Parallel Discovery of Fixed-sized Connected k-Core Skyline Communities

by

Parisa Esmaeilian Ghahroudi
B.Sc., Sharif University of Technology, 2020

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

MASTER OF SCIENCE

in the Department of Computer Science

Parallel Discovery of Fixed-sized Connected k-Core Skyline Communities

by

Parisa Esmaeilian Ghahroudi
B.Sc., Sharif University of Technology, 2020

Supervisory Committee

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

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Abstract

Graphs are powerful when it comes to representing complex relationships between objects, where nodes and edges represent entities and relationships between them respectively. In recent years, the concept of community structures in graphs has gained significant attention due to its broad applications in various fields, such as social media analysis, physics, biology, and more. Community structures represent groups of nodes that have close relationships with each other, providing valuable insights into the underlying relationships within the graph. Graph nodes are often associated with attributes that contain valuable information, and it would be informative to take them into account when looking for communities. One way to do so is through the use of skyline communities, which represent community structures of a graph that are pareto optimal with respect to attribute values of nodes.

In this study, we focus on k-Core subgraphs, where every node has a degree of at least k, and look for those holding skyline properties. We propose both sequential and parallel algorithms for discovering skyline k-Core subgraphs and perform experiments to investigate how input parameters, such as the dataset, the size of the community, the number of attribute dimensions, etc., affect the performance of our solution. Our proposed approach is a progressive algorithm that can be stopped at any point, providing the assurance that any output obtained is a skyline community. We demonstrate the effectiveness of our approach on a large dataset which is able to achieve acceleration rates as high as 10x over the state-of-the-art method. Moreover, the parallelised version attains super-linear acceleration rates with 2-3 cores (2.25x and 3.29x respectively) and a speedup as high as 34x over the sequential version when utilising 48 cores.

Keywords: Skyline, Community Discovery, k-Core, Parallel Computing

Table of Contents

List of Tables

List of Figures

List of Algorithms

Acknowledgments

Words cannot express my gratitude to my professor, *Dr. Sean Chester*, who generously provided knowledge and expertise. This endeavor would not have been possible without his invaluable patience and support.

I am also grateful to my spouse, *Nima Fathollahi*, who is one of the greatest inspirations in my life. I could not have undertaken this journey without his moral support.

Parisa Esmaeilian Ghahroudi,
Victoria, 8th March, 2023.

"You need to realise something if you are ever to succeed at chess. (...) And the thing you need to realise is this: the game is never over until it is over. It isn't over if there is a single pawn still on the board. If one side is down to a pawn and a king, and the other side has every player, there is still a game. And even if you were a pawn - maybe we all are - then you should remember that a pawn is the most magical peice of all. It might look small and ordinary but it isn't. Because a pawn is never just a pawn. A pawn is a queen-in-waiting. All you need to do is find a way to keep moving forward. One square after another. And you can get to the other side and unlock all kinds of power."

Matt Haig, The Midnight Library (2020)

Chapter 1

Introduction

We are constantly surrounded by datasets and we often rely on the information extracted from them to make decisions. In some cases, we need to select the best data points or records, such as choosing the most impactful researchers in a specific field to collaborate with, identifying the most profitable products to focus on, determining the most effective treatments for a particular disease, etc., applications are endless. When records are associated with only one attribute, we can sort the data points in ascending or descending order based on our preference and select the top m records. However, if the records have multiple features, sorting becomes more complex and it is no longer intuitive.

To address this issue, skyline points were introduced in [BKS01] as a way to filter out the best records from a large dataset and effectively reduce the decision-making space. To put it in simpler terms, skyline points refer to data points in a dataset that are not worse than any other record in the dataset when considering multiple attributes. Figure 1.1 demonstrates a simulated dataset of real state properties with two attributes, price and distance to downtown. The blue data points form the skyline set. Assuming we prefer low values for all attributes, any point in the blue region is worse than at least one other point. In the context of skyline language, when one point is superior to another point, it is expressed as the former dominating the latter. As a result, skyline points are those records that are not dominated by other points.

