size. Most of this work is in the context of secondary storage, where composition impacts the indexing strategy; in main memory, it is arguably trivial.

3.2 Skylines over groups

Papadias et al. [21] also consider the composition of skylines and grouping, which they call a *group-by skyline*. Subsequently, Li et al. [14] and Im and Park [11] independently proposed *skyline groups*. These differ from the group-by skyline in that they are not disjoint; whereas a group by operator partitions points into equivalence groups, skyline groups are taken from all combinations of points of a fixed size k. While Im and Park suggest the use of any monotone function for aggregation, the methods proposed in either work specifically focus on MIN, MAX, and SUM.

In group by skylines and skyline groups, a group of points $G = \{p_1, p_2, \ldots, p_k\}$ is cast into a representative virtual point:

$$\hat{p}_G \equiv (f(p_1, \ldots, p_k), \ldots, f(p_1, \ldots, p_k)).$$

Definitions of dominance (Def. 3) and skylines (Def. 4) can then be straightforwardly applied to groups by applying them to their representative virtual points.

Liu et al. [19] later argue that [14, 11] miss some pareto optimal groups. They introduce an alternative, called *g-skylines*, that, rather than cast a group into a point, modifies Def. 3 directly. Specifically:

Definition 6 (Group-based dominance [19])**.** *Given two inequal groups G and G' of d-dimensional points, in which $G[i]$ denotes the i'th point of G, we say that G* dominates *G', denoted $G \succ G'$, iff there exists a permutation $\hat{G}$ of G such that:*

$$\bigwedge_{i=1}^{k} \hat{G}[i] \succeq G'[i].$$

In other words, *group-based dominance* implies that there is a one-to-one mapping from points of $G \setminus G'$ to points of $G' \setminus G$ such that each point of G' is dominated by its corresponding point G.

Liu et al. [19] prove that this definition will lead to a result that is a superset of the skyline groups obtained with aggregation function SUM. Interestingly, Zhu et al. [25] subsequently argue that the output size of g-skylines are prohibitively large as they are lower-bounded by the number of size-k combinations of skyline points. They calculate that for a group size of $k = 5$ with just 1000 skyline points, it is guaranteed to discover at least 10^{13} g-skyline groups; they remark on the importance of reducing this.

3.3 Skyline groups constrained by connectivity

To our knowledge, Lin et al. [17] provide the only work that deliberately composes groups and selection. However, a recent, fascinating application of skylines has done so indirectly: identifying pareto-optimal cohesive sub-communities of networks is equivalent to identifying skyline groups, constrained by network connectivity. For sparse networks, this constaint can be very selective, partly addressing the concern expressed by Zhu et al. [25].

Li et al. [15] provide the first work in this area, identifying k-cores that adhere to the definition of skyline groups under MIN, arguing that this is preferable to MAX and SUM because it avoids outliers. Subsequently, Zhang et al. [23] apply the g-skyline definition and focus on k-cliques. We contrast these in Section 3.4

Finally, Zhang et al. [24] quite recently extended the application of g-skylines to the restricted class of bipartite graphs.

3.4 Comparison of current models

In consideration of the previous subsections, we identify four models for skyline community discovery that merit comparison:

- `PERMUTE`: The g-skyline concept [19], as proposed by Zhang et al [23, 24] and criticised by Zhu et al. [25].
- `SUM`: The skyline groups concept [14, 11], aggregated in the direction of optimisation. This is remarked [19, 25] to produce a subset of PERMUTE.

- MIN: The skyline groups concept [14, 11], aggregated *against* the direction of optimisation, as proposed by Li et al. [16, 15].

- GENERAL: The generalisation of the g-skyline concept [19] that we presented in Definition 5.

Each of these models favours specific group characteristics. We construct a toy example out of these differences in Figure 3.1. We describe below the pertinent characteristic of each group:

- G1: The skyline, which is not connected: {b,c,f}

- G2: Skyline nodes with one bad node (red circle)

- G3: Bad nodes with one skyline node (purple square)

- G4: Every node is slightly better than the expected skyline group's worst node (yellow pentagon)

- G5: If w_i is the worst attribute in the i'th dimension of the expected skyline group, a group where every node is a little better than w_i in the i'th dimension (orange diamond)

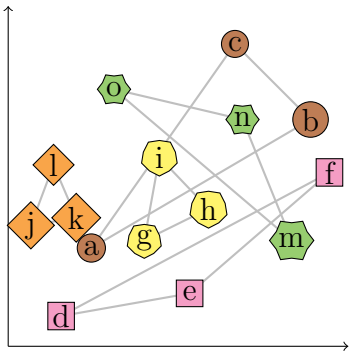
- G6: An average group where some attributes can be really good and some are bad (green bolt)

Some aspects to observe are that every point in groups G4 and G5 are dominated by individual points in groups G3 and G6. Only G2 and G3 contain skyline points. G2 and G5 contain objectively weak points, namely the minimums of the dataset with respect to the x- and y-axes. With that in mind, we reflect below on the idiosyncracies of each model:

GENERAL: outputs G2 and G3. It favours groups with at least one really strong vertex that cannot be dominated. As a result (c.f., G3), it can include weak vertices. It requires novel algorithms (Section 5).

SUM: outputs G3 and G6. It favours groups in which every vertex is pretty good, but it can miss most of the best vertices in the dataset (c.f., G2). It admits very fast linear optimisation.

Group	Points
G2	{a,b,c}
G3	{d,e,f}
G4	{g,h,i}
G5	{j,k,l}
G6	{m,n,o}



Model	Output Groups	Excluded Groups
GENERAL	G2, G3	G4, G5, G6
SUM	G3, G6	G2, G4, G5
MIN [15]	G4, G5	G2, G3, G6
PERMUTE [23]	G2, G3, G4, G5, G6	-

Figure 3.1: A thorough comparison of skyline community models on a toy dataset designed to identify the primary differences in the characteristics of groups each model prefers or excludes. The table below indicates which groups are output or excluded by each model that we consider in this work.

MIN [15]: outputs G4 and G5. Prefers avoiding groups with weak vertices, and is thus agnostic to how good some of the non-weak vertices are (c.f., G2, G3). On this manufactured example, it does not return any group returning any skyline point. Integrates with peeling algorithms for k-core listing.

PERMUTE [23]: outputs everything! Integrates with k-clique listing.

To summarise, we see that GENERAL, SUM, and MIN express an interesting trade-off space, with SUM preferring pretty good points on average, GENERAL preferring to capture groups that contain the best points in the dataset, and MIN preferring to avoid the weakest points as best as it can, with debatable success on G5. True to the original claim, PERMUTE takes no position on those trade-offs and, for this example anyway, returns the union of the other methods.[1]

[1]In general, PERMUTE might not return all groups that MIN does.

Chapter 4

A Review and Comparison of Related Algorithms

Following Lin et al. [18], we denote as $D(S) = S \setminus \mathrm{SKY}(S)$ "non-skyline" vectors, i.e., those for which at least one other vector in $S \subseteq P$ can be found that dominates it.

The skyline concept can be used to recursively partition a set of vectors into *layers*.

Definition 7 (Skyline layer [19])**.** *Given a set P of vectors in d-dimensional space, $\lambda_1 = \mathrm{SKY}(P)$. Moreover, $layer_i = \mathrm{SKY}\left(P \setminus \bigcup_{j=1}^{i-1} \lambda_j\right)$, for $i > 1$.*

Observe that each vector in $v \in P$ is thus in one unique layer. We denote the index of the layer of v by $\lambda(v)$.

Observe that this is a generalisation of the definition given by Liu et al. [19] and used for community detection by Zhang et al. [23]; they require a one-to-one mapping between points in U' and points that dominate them in U, necessitating $|U| = |U'|$ and generating substantially many skyline groups. The generalisation in Definition 5 allows for several points in one group to be dominated by the same point in the other group, which causes the dominance predicate to be true more often and the skyline operator to be more selective. We will use it to find the set of k-clique communities in G that are not group-dominated, as stated below:

Analogously a vertex u dominates another vertex v iff $w(u) \succ w(v)$ [16].

4.1 Access order

Chomicki et al. [5] demonstrated with the *Sort-First Skyline* (SFS) algorithm that access order could significantly accelerate skyline computation if points are accessed

according to a function with the property that points can only be dominated by ones that precede it, such as Manhattan Norm, by exploiting the transitivity property of dominance (Property 1).

Liu et al. [19] adapted this idea for the generation of skylines over groups by accessing points according to their skyline layer (Definition 7 above). We adapt this idea for skyline communities and describe it below.

We can induce a partial order over the vertices of a graph $G = (V, E)$ with vertex labelling function w: given two vertices $u, v \in V$, we order u first iff $w(u) \succ w(v)$. To flatten the partial order into a canonical total order, we "tie-break" by vertex id. The partial order corresponds to the skyline layers of the vertex labels.

We denote the access order of a vertex v by $\eta(v)$ and observe both that the range of η is $[1, n]$ and $\eta(u) \neq \eta(v), \forall u, v \in V$. We have that $\eta(u) < \eta(v)$ iff:

- $\lambda(u) < \lambda(v)$ or
- $\lambda(u) = \lambda(v)$ and $u.ID < v.ID$

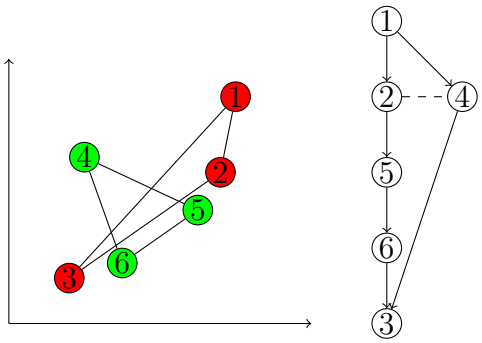


Figure 4.1: Skyline layer structure

Figure 4.1 presents a sample network and its corresponding skyline layer structure, where the first layer is {1}, second layer is {2, 4}, third layer is {5} and so on. So the access order for this graph is {1,2,4,5,6,3}.

Lemma 1. *Given two k-cliques S and S':*

$$\min_{s \in S \setminus S'} \eta(s) < \min_{s' \in S' \setminus S} \eta(s') \implies S' \not\succ S.$$

Proof. A k-clique S' can dominate another k-clique S, if every node in $S \setminus S'$ is dominated by at least one node in $S' \setminus S$. If $u \leftarrow \min_{s \in S \setminus S'} \eta(s)$ and $v \leftarrow \min_{s' \in S' \setminus S} \eta(s')$

and $\eta(u) < \eta(v)$, it means the skyline level of node v is higher or equal to the skyline level of node u. And from the definition of skyline layer we know that a node can be dominated only by a lower skyline level node. So, node v can't dominate node u. Thus, $S' \not\succ S$ is true. □

Using the above lemma we can accelerate the group dominance check. Before considering a new group as skyline, we usually check if any of the previously generated skyline groups can dominate this group. So, instead of comparing with all these groups, we can skip some of the group comparisons just by looking at the lowest order node of these groups.

Lemma 2. *Consider S and S' are two k-cliques generated from the graph $\vec{G}$. Here $\vec{G}$ is the directed version of the input graph G, where all the adjacency nodes has higher access order value than the node itself. That is, there is an edge between node u and v, $u \to v$, only if $\eta(u) < \eta(v)$. If S is generated before S', then $S' \not\succ S$*

Proof. The adjacency list in $\vec{G}$ is sorted on access order and we generate k-cliques sequentially starting from the first node in the access order. So, every generated k-clique has a sorted order. And the preceding k-cliques are guaranteed to have lexicographical lower sorted order. That is, if a groups S is generated before S', the first unique node in S will have lower access order than the first unique node in S'.

From lemma 1, we know that a group can't be dominated by another group, if the lowest order node of the first group is smaller than the lowest order node of the second group. That is, $S' \not\succ S$ □

Using this lemma, we can progressively output the skyline k-cliques and also reduce the comparison space. Because our graph is sorted on access order and we know that any groups that are generated in future can't dominate the current group, so we don't have to compare the current group with every k-cliques in the graph to confirm whether it's a skyline or not.

Lemma 3. *Given three k-cliques S, S' and S'', if $S \succ S'$ and $S' \succ S''$, then $S \succ S''$*

Proof. $S' \succ S''$, means every node in S'' is dominated by at least one node in S'. We know, if a node $u \succ v$, then every attribute of node u is either equal and greater than node v and at least one attribute is strictly greater. When $S \succ S'$, it means there is at least one node in S which attribute is greater than any of the nodes in S'. So, if S can dominate S', it can also dominate S''. □

In function 3, 7 and 10, when a group is dominated by an existing skyline group, we discard that group. For the future generated groups, we are not going to compare them with this group. Because if any future group is dominated by this group, they will also be dominated by the existing skyline group that dominated this group.

4.2 Complete Algorithm Description

IsDominated

In the following function 1, IsDominated, we check a k-clique s against another k-clique c for dominance. Namely, if s dominates c, we return True, otherwise, we return False.

The algorithm has two parts. In the first part, line 1-2, we remove the common nodes between the two groups and in the second part, line 4-6, we check for dominance which has a quadratic time complexity in k.

Algorithm 1: IsDominated

Input: a k-cliques s, a k-clique c
Output: *True* if $s \succ c$; *False* otherwise

```
1 unique_s = s \ c
2 unique_c = c \ s
3 for v ∈ unique_c do
4     if ∄ u ∈ unique_s s.t. u ≻ v then
5         return False
6 return True
```

Find-k-Cliques

In the recursive function 2, Find-k-Cliques, we find all the k-Cliques involving node u. We use the directed version of G, $\vec{G}$ which we calculate after creating the skyline layer structure. In $\vec{G}$, there is an edge $u \to v$ only if $\eta(u) < \eta(v)$, where η represents the access order of a node in the skyline layer structure. This constraint makes sure that the result will be unique and sorted based on the skyline layer access order.

To form a k-Clique, every node has to be connected with every other node. So, in each stage of the recursion, we calculate the intersection of neighbours between previous nodes and the current node. If the list of nodes after intersection is less than

$k-1$, then we can't make a k-Clique by adding more nodes with the current list. If not then for each node in the intersected list, we add it to the current nodes list and call the next recursion by reducing the value of k by 1. When the value of k is 1, we add the node with current nodes list and output it as a k-Clique.

Algorithm 2: Find-k-Cliques

```
1  Let η be a total ordering on the nodes of the input Graph G, based on the
   skyline layer structure
2  G⃗ ← directed version of G, where u → v if η(u) < η(v)
3  N(u) ← neighbours of node u in G⃗
   Input: current nodes list c, node u, size k
   Output: Set of k-cliques: C
4  C ← ∅
5  if k = 1 then
6      c ← c ∪ {u}
7      C ← C ∪ {c}
8      return C
9  I = N(u)
10 for p ∈ c do
11     I = I ∩ N(p)
12 if |I| ≥ k − 1 then
13     for idx = 0 to |I| do
14         v ← I[idx]
15         C ← C ∪ Find-k-Cliques(c ∪ {u}, v, k − 1)
16 return C
```

Chapter 5

Proposed Algorithm

In this chapter, we introduce the algorithm to solve the kCSP problem. Broadly, we first process individual vertices to generate a data structure that encodes dominance relationships. We then iterate this data structure to generate cliques. This corresponds to an approach of answering a generic skyline query first, then satisfying the connectivity constraints in an order that is guaranteed to produce skyline communities.

We described the dominance-based data structure and the access order it facilitates in the previous chapter (Section 4.1). We then describe how we integrate that into a broader algorithm to generate skyline k-clique communities (Section 4.2). Finally, we provide a proof of correctness for the algorithm (Section 5.5).

5.1 Baseline

Function 3, Sky-Clique, computes all the skyline k-cliques in the graph G. We precalculate the skyline layer structure of G based on [19]. Then we process each node in the skyline layer until we get the expected result. In our experiment, we generated groups that has at least one member from the first skyline layer. The intuition here is, the skyline group will have at least one member (i.e. an extraordinary member) that is not dominated by any other member in the whole network.

In line 4, we calculate the k-cliques involving the current node, which is sorted based on the access order, described in Section 4.1. This sorted order makes sure that a group that is generated later can't dominate a prior group, which is explained in our lemma 2. Then in line 7-10, for each k-clique, we do the dominance check with

the existing skyline groups. If none of the skyline groups can dominate the current k-clique, we add it to the skyline list. Otherwise, we discard the current k-clique.

Complexity: The complexity of function 3 is $|C|$x$|S|$, where $|C|$ is the number of k-cliques in the graph and $|S|$ is the number of skyline k-cliques.

Algorithm 3: Sky-Clique

Input: Skyline layer structure of graph G, size k
Output: Set of skyline k-cliques: S

```
1  S = ∅
2  L = AccessOrderNodes
3  for u ∈ L do
4      C_u ← Find-k-Cliques({}, u, k)
5      for idx = 0 to |C_u| do
6          c = C_u[idx]
7          IsSky = True
8          for s ∈ S do
9              if IsDominated(s, c) = True then
10                 IsSky = False
11                 break
12         if IsSky = True then
13             S ← S ∪ {c}
14 return S
```

5.2 Trie Approach

Trie Insert

The following function 4, inserts a new k-clique in the Trie data structure. Starting from the TrieHead, for each node u in the k-clique, it checks whether a branch exists from the current Trie node with value u. If there exists a branch it updates the current Trie node with that branch, otherwise it creates a new branch with the value u and updates the current Trie node with the newly created branch.

Trie Dominate

Function 5, checks whether a k-clique s is dominated by any of the k-cliques inside the Trie data structure. Starting from the TrieHead, for each branch it checks whether the

Algorithm 4: TrieInsert

Input: a k-clique s
Output: void

```
1 T = TrieHead
2 for u ∈ s do
3     if ∃T.next : T.next.val = u then
4         T = T.next
5     else
6         T_new = new Trie node with val u
7         T = T_new
```

branch has common prefix with the current k-clique s. If it does, then the algorithm goes to the next Trie level until there is a mismatch or it reaches the last level. Otherwise, it checks the skyline layer level of the Trie node and the node from s. According to our Lemma 1, if the skyline layer level of a node is higher than another node, then the first node can't dominate the later. So, in line 8, if the skyline layer level of the Trie node is smaller than the node from s, we compare s with all the branches from the current Trie node. If at least one of the branches dominate s, we return true. Otherwise we continue the checking. This condition is not that strict and because of this we get some false-positive result. To exclude the false-positive result one can just change the condition to $Level(v) \leq Level(u)$ and it will produce the correct result with the overhead of higher execution time.

Let's assume we have two groups $\{1, 2, 3\}$ and $\{2, 3, 4\}$. Node 1, 2 and 3 are from the first skyline layer and node 4 is from the second skyline layer and node 1 dominates node 4. In our Trie, group $\{1, 2, 3\}$ will be inserted first and when we compare the second group $\{2, 3, 4\}$ we compare by removing the common prefixes from the groups. As these two groups have no common prefix, we will compare node 1 and 2 and as both of them are from the same layer we are not checking for dominance between these two groups. Thus the second group is also considered as skyline and inserted into the Trie.

In our baseline implementation, we compare the groups by removing the common nodes first. As these two group has node 2 and 3 as common, we only compare node 1 from the first group and node 4 from the second group. And as node 1 dominates node 4, according to our baseline solution the first group dominates the second, so it doesn't produce the second group as skyline. Whereas, our Trie dominate method

Algorithm 5: TrieDominate

Input: Trie node T, a k-clique s, current position ind
Output: $True$ if $\exists h \in Trie : h > s$, $False$ otherwise

```
1  u = s[ind]
2  while there is next in T do
3      v = T.next.val
4      if u = v then
5          if TrieDominate(T.next, s, ind+1) = True then
6              return True
7      else
8          if Level(v) < Level(u) then
9              C_v = list of branches from T
10             if ∃c ∈ C_v : c > s[ind, ind + 1, ... |s|] then
11                 return True
12 return False
```

produces this group as a false-positive result because of it's prefix based common node removal technique.