What if we are trying to find a group of points that is superior to any other group of the same size, in other words, groups that are not dominated by any other group? This is applicable in various scenarios such as creating a hockey team or selecting suppliers. One way to solve this is by forming a set consisting of only skyline points. However, this approach has two problems: firstly, we may not have enough skyline points to form such a set. Secondly, we overlook all sets that have at least one non-skyline point, some of which may be of interest. This led to the development of the concept of skyline groups, which was independently

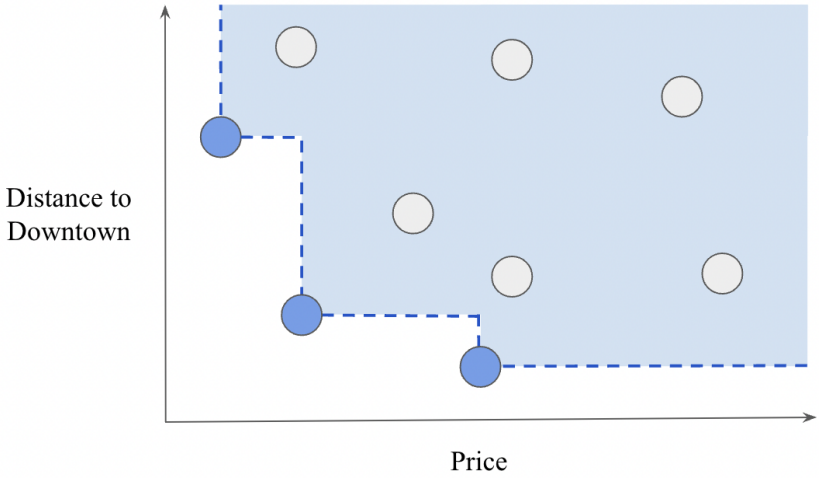


Figure 1.1: A Simulated Dataset of Real State Properties: Assuming we prefer low values for all attributes, any point in the blue region is worse than at least one other point. The blue points form the skyline set.

introduced by [IP12] and [Zha+14].

Skyline groups have been integrated into the study of community detection in graphs, which involve nodes and edges representing entities and relationships between them, respectively. Subgraphs containing nodes with strong connections to others are known as community structures and are significant as they reveal important insights about relationships within the graph. The two group-colored subgraphs in Figure 1.2, represent two possible cohesive subgraphs of the original graph which are locally densely connected.

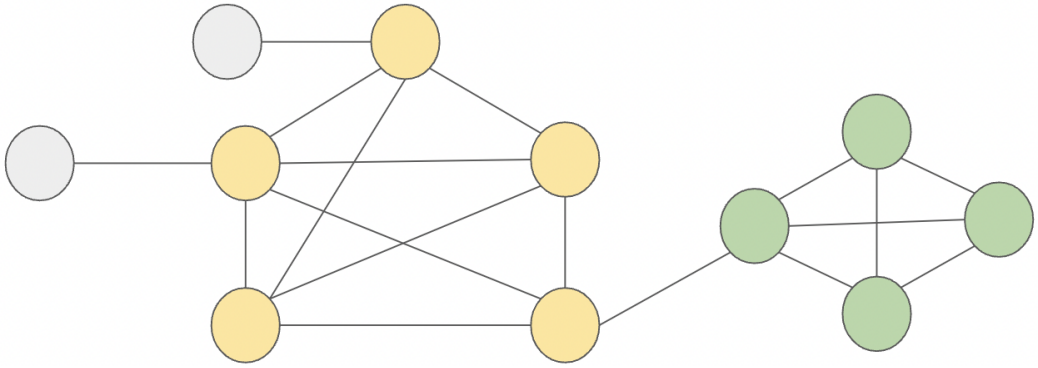


Figure 1.2: An Example of Cohesive Sub-communities of a Graph: These subgraphs are locally densely connected.