Filter False Positive

Function 6, is a $O(n^2)$ checking to filter out the false positive results from function 5. The skyline layer access order ensures that the first group will always be a true skyline-group. So we add the first group in the final result S', and for each of the remaining groups in S we check whether any group in S' can dominate it. If not then we add it to S', otherwise we discard it and go to next group.

Sky-Clique+

Function 7 is almost similar to function 3, where we compute the skyline k-cliques in the graph G. The only difference is, in function 7 we do the dominance check using a Trie data structure, whereas function 3 does a naive dominance check. Also, the Trie domination check 5 in line 7, can produce some false positive result. So we filter out the result in line 10, using the function 6.

As a final note, we observe that Zhu et al. [26] introduce a multicore parallel algorithm for skyline groups, based on an earlier algorithm for standard skyline queries [4].

Algorithm 6: Filter-False-Positive

```
Input: Set of k-cliques: S
Output: Set of skyline k-cliques: S'
if S = ∅ then
    return ∅
S' = {S[0]}
for s ∈ (S − {S[0]}) do
    IsSky = True
    for s' ∈ S' do
        if IsDominated(s', s) = True then
            IsSky = False
            break
    if IsSky = True then
        S' ← S' ∪ {s'}
return S'
```

Algorithm 7: Sky-Clique+

```
Input: Skyline layer structure of graph G, size k
Output: Set of skyline k-cliques: S
TrieHead = NULL
S = ∅
L = AccessOrderNodes
for u ∈ L do
    C_u ← Find-k-Cliques({}, u, k)
    for idx = 0 to |C_u| do
        c = C_u[idx]
        if TrieDominate(TrieHead, c, 0) = False then
            S ← S ∪ {c}
            TrieInsert(TrieHead, c)
S = Filter-False-Positive(S)
return S
```

This work has not been adapted to the detection of pareto-optimal cohesive sub-communities.

5.3 Quad Tree Approach

We observe that the program execution time mostly affected by the skyline-group size, namely because for every generated k-clique we compare them with the existing skyline-groups before deciding whether they should be added to the skyline-groups list or not. In our Trie implementation we tried to reduce that compare space by using the lowest order node between the two groups, which produces some false-positive result, because of the prefix based comparison.

So, we decided to reduce the compare space by using quad tree approach. A group S can be dominated by another group S', if every node of S either equal to or dominated by at least one node in S'. And a node u can dominate another node v, only if every attributes of u is either equal to or greater than v and at least one attribute is strictly greater. We can deduce this relation to our groups as well and say, a group S' can dominate another group S, if the upper bound of S''s attribute is greater than the upper bound of S. The upper bound of a group is the highest attribute value of the group members in each dimension. If the upper bound of a group is not higher than another group, that means there is at least one node in the first group which won't be dominated by the second group, thus we can skip this comparison.

Upper Bound

In the following function 8, we calculate the maximum attribute value among the members of a group s, in each dimension. In line 1, we initialize the array with a big negative value and in line 2, for each member we update the corresponding attribute in line 4.

Algorithm 8: UpperBound

Input: A group s
Output: List of attributes

```
1 Attr_s[d] = {-inf, -inf, -inf, ..., -inf}
2 for u ∈ s do
3     for idx = 0 to d do
4         Attr_s[idx] = max(Attr_s[idx], Attr[u][idx])
5 return Attr_s
```

Quad Group Search

Function 9, returns the skyline group ids from the quad tree, which has a possibility of dominating the current group. Line 1, checks if the quad node is a leaf node or not. If yes then it returns all the skyline group ids from this nodes region. In line 4, we check whether the upper bound of the current group falls into the bottom left region of the quad tree. If yes then each region of the current node can dominate the group, so we search every region of q.

If a groups upper bound is on the top left region of q, only the nodes from the top left and top right can dominate it. A bottom right region group can be dominated by the bottom right and top right region nodes. And a top right region group can only be dominated by the top right region nodes of q. So we reduce the search space accordingly and return only those group ids, which has a chance to dominate the current group.

Algorithm 9: QuadGroupSearch

Input: Quad node q, current upper bound b
Output: Set of skyline group ids: S_{ids}

```
1  if q is a leaf node then
2      return q.vals
3  S_ids ← ∅
4  if b is in bottomLeft of q then
5      S_ids = S_ids ∪ QuadGroupSearch(q.bottomLeft, b)
6      S_ids = S_ids ∪ QuadGroupSearch(q.bottomRight, b)
7      S_ids = S_ids ∪ QuadGroupSearch(q.topLeft, b)
8      S_ids = S_ids ∪ QuadGroupSearch(q.topRight, b)
9  else if b is in topLeft of q then
10     S_ids = S_ids ∪ QuadGroupSearch(q.topLeft, b)
11     S_ids = S_ids ∪ QuadGroupSearch(q.topRight, b)
12 else if b is in bottomRight of q then
13     S_ids = S_ids ∪ QuadGroupSearch(q.bottomRight, b)
14     S_ids = S_ids ∪ QuadGroupSearch(q.topRight, b)
15 else if b is in topRight of q then
16     S_ids = S_ids ∪ QuadGroupSearch(q.topRight, b)
17 return S_ids
```

Sky-Clique++

In the following function 10, we calculate the skyline k-cliques using the quad tree to reduce the comparison space. In line 8, we calculate the upper bound of the current k-clique. Then in line 9, by using the quad tree optimization we only get those existing skyline k-cliques, which has a possibility of dominating the group c. Then in line 11 to 14, we check the current group c with every skyline k-cliques from line 9. If none of these group can dominate c, we add c to the list of skyline k-cliques in line 16 and also add the upper bound of c to the quad tree in line 17. Otherwise, we discard c and continue.

Algorithm 10: Sky-Clique++

```
   Input: Skyline layer structure of graph G, size k
   Output: Set of skyline k-cliques: S
 1 QuadHead = NULL
 2 S = ∅
 3 L = AccessOrderNodes
 4 for u ∈ L do
 5     C_u ← Find-k-Cliques({}, u, k)
 6     for idx = 0 to |C_u| do
 7         c = C_u[idx]
 8         Attr_c = UpperBound(c)
 9         S_quad = QuadGroupSearch(QuadHead, Attr_c)
10         IsSky = True
11         for s ∈ S_quad do
12             if IsDominated(s, c) = True then
13                 IsSky = False
14                 break
15         if IsSky = True then
16             S ← S ∪ {c}
17             Insert Attr_c into the Quad Tree
18 return S
```

5.4 An Example

In figure 4.1, for 2-clique, first we will generate $\{1, 2, 3\}$. As the set of skyline k-cliques, S is empty at the beginning, $\{1, 2, 3\}$ will be added to S. So after first

iteration $S = \{1, 2, 3\}$. In the second iteration, no more nodes left in the skyline layer and so our Algorithm will terminate and return $S = \{1, 2, 3\}$ as skyline k-clique. Even if we generate the other 2-clique $\{4, 5, 6\}$, it will be discarded by our algorithm. Because from the skyline layer structure we can see node 1 dominates node $\{4, 5, 6\}$, which means group $\{1, 2, 3\}$ will cover the dominance range of group $\{4, 5, 6\}$ and more, thus making it a non-skyline k-clique.

5.5 Algorithm Proof

Proof. **Statement**: At any step i, $result_i = result_n \setminus clique_{sorted}[i+1, i+2, \ldots, n]$

Base Case: For $i = 0$, $result_0 = clique_{sorted}[0]$

The first computed k-clique is a skyline k-clique. Because of the access order sorting, the first k-clique will have at least one node that is from the 1st skyline layer and isn't dominated by any other nodes in the graph. Thus no other k-cliques in the graph can fully dominate this group, which makes it a skyline k-clique.

Induction Hypothesis: Assume the statement holds for $i = k$, meaning $result_k$ is true: $result_k = result_n \setminus clique_{sorted}[k+1, k+2, \ldots, n]$

Inductive Step: Now, let's prove the statement holds for $i = k + 1$. That is: $result_{k+1} = result_n \setminus clique_{sorted}[k+2, k+3, \ldots, n]$

At this step, $clique_{sorted}[k+1]$ is compared with every group in $result_k$. If $\nexists U \in result_k$, such that $U > clique_{sorted}[k+1]$, then $result_{k+1} = result_k \cup clique_{sorted}[k+1]$, otherwise $result_{k+1} = result_k$

According to our lemma 2, no group in $clique_{sorted}[k+2, k+3, \ldots, n]$ can dominate $clique_{sorted}[k+1]$, so we can progressively add this to our final result.

According to Algorithm 3, a k-clique is added to the result only if it is not dominated by any of the calculated skyline k-cliques. So, if $clique_{sorted}[k+1]$ is a skyline k-clique, it will not be dominated by any of the k-cliques in $result_k$ and will be added to $result_{k+1}$. Therefore, we can say $result_{k+1}$ is true.

Conclusion: For $i = n$, $result_n = result_n \setminus \{\}$ □

Chapter 6

Experiments

In this section, we conduct two empirical investigations. The first, in Section 6.1, overview of compared model.

The second, in Section 6.2, is a case study, similar to those in [23, 15]. It expands the comparison of models from Section 3.4 to a large, real dataset.

Finally, in Section 6.3, compares the efficiency of the algorithms presented in Section 5.

6.1 Model Comparison

In Section 3.4, we provided a comparison of skyline community models on a contrived dataset to illuminate their differences; In these experiments we compare their results on real data.

Experiment Setup

Here, we describe the dataset construction, the experiments that we run, and the architecture on which the experiments are conducted.

Models Compared We compare four models of interest. To recall:

- **SUM** Each group is represented by a virtual point created by an element-wise *summation* per dimension of all points in the group.
- **MIN** [15] Each group is represented by a virtual point created by independently *taking the smallest (i.e., worst) value* on each dimension, selected from all points in the group.

- PERMUTE [23] A group is said to dominate another non-equal group if it can be put into one-to-one correspondence such that each point in the second group is equal to or dominated by *the corresponding point* in the first group.
- GENERAL A group is said to dominate another non-equal group if its distinct elements can be put into many-to-one correspondence such that each point in the second group is equal to or dominated by *some point* in the first group.

Dataset As in [23, 15], we use a subset (AminerSmall) of the *Aminer* co-authorship dataset[1] that may be more familiar to the reader. The vertex subset contains the authors in the areas of data mining, machine learning, data management and databases; we retain edges between authors who have co-authored at least three papers.

The full *Aminer* dataset contained 1,712,433 authors and 4,258,615 collaboration relationships; moreover, each author node has five numberic attributes (#paper, #citation, h-index, p-index and up-index). The h-index metric [10] measures the academic influence of an author and the p-index metric [1] is a measure of paper count, weighted by position in the author list.

The AminerSmall subset contains 7,120 nodes and 9,218 edges; we use four numeric attributes for each author normalised to the range $[0, 1]$: number of papers, number of citations, h-index and p-index, which we expect have positive correlation to each other.

Queries Again, following [23, 15], we present a case study on three separate query types:

Subspace Team Search: We identify groups that are pareto-optimal with respect to some subspace. In particular, we consider the subspace {#paper, p-index}

Dynamic Team Search: We identify groups that are related to a target vertex using a dynamic, computed vertex label. In particular, we measure the relative overlap of co-authorship networks using the Jaccard Index [12]; i.e., given a query vertex v, we compute for every vertex u the label: $(N(u) \cap N(v))/(N(u) \cup N(v))$. This dynamic attribute is used jointly with the static h-index.

Partial-Order Based Search: We identify groups using an attribute that is only partially ordered. In particular, we define a preference hierarchy over research areas (Machine Learning = 3, Data Mining = 2, Artificial Intelligence = 1, Statistics =

[1] https://www.aminer.cn/aminernetwork

1, Other = 0). The SUM model is excluded because it cannot handle partial orders without weighting the preferences.

6.2 Case Study

Subspace Team Search

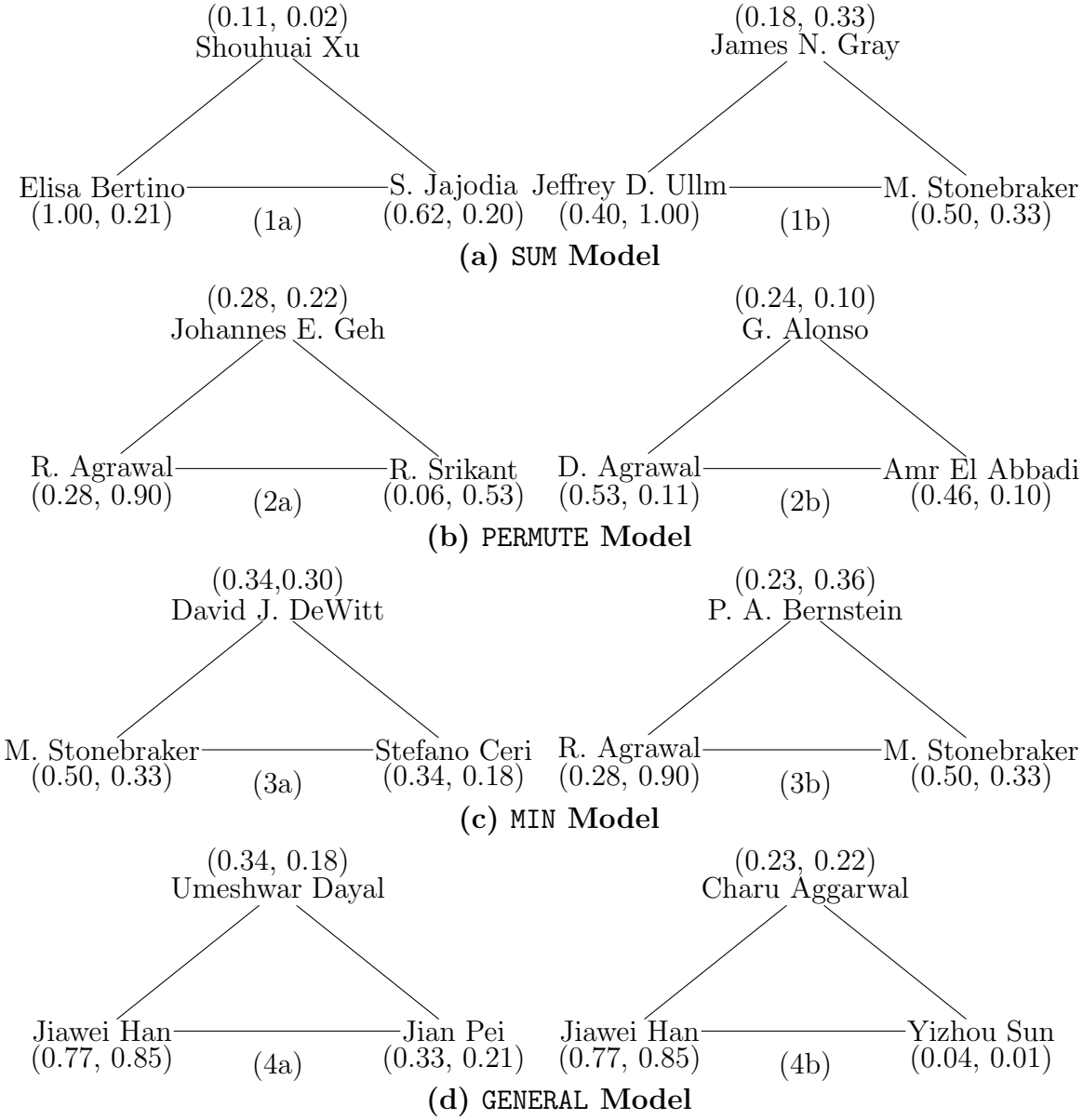


Figure 6.1: Different model results for k=3 clique and based on the number of publication and p-index

We begin with the subspace team search query. For each model, we report in Figure 6.1 some "exceptional" results. In our experiment, we observe that each model generates some groups which are common to each other. By "exceptional", we mean the unique results that are generated by each model.

$team_{1a}$ has the highest total number of publications and $team_{1b}$ has the highest total h-index in the network. This is to be expected because the SUM approach aggregates each attribute by computing their average.

All the models except MIN output these two groups. MIN doesn't produce these groups, because $min(Attr_{1a}) = (0.11, 0.02)$ and $min(Attr_{1b}) = (0.18, 0.33)$, which are dominated by $min(Attr_{3b}) = (0.23, 0.33)$.

On the other hand, PERMUTE outputs $team_{2a}$ and $team_{2b}$, despite having an overall poor total over each attribute.

$team_{3a}$ and $team_{3b}$ has a good minimum value for each attribute and both MIN and PERMUTE output these groups. Our model doesn't output these groups, because there is no skyline nodes in them and SUM doesn't output these groups either because their average attribute isn't good compared to other skyline groups.

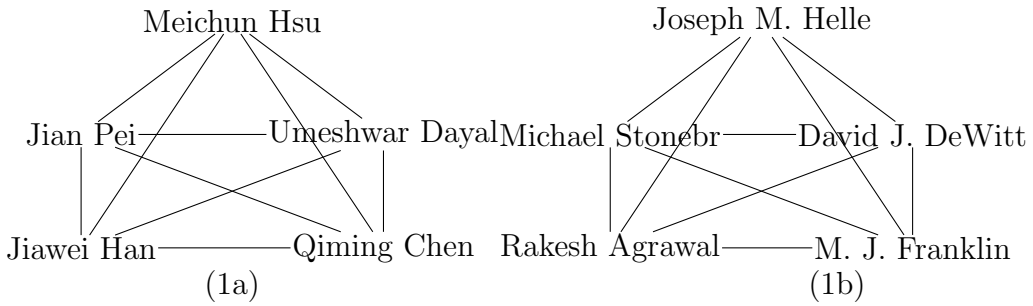
$team_{4a}$ is a very good group in terms of the overall quality (good publication and h-index), which is missed by MIN because of the minimum attribute criterion. On the other hand, $team_{4b}$ has one skyline member and two other members who are comparatively poor, specially "Yizhou Sun". MIN and SUM don't output this group, but our model and PERMUTE produce this group as skyline group.

Other experiments based on the number of citation and h-index, number of publication and h-index etc. produced the similar effect on results for different models.

Searching groups based on a query node

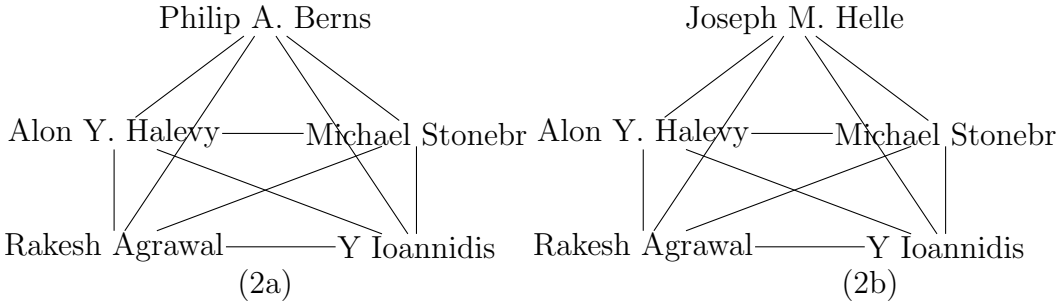
In this case study, our goal is to find communities where members are similar to a given query node u and have a good academic influence. The similarity between two nodes is measured by their Jaccard similarity. The Jaccard similarity between two nodes u and v is $(N(u) \cap N(v))/(N(u) \cup N(v))$. Higher Jaccard similarity means the node v is quite similar to node u.

In Figure 6.2, we show the results of different models (SUM, PERMUTE, MIN and Our) for k=5 clique and based on h-index and jaccard similarity with "Jiawei Han". For comparison we pick only the exclusive results from each model, as some of the groups are produced by all four models.