As an example, imagine a social network, where each person is represented

by a node, and if two people are friends, there is an edge between their nodes. In this network, a community is a group of people who know each other either directly (through a friendship edge) or indirectly (through a chain of friendship edges connecting them). A community is typically defined as a dense subgraph with short distances between its nodes, indicating a cohesive sub-community.

Community detection has diverse applications, including identifying influential individuals or groups, targeted marketing, and gaining insights into how diseases or information propagate through a network. In most graphs, nodes are associated with attributes that contain valuable information, but these attributes are often overlooked or simplified to a single attribute for convenience. To address this limitation, the concept of skyline groups was combined with community detection to create skyline communities, which are cohesive subgraphs that are optimal in terms of node attributes as well.

K-Clique and k-Core are two types of community structures commonly used in community discovery problems. A k-Clique is a subgraph of size k where every node is directly connected to all other nodes. A k-Core is a subgraph in which every node has a degree of at least k. This means that a k-Clique is equivalent to a (k-1)-Core of size k. K-Core subgraphs can be considered a more general version of k-Cliques. It is important to note that a k-Core may not necessarily be a connected graph. A k-Core that consists of two or more separate components with no connections between them does not represent a cohesive community. Therefore, when we talk about k-Core skylines, we specifically mean connected k-Core skylines.

[Li+18] investigated k-Core skylines that have the property of being maximal, meaning that there are no other k-Core skylines that contain them. The authors used a group-wise dominance definition that computes a representative for each group and then performs point-wise dominance tests. [LZY20] used the same dominance definition but proposed a new method for computing skyline k-Cores of a fixed size. However, this aggregation-based dominance definition has a limitation in that it discards the attribute value of all nodes and replaces them with a single point for each group. As a result, it may not be able to consider all the information in a group and may miss interesting groups.

[Zha+19] proposed a new method for discovering skyline k-Cliques using a different dominance definition. This definition involves one-to-one matching point-wise dominance tests to decide between two groups. [Akb22] also studied skyline k-Cliques but introduced a new group-wise dominance relationship called generalised dominance which compares two groups, U and U', by ignoring any shared elements and stating that U is dominated by U' if every point in U is dominated by at least one point in U'. The drawback of skyline k-Clique is that it imposes a strong constraint on groups, meaning that every node in a group must be connected to all other members.

In this work, we propose a novel technique for identifying fixed-size skyline k-Core communities based on the generalised group-dominance definition. We assume an upper-bound on the size of the communities such that this size is no greater than $2k+1$. In a k-Core community with a larger size, there may be nodes that are connected to fewer than half of all nodes. Such a community can, in fact, be two almost disconnected communities and we are not interested in discovering such unions of disjoint half-groups. Moreover, this size constraint helps with removing the overhead of checking if skyline groups are connected graphs and achieving better performance on listing skyline k-Cores which is described in more depth in Sections 3.3 and 4.2.5.

We choose k-Core structures as they represent cohesive communities while allowing for flexibility in node connectivity. By utilising the general group dominance definition, we are able to consider all attribute values of nodes when identifying interesting skyline groups. The discovery of skyline groups, particularly skyline communities, is a challenging and time-consuming task. Therefore, we also introduce a parallel algorithm to efficiently identify skyline communities in a given graph.