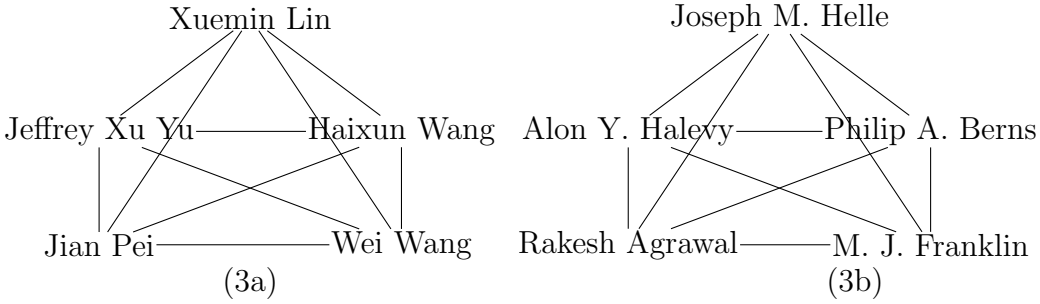
(1a) (1b)

(a) `SUM` Model



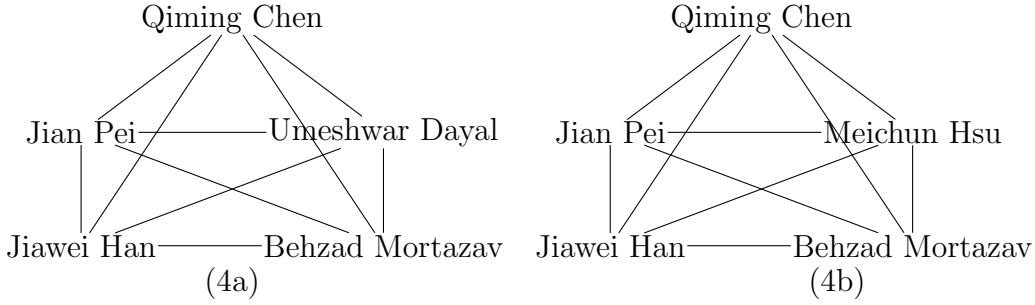
(2a) (2b)

(b) `PERMUTE` Model



(3a) (3b)

(c) `MIN` Model



(4a) (4b)

(d) `GENERAL` Model

Figure 6.2: Different model results for k=5 clique and based on h-index and jaccard similarity with "Jiawei Han"

Here, $team_{1a}$ has the highest similarity with "Jiawei Han" and it's been produced by all four models. On the other hand, $team_{1b}$ has the highest h-index and is missed by MIN and our model. We miss this group because it doesn't have any skyline nodes and MIN misses it because of the minimum attribute criteria. Based on our goal of finding communities similar to "Jiawei Han", this group has the least similarity.

$team_{2a}$ is produced by the PERMUTE and SUM model, which has a good h-index but less similarity with the query node. $team_{2a}$ and $team_{2b}$ has only 1 unique member between them, "Philip A. Bern" and "Joseph M. Helle". Although both of them has very poor similarity with "Jiawei Han", PERMUTE produces both of them as output, because "Philip A. Bern" has a better similarity and "Joseph M. Helle" has a better h-index.

Despite having an overall poor attribute, $team_{3a}$ is produced by PERMUTE and MIN model. Whereas $team_{3b}$ is only produced by MIN. Both of these group has a very poor similarity with our query node "Jiawei Han".

$team_{4a}$ and $team_{4b}$ are produced by all the models except MIN doesn't produce $team_{4a}$. Between these two groups "Umeshwar Dayal" and "Meichun Hsu" are unique. Both of them has a good similarity with the query node, but "Umeshwar Dayal" has a better h-index and "Meichun Hsu" has a better similarity.

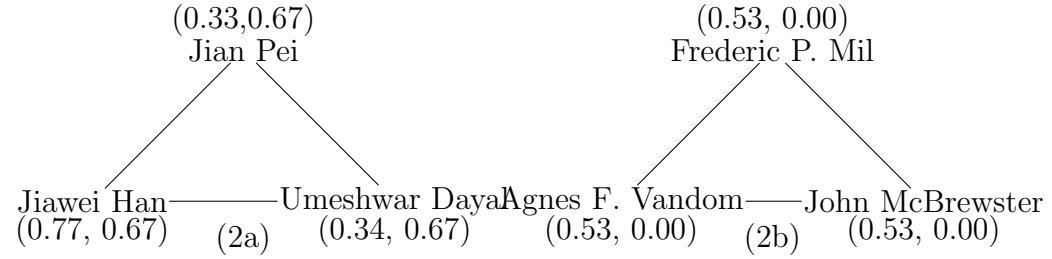
For this particular query node, our model generated groups that has "Jiawei Han" in every one of them. But that's not always the case. Our model can also generate group with least similarity to query node, if the group has a very good value in other attributes.

Search groups based on the partial order

In this case study our goal is to find skyline groups based on some partial order (research area). We want to find groups where the order of research topics are as follows: Machine Learning, Data Mining, Artificial Intelligence, Statistics and so on. As AGGR operates on numerical attributes and on total order we will not include this model in this case study.

Figure 6.3, represents the result of this case study, where we aim to find groups based on research topic. This result includes PERMUTE, MIN and our model for k=3 clique and based on the number of publication and partial ordering on research topic. Here, we picked results that are unique to each model and better describe the differences between each model.

(a) PERMUTE Model



(b) MIN Model

(c) GENERAL Model

Figure 6.3: Different model results for k=3 clique and based on the number of publication and order on research topic(Machine Learning, Data Mining, AI, Statistics and so on)

$team_{1a}$ and $team_{1b}$ are produced by PERMUTE model, but not by MIN and our model. Despite having a skyline node our model doesn't produce $team_{1a}$, because it is dominated by $team_{3a}$. MIN doesn't produce this group because $min(Attr_{1a}) = (0.32, 0.00)$ is dominated by $min(Attr_{2a}) = (0.33, 0.67)$. $team_{1b}$ doesn't have any skyline node, so our model doesn't produce this and MIN doesn't produce this group because of the minimum attribute condition.

$team_{2a}$ is a good group generated by MIN and PERMUTE model. Although this group has good attribute our model doesn't produce this group because it doesn't have any skyline node. On the other hand, $team_{2b}$ has the least similarity with our

expected result (based on the research topic). So our model doesn't produce this group, but both MIN and PERMUTE does.

$team_{3a}$ is generated by our and PERMUTE model, but missed by MIN. $team_{3a}$ is a very good group in both dimension, except the number of publication of "Shouhuai Xu". Because of this reason, MIN doesn't produce this group as it is dominated by group's that has better minimum attribute ($team_{2a}$). For this dataset, $team_{3b}$ has the highest similarity to our expected research topic and is output by all three models.

6.3 Algorithm Performance

In Section 5, we introduced a novel algorithm for the GENERAL model, along with spatial indexing techniques to accelerate dominance tests. In these experiments we evaluate the scalability and relative performance of those algorithms.

Experiment Setup

Dataset	Verts	Edges	Tri	CC	d_{max}	k_{max}
email-Enron	36K	367K	727k	0.50	1383	43
soc-Delicious	536K	1.3M	1.5M	0.03	3216	33
com-Youtube	1.1M	2.9M	3M	0.08	28754	51
soc-Flixster	2.5M	7.9M	24M	0.08	1474	68

Table 1: Statistical properties of the selected datasets (Verts = Vertices, Tri = Triangles, CC = Clustering Coefficient, d_{max} = Maximum degree, k_{max} = Maximum k-core)

Here, we describe the dataset construction, the experiments that we run, and the architecture on which the experiments are conducted.

Methods Compared We compare four methods of interest:

- **Baseline** (Algorithm 3) The adaptation of [23] for the GENERAL model. This serves to contextualise the impact of the novel ideas presented in the following methods.
- **Trie** (Algorithm 7) The GENERAL model, using trie-accelerated dominance tests.

- **FP-Trie** (Algorithm 7 without Line 11 to filter false positives) The GENERAL model, using trie-accelerated dominance tests with one-sided error. This demonstrates the cost and importance of the post-processing filter of the Trie method.
- **QuadTree** (Algorithm 10) The GENERAL model, using quad-tree-accelerated dominance tests.

Datasets To study the algorithms at scale with realistic connectivity constraints, we combine large real networks with synthetic labels. The four networks are obtained from the Stanford networks repository.[2] and described in Table 1. The maximum k-core value is often considered to be an indicator of problem difficulty for cohesive sub-community discovery.

To study the independent impact of the size of the label set, we generate synthetic label vectors. Floating point labels are generated in the range $[0, 1]$ from a uniform distribution and the dimensions of the vectors are generated independently. It is generally known that in skyline problems, increasing correlation between dimensions decreases problem difficulty [2].

Experiments We analyse the impact of two input parameters on algorithm perfomance: k, the size of the cohesive sub-communities that we endeavour to discover; and d, the number of labels on each vertex. These parameters are directly manipulated in the experiments to study their effect in a controlled fashion. Default values for these settings are $k = 4$ and $d = 3$.

Indirectly, we also observe the impact of network characteristics. By comparing results on different datasets, we can observe the joint impact of the number of vertices, number of edges, maximum degree in the graph, and maximum coreness value in the graph.

Architecture The algorithms are implemented in C++ and compiled using gcc 10.2.0 and -Ofast optimisation. The methods are compiled independently using template parameters. The experiments are run on a 6-core Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz machine with 16GB of memory. Timings are measured by instrumenting the code with the C++ std::chrono library and doesn't include the time to read the network into main-memory. It also excludes the time to filter the network and build the skyline layers. The networks are loaded into a main-memory adjacency list and vertex labels are loaded into a main-memory vector of vectors. The timing

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

does not include the cost of writing results to the console or to file, but it does include recording results in a main-memory vector of vectors.

Results and Discussion

Here, we report and discuss results.

Number of skyline communities We begin by inspecting the output size of the GENERAL model, as well as the accuracy of the FP-Trie method. Note that the output size for Baseline, Trie, and QuadTree are identical and all represent the GENERAL model; thus, only Trie is shown.