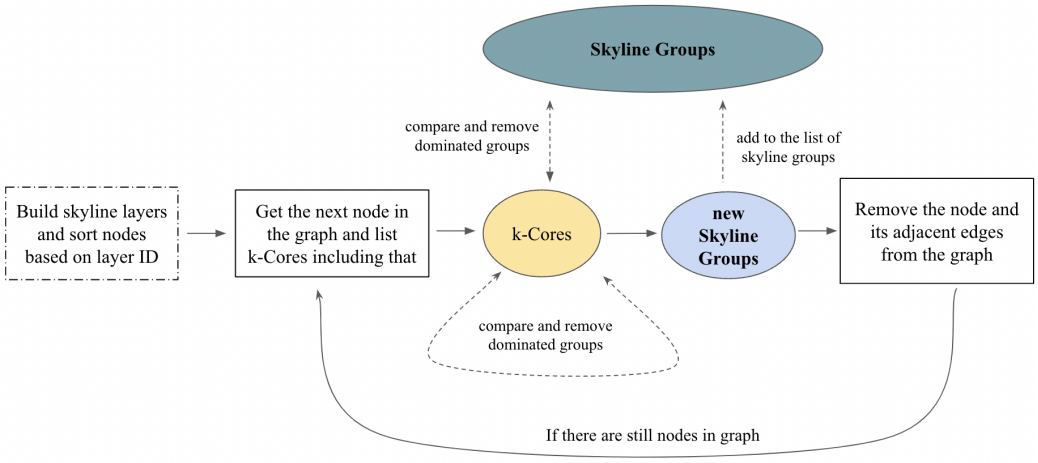


Figure 1.3: An Abstract Representation of the Proposed Solution

Our proposed approach is a progressive algorithm that can be terminated at any stage guaranteeing that all groups generated meet skyline properties. To begin, we sort the nodes in a particular order that ensures no point dominates its predecessors. This sort helps us skip performing unnecessary dominance tests and is discussed with more depth in section 4.2.1. Next, we go through the nodes and for each node, we generate all possible k-Core communities, compare them to each other and known skylines, and add any non-dominated groups to the list of results. To prevent the generation of duplicate groups, we eliminate

the accessed node and all adjacent edges from the graph once we generate all possible groups that contain that node. We also use various pruning rules and optimisation techniques to efficiently list k-Cores. The general process of the solution is illustrated in Figure 1.3.

In the parallelised version of the algorithm, nodes are processed in batches, with each thread handling one node. Each thread generates all possible k-Cores, including the node assigned to it, and compares them to each other and known skylines. Then, the threads share their results with each other through synchronization, allowing each thread to compare its local results with those of other threads and accurately identify skyline groups.

We conducted thorough experiments to investigate how our algorithm behaves considering various factors such as the number of attribute dimensions, graph sparsity, and community size. Generally speaking, increasing the number of dimensions and the community size leads to longer processing time. This is because there are more possible combinations of nodes that satisfy the k-Core requirements, which requires additional time to list and compare them. Similarly, we observed longer processing times when the gap between the community size and the minimum number of required connections (k) grows.

Our proposed method is able to achieve speedups up to 10x over the baseline (our adaptation for Definition 9 of [Zha+19]) for a large dataset. Additionally, we assessed the parallel scalability of our approach, which can attain super-linear and linear acceleration rates, depending specifically on the dataset sparsity and thread count. Our parallelised algorithm achieves a high acceleration rate of up to 34x on a large dataset when running on 48 threads.

1.1 Contributions

Given an undirected multi-variate vertex-labelled graph G, we make the following contributions in this study:

- A novel method to generate a list of connected k-Core subgraphs of graph G, inspired by [Con+18]: These subgraphs have a fixed size that is no greater than $2k + 1$.
- A novel algorithm that discovers the skyline k-Cores of graph G: We use the dominance definition presented in [Akb22] to identify skyline communities. Our proposed solution is able to achieve up to 10x speedup on a large dataset over the algorithm introduced in [Zha+19] (for identifying skyline k-Cliques).
- Parallelisation of our proposed solution: This parallel algorithm is able to

attain super-linear and linear acceleration rates on realistic sparse graphs over the sequential version.

1.2 Overview of the Following Chapters

Chapters 2 and 3 present a literature review on skyline related studies and provide background in both skyline and community structures respectively. In Chapter 4, we introduce our proposed solution for discovering skyline k-Core communities sequentially and in parallel. We also present our pruning rules and optimisation strategies that we took advantage of in this work. Chapter 5 is dedicated to our experiments conducted to evaluate the performance of our novel solution to skyline k-Core discovery. Finally, we make conclusions and describe how our work can be further expanded in Chapter 6.

Chapter 2

Related Work

Skyline operator was introduced in [BKS01] to filter out interesting d-dimensional points from a potentially large dataset. An interesting point is defined as a point with optimal attribute values in one or more dimensions. This problem was formerly known as the maximum vector problem ([KLP75]). The skyline points of a dataset are extracted by performing dominance tests on pairs of data points and deciding if one dominates the other. The non-dominated points form the skyline set.

As conducting dominance tests on every pair of points is expensive, many optimisation techniques are introduced to reduce the number of dominance tests needed to identify skyline points of a dataset. [Cho+03] showed that presorting data points such that a point may be dominated by only its preceding points, speeds up the computation since two-sided dominance tests can safely be replaced with one-sided ones. [ZMC09] and [LH09][LH10] independently introduced incomparability between points that do not dominate one another. In high-dimensional space, most point pairs become incomparable. This is used as a key factor for removing unnecessary dominance tests by partitioning the space into 2^d subregions, where d denotes the number of dimensions, and skipping dominance tests between points of certain subregions.

Skyline groups were introduced independently by [IP12] and [Zha+14] to answer queries including sets of points that are pareto optimal. The group-wise dominance relationship mostly belongs to one of the two following categories:

1. Aggregation-based where a representative point is calculated for each group and used to perform point-wise dominance tests as in [Li+15][Li+18].

2. Permutation-based, introduced in [Liu+15], where one possible permutation of each group is used to conduct one-to-one matching point-wise dominance tests.

A more general dominance definition is introduced by [Akb22] which requires point-wise dominance tests between group members but does not limit them to one-to-one matchings.

Community discovery has caught attention in recent years and has applications in social network analysis, physics, biology, etc. A community is usually defined as a dense subgraph in which the distance between any two nodes is short enough so the subgraph represents a cohesive sub-community. However, in many community discovery studies attribute values of nodes are ignored or reduced to only one attribute to simplify the problem. To address this issue, the skyline group concept is combined with multi-valued graphs in search of pareto optimal groups which represent cohesive sub-communities as well. [Zha+19] introduced a new algorithm to discover skyline k-Cliques (complete subgraphs of size k) under the permutation-based group dominance. [Akb22] also visited skyline k-Cliques under the general group dominance defintion described earlier. [Li+18] and [LZY20] studied skyline k-Cores, a generalised version of k-Cliques, and took advantage of an aggregation-based group dominance. [LZY20] focused on identifying k-Core skylines of a fixed size, whereas [Li+18] aimed to find maximal k-Core skylines, those that are included in any other k-Core skyline.