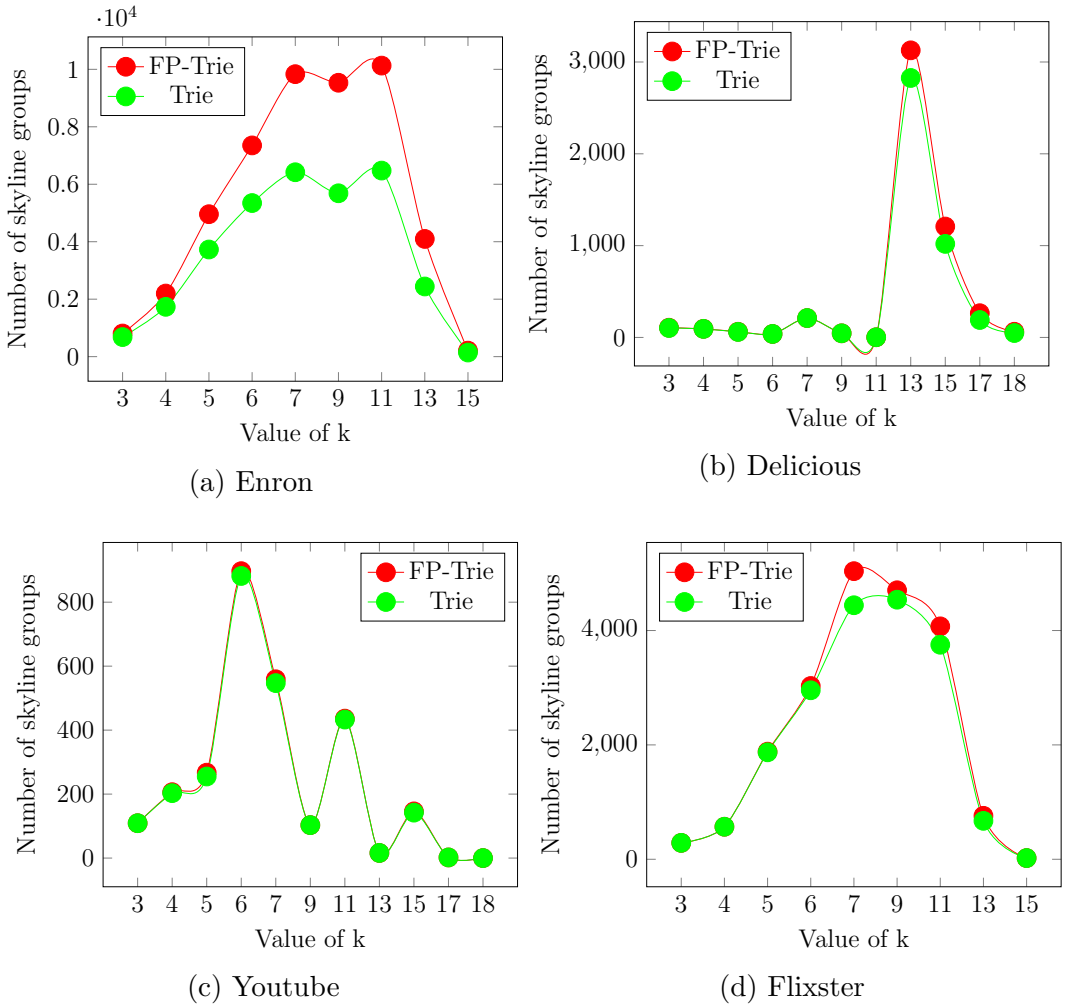
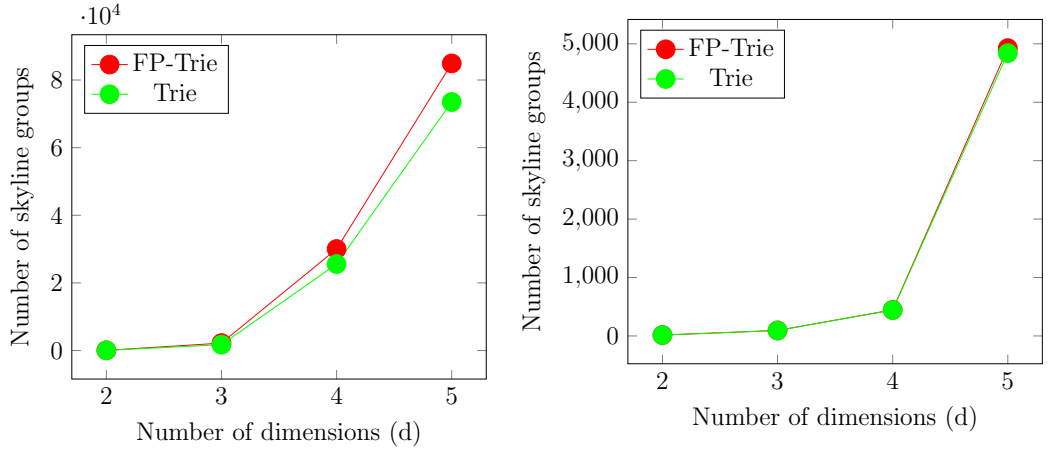


Figure 6.4: Output size comparison between the false positive and actual result in different datasets on varying k (d=3)

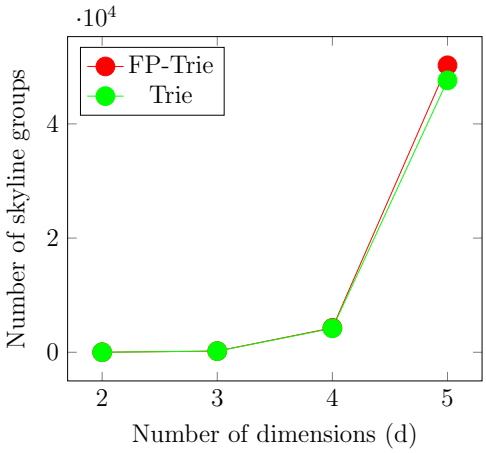
Figure 6.4 illustrates the number of skyline communities in each dataset, as a

function of k. Each subfigure shows a different network, the y-axis shows the total number of communities in the skyline, and the x-axis shows the number of vertices in each community, i.e., k. We vary k from 3 until a value large enough that the output on that network is nearly empty.

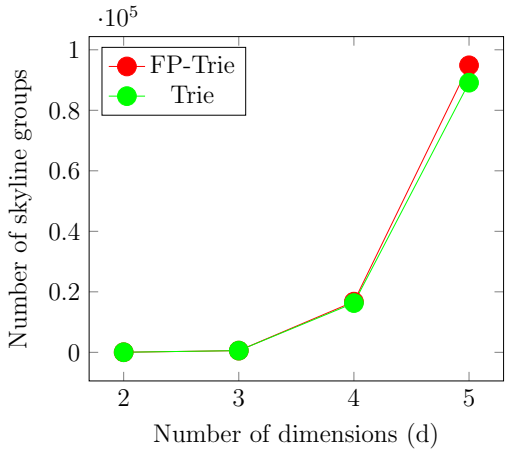


(a) Enron

(b) Delicious



(c) Youtube



(d) Flixster

Figure 6.5: Output size comparison between the false positive and actual result in different datasets on varying d (k=4)

Figure 6.5 illustrates the number of skyline communities in each dataset, as a function of d. Each subfigure shows a different network, the y-axis shows the total number of communities in the skyline, and the x-axis shows the number of dimensions in the attributes, i.e., d. We vary d from 2 to 5, where we can see the output size increases exponentially as we increase the dimension.

The first observation is that the Enron dataset produces the largest output. This

is interesting, because that dataset is much smaller than the others, has a smaller maximum degree, and has only the second largest maximum coreness.

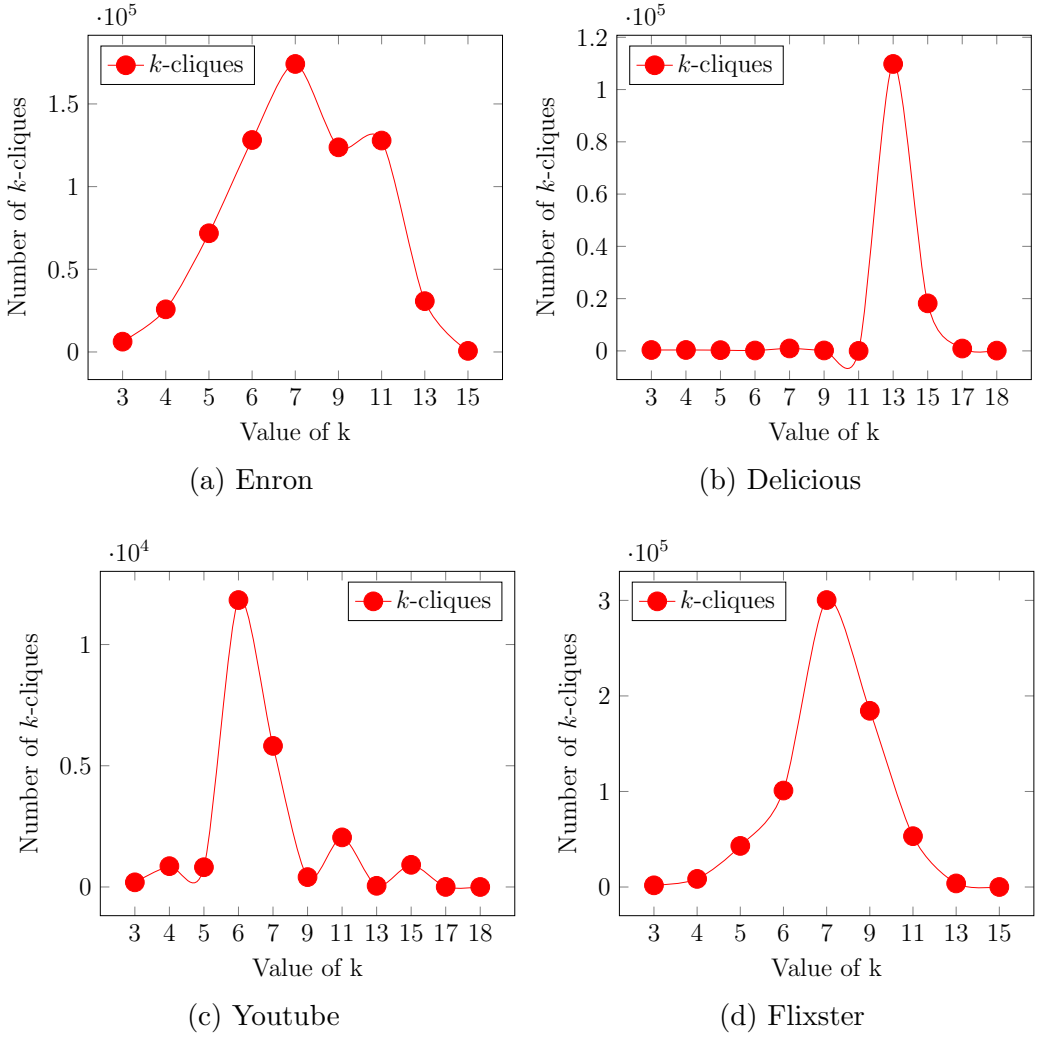
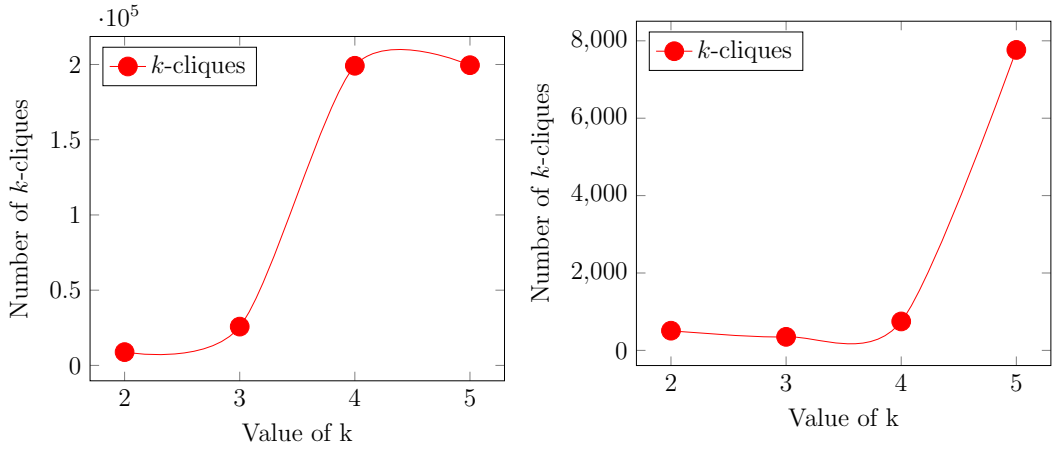