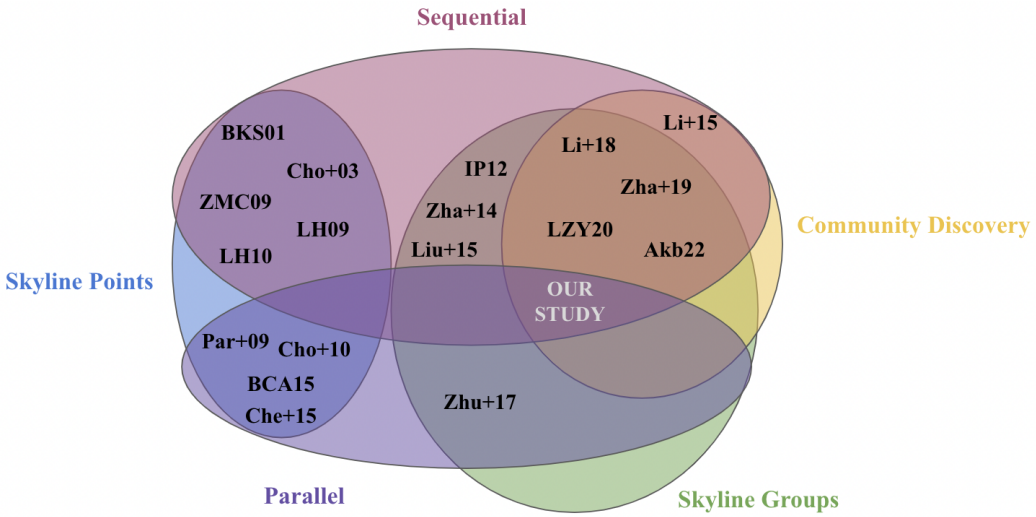


Figure 2.1: A Venn Diagram of Conducted Studies Related to Skyline

The processing of skyline queries, and in particular the discovery of skyline groups, is a complex task that is often associated with long processing times. Consequently, the implementation of multi-threaded and parallelised techniques of the relevant algorithms is a topic of significant interest. A new technique is introduced in [Cho+10] for vectorising dominance tests using SIMD instructions. [Par+09] presented a multi-core algorithm to compute the skyline set of a dataset

Group Dominance \ Community	k-Clique	k-Core
Aggregation-based	-	Li+18, LZY20
Permutation -based	Zha+19	-
General	Akb22	OUR STUDY

Table 2.1: Studies Related to Skyline Community Discovery: Note that all techniques presented for discovering k-Core subgraphs can handle k-Cliques as well.

in parallel. This method divides the dataset into smaller portions, handles each segment on an individual core, and subsequently combines the outputs. [Che+15] also introduced a novel skyline multi-core algorithm, *Hybrid*, which processes points in blocks instead. This technique is revisited in [Zhu+17] to introduce a new method for discovering skyline groups under permutation-based group dominance. [BCA15] also presented a new parallel algorithm designed for the GPU to speed up the computation of skyline points.

Figure 2.1, depicts how the referenced studies fit into skyline literature. In this study, we are interested in skyline k-Cores under the general group-wise dominance definition introduced in [Akb22]. We choose this dominance definition since given two groups of the same size, it is more probable that one dominates the other under the generalised group dominance and as a result, fewer skyline groups are generated. Smaller output size is favorable in many applications since a large number of skyline groups may not aid in effectively reducing the decision-making space. We present novel algorithms to discover such skyline groups both sequentially and in parallel. Table 2.1 highlights the details of studies in the area of skyline community discovery.

Chapter 3

Preliminaries

The problem addressed by this study, finding *best* cohesive communities of a vertex-labelled graph, requires knowledge in both skyline and community structure. This section provides background on both topics (Sections 3.1 and 3.2) and presents our problem definition accordingly (Section 3.3).

3.1 The Skyline Operator

The skyline operator is used to extract the best points or best sets of points of a d-dimensional dataset. This is widely used when lower/higher attribute values are preferred as it narrows down the search space effectively. In this work, we prefer lower values for all attributes of a given dataset though it is trivial to adapt the work for a preference towards larger values.

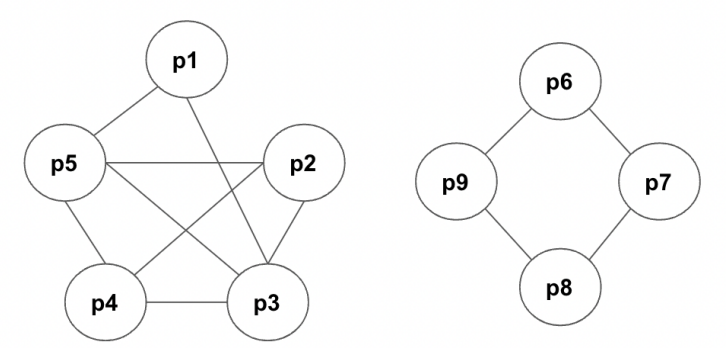
Definition 1. (Point Dominance)[KLP75][BKS01] Given two points u and v in d-dimensional space, where u^i denotes the value of i^{th} attribute, we say u dominates v, denoted by $u \prec v$, iff $u^i \leq v^i$ for all $i \in [1, d]$ and there is a dimension $j \in [1, d]$ with $u^i < v^i$. We use $u \preceq v$ to denote u dominates or equals v.