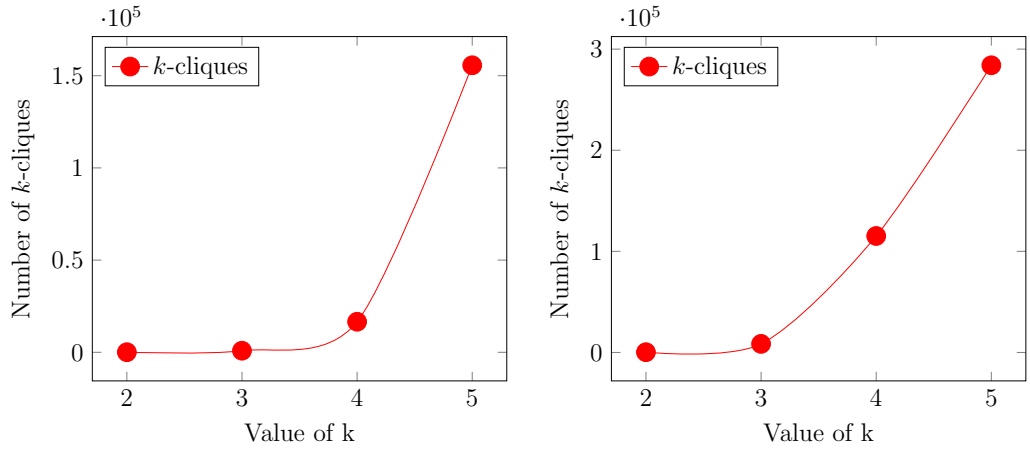
Figure 6.6: Total number of k-cliques in different datasets on varying k (d=3)

Next, note that if Trie and FP-Trie have similar output sizes, then FP-Trie does not produce very many false positives and the necessity of post-processing is lower. We observe that Trie and FP-Trie produce similar output sizes on all datasets except Enron. Compared to the other datasets, Enron dataset has higher average clustering coefficient (0.50) (Delicious(0.03), Youtube(0.08), Flixster(0.08)). Because of that, each node in Enron is clustered with multiple k-clique. When two groups has no common prefixes, the Trie dominance test assumes the current group as skyline, if the first node of the current group has a higher or equal skyline layer level compared to the Trie node. In our actual dominance test we remove all the common nodes

between two groups and compare only the unique nodes. In Trie, we can't remove common nodes other than the prefixes. As a result, if node 1 and 2 are from the same skyline layer, even though node 1 dominates node 5 and 7, we say group {1, 2, 5} can't dominate group {2, 5, 7}. In Enron dataset, we generate a number of k-cliques, where the prefix is different, but they have some common node which can't cancel each other out, and we end up with more false positive results.



(a) Enron (b) Delicious



(c) Youtube (d) Flixster

Figure 6.7: Total number of k-cliques in different datasets on varying d (k=4)

Finally, we observe that no dataset has a large output size beyond $k = 18$. As the size of k increases, we remove nodes that are not part of any k-clique. Thus, the total number of k-cliques decreases and so does the output size. Figure 6.6 better explains this, where we can see the total number of k-cliques decreases as we increase the value of k.

Effect of varying k Next, we analyse how the size of the communities impacts performance. Figure 6.8 reports execution time in milliseconds on the y-axis as a function of community size, k, on the x-axis. For each plot, we use the same range for k as identified in the previous experiment.

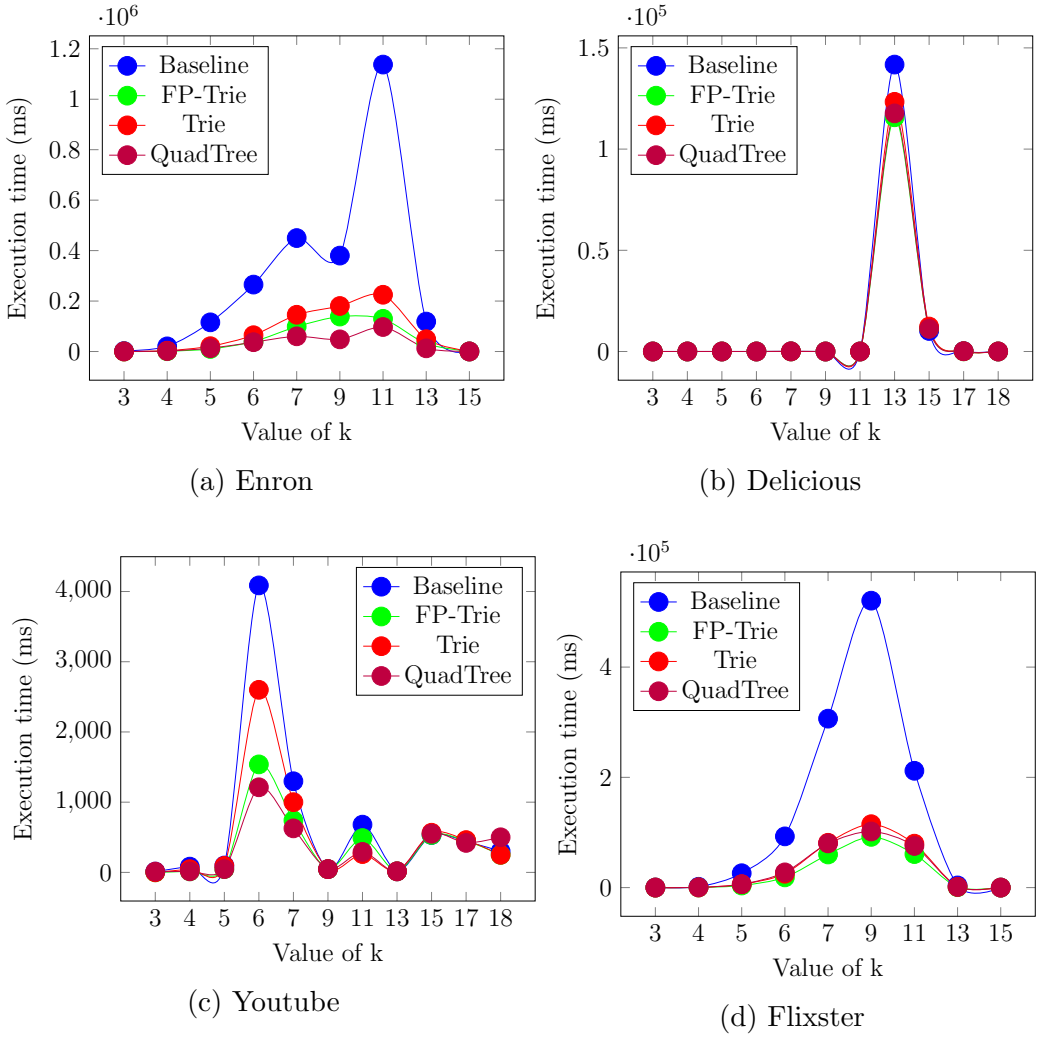


Figure 6.8: Efficiency in different datasets on varying k (d=3)

Observe first that the performance of Baseline follows the output size *very* closely. Although execution time does not spike as quickly on Enron and Flixster as the output size does, the curves are nonetheless very similar. This indicates the size of the skyline has a greater impact on performance than do the input parameters or graph characteristics. This is not surprising, as each candidate group must be compared against the known skyline solutions. In fact, most skyline problems suffer from this problem, which is most apparently evidenced by poor performance on anti-correlated

dimensions.

Fortunately, the spatial indexing techniques dampen this effect substantially, up to a factor of four on the YouTube dataset. This reflects that the major cost of the algorithm is testing dominance between groups. Moreover, the execution time for the Youtube dataset doesn't follow a smooth pattern, which is the result of graph filtering. When we filter the graph for higher core, we remove nodes that are not part of that higher k-core, and those removed nodes can be from the first skyline layer. As a result, some nodes are going out of the first skyline layer and some are coming in. So, the size of the output is also changing based on the number of nodes in the first skyline layer. Because in our experiment we are calculating groups which starts with the first skyline layer node. So if the number of nodes in the first skyline layer is higher, the output size is also higher. And so is the execution time, and vice versa. This is exactly the observation made by Zhu et al. [25]; the network connectivity constraints and generalised problem definition appear inadequate to fully address that concern.

Effect of varying d Finally, we analyse how the size of the vertex label vectors affects performance. Figure 6.9 reports execution time in milliseconds on the y-axis versus dimensionality, $d \in [2, 5]$, on the x-axis. All four methods are shown as separate trendlines. Note that the y-axis is logarithmic to facilitate observing the large impact that dimensionality has.

The first observation, revealed by switching to a logarithmic y-axis, is that performance degrades sharply with d for all methods. This is consistent with prior literature on skyline computation [2]. Similarly, we observe that d has a more profound impact than the dataset: For example, all methods process YouTube at $d = 5$ slower than Enron at $d \leq 4$. Also, we can see Delicious is so much faster at $d = 5$ than the other three dataset. It has fewer nodes and small clustering coefficient, which results in less number of k-cliques and faster processing time. This suggests that the skyline computation, which is most sensitive to d, is more expensive than the clique listing, which is most sensitive to the edge set.

Comparing the methods, we observe, as expected, that the FP-Trie implementation is the most efficient, even as d increases. The $O(n^2)$ post-processing can remove false positives and maintain a performance advantage over the baseline method; however, the QuadTree method consistently outperforms this approach, and the improvement offered by QuadTree over Trie grows slowly with increasing d.

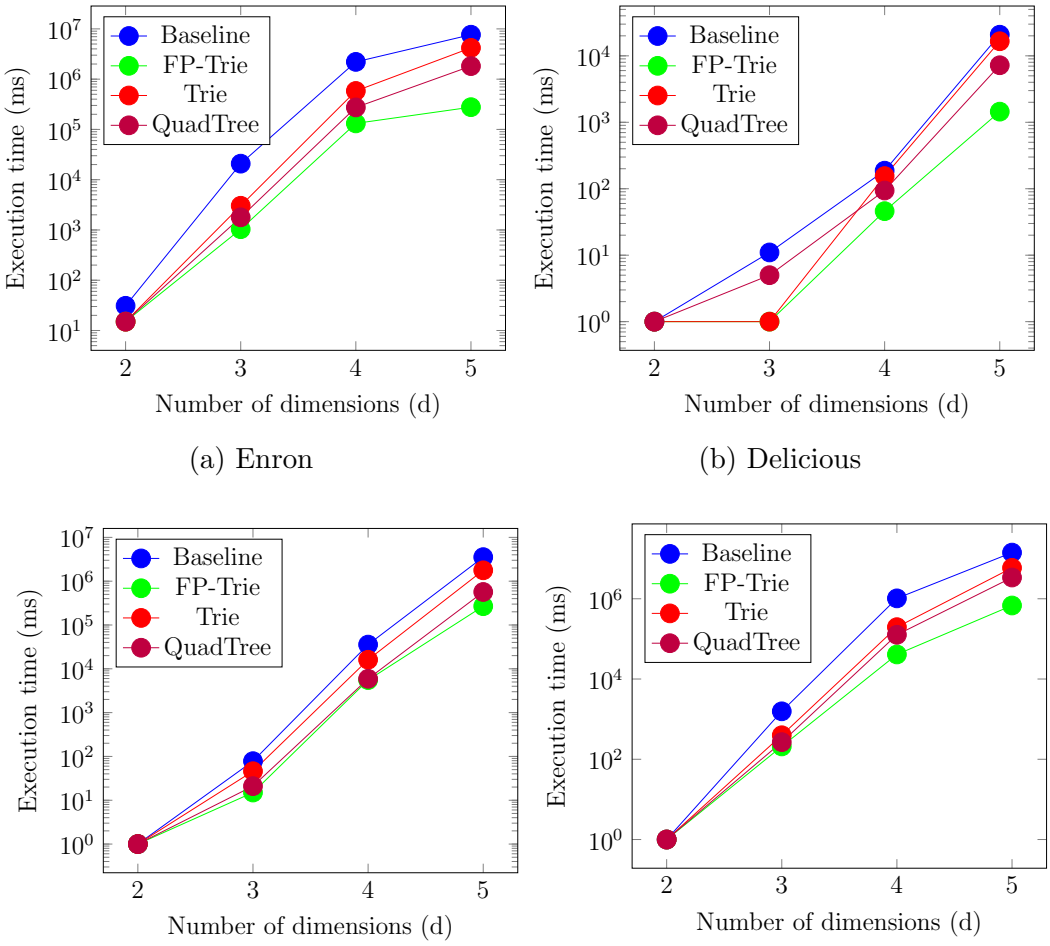


Figure 6.9: Efficiency in different datasets on varying d (k=4)

Chapter 7

Conclusions

7.1 Conclusion

In this work, we propose a novel skyline model to retrieve cohesive subgraphs (i.e. k-clique, k-core etc.) from a multi-valued network. An efficient progressive algorithm is proposed to calculate the skyline groups where we reduce the number of group comparison by using the Quad Tree data-structure. The resulting communities identified by our model cannot be dominated by the other communities in the d-dimensional attribute space. Also, based on the user requirement, the solution can do an early exit when the specified number of groups are found, making the overall computation faster. Extensive experiments in both real-world and synthetic multi valued networks demonstrate the efficiency, scalability and effectiveness of our solutions.

7.2 Future Work

Here are the directions that can be followed to further extend the scope of this thesis:

- Our model works for both the k-core and k-clique dominance check. But in our work, we proposed the algorithm only for k-clique which can be extended for k-core as well. In our proposed algorithm, we need to calculate all the k-cliques in the network. Because of this, the algorithm can't be generalized for k-kore, as we didn't find any efficient solution where we can generate all the k-cores of a network.

- Before determining a group to be a skyline group, we compare the group with

every prior skyline groups. As this comparison is independent, it can be done in parallel which will make the solution faster.

- The current IsDominated (Algorithm 1) method has a quadratic time complexity in k. This method can be modified to run in parallel or have some pruning rules so that the time complexity isn't quadratic.

Bibliography

[1] Nicholas Assimakis and Maria Adam. A new author's productivity index: p-index. *Scientometrics*, 85(2):415–427, 2010.

[2] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430, 2001.

[3] Sean Chester, Michael Lind Mortensen, and Ira Assent. On the suitability of skyline queries for data exploration. In *Proc. ExploreDB*, pages 161–166, 2014.

[4] Sean Chester, Darius Šidlauskas, Ira Assent, and Kenneth Sejdenfaden Bøgh. Scalable parallelization of skyline computation for multi-core processors. In *Proc. ICDE*, pages 1083–1094, 2015.

[5] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 717–719. IEEE Computer Society, 2003.

[6] Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs*. WWW '18, page 589–598, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[7] Evangelos Dellis, Akrivi Vlachou, Ilya Vladimirskiy, Bernhard Seeger, and Yannis Theodoridis. Constrained subspace skyline computation, 2006.

[8] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 102–113, 2001.

[9] Mark S Granovetter. The strength of weak ties. *American Journal of Sociology*, 78(6):1360–1380, 1973.

[10] Jorge E Hirsch. An index to quantify an individual's scientific research output. *PNAS*, 102(46):16569–16572, 2005.

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

[12] Paul Jaccard. The distribution of the flora in the alpine zone. *The New Phytologist*, 11(2):37–50, 1912.

[13] H T Kung, Fabrizio L Luccio, and Franco P Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4):469–476, 1975.

[14] Chengkai Li, Nan Zhang, Naeemul Hassan, Sundaresan Rajasekaran, and Gautam Das. On skyline groups. In *Proc. CIKM*, pages 2119–2123, 2012.

[15] Rong-Hua Li, Lu Qin, Fanghua Ye, Jeffrey Xu Yu, Xiaokui Xiao, Nong Xiao, and Zibin Zheng. Skyline community search in multi-valued networks. In *Proc. SIGMOD*, page 457–472, 2018.

[16] Rong-Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. Influential community search in large networks. *Proc. VLDB Endow.*, 8(5):509–520, January 2015.

[17] Ming-Yen Lin, Yueh-Lin Lin, and Sue-Chen Hsueh. Discovering group skylines with constraints by early candidate pruning. In *Proc. Advanced Data Mining and Applications*, pages 49–62, 2017.

[18] Xuemin Lin, Yidong Yuan, Qing Zhang, and Ying Zhang. Selecting stars: The k most representative skyline operator. In *Proc. ICDE*, 2007.

[19] Jinfei Liu, Li Xiong, Jian Pei, Jun Luo, and Haoyu Zhang. Finding pareto optimal groups: Group-based skyline. *PVLDB*, 8(13):2086–2097, 2015.

[20] Michael Lind Mortensen, Sean Chester, Ira Assent, and Matteo Magnani. Efficient caching for constrained skyline queries. In *Proc. EDBT*, 2015.

[21] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progessive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.

[22] Yufei Tao, Xiaokui Xiao, and Jian Pei. SUBSKY: Efficient computation of skylines in subspaces. In *Proc. 22nd International Conference on Data Engineering (ICDE)*, 2006.

[23] Chen Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, Fan Zhang, and Xuemin Lin. Selecting the optimal groups: Efficiently computing skyline k-cliques. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 1211–1220, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Yuting Zhang, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. Pareto-optimal community search on large bipartite graphs. In *Proc. CIKM*, pages 2647–2656, 2021.

[25] Haoyang Zhu, Xiaoyong Li, Qiang Liu, and Hao Zhu. Computing skyline groups: An experimental evaluation. *Tsinghua Science and Technology*, 24(2):171–182, 2019.

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