Given a dataset, the set of points that are not dominated by other points is called the skyline set.

Definition 2. (Skyline)[BKS01] Given a dataset P, the set $\{u \in P \mid \nexists v \in P, v \prec u\}$ is the skyline set of P and is denoted by $SKY(P)$.

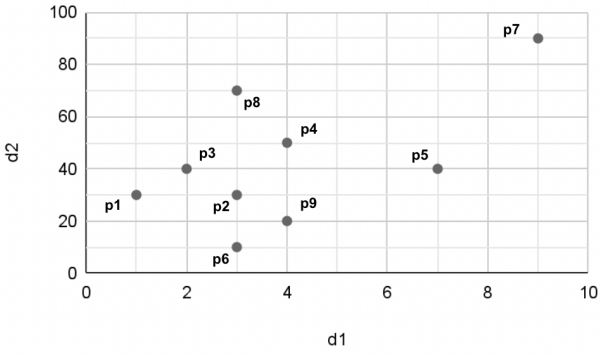
Using the skyline operator (SKY) over a dataset multiple times leads to the skyline structure defined below.

Definition 3. (Sky Layers)[Cha+00][Liu+15] Given a set of points P in d-dimensional space, the skyline layers are denoted by L_i where i presents



(a) Undirected Graph

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) Attribute Values of Nodes

Figure 3.1: An Example of an Undirected Multivariate Vertex-labelled Graph

the layer number and layers are defined as follows. $L_1 = SKY(P)$ and $L_i = SKY(P \backslash \bigcup_{j=1}^{i-1} L_j)$. The layer number of point p is denoted by $\lambda(p)$.

Note that each $p \in P$ belongs to a unique layer. We define the parent set of a node p, as the set including all points that dominate p.

Example 1. *In Figure 3.1b, $p_6 \prec p_2$ while $p_6 \not\prec p_1$. $\{p_1, p_6\}$ is the set of skyline points and Figure 3.2 depicts the skyline layers of the dataset.*

Definition 4. (Parent Set)[Liu+15] Given a set of points P, the parent set of $p \in P$ is denoted by $\rho_p = \{p' \in P \,|\, p' \prec p\}$.

Example 2. *In Figure 3.1b, parent set of p_1 is empty while parent set of p_4 consists of $\{p_1, p_6, p_2, p_3, p_9\}$.*

The skyline layers shown in Figure 3.2 clearly shows that a node p with $\lambda(p) = i$ cannot dominate any node p' with $\lambda(p) \leq i$. The following property is established based on this.

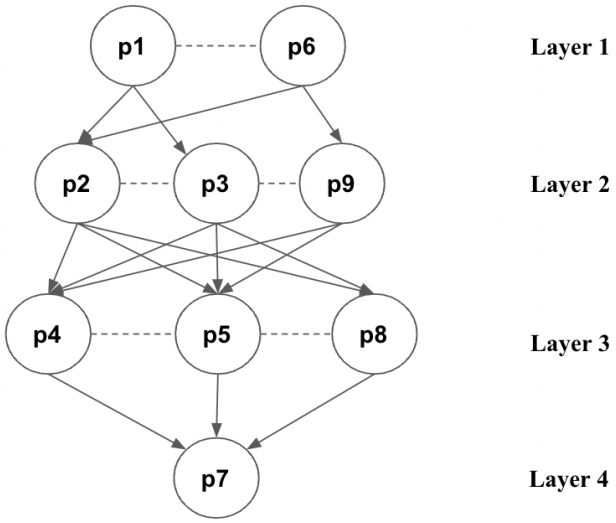


Figure 3.2: Skyline Layers of the Dataset Shown in Figure 3.1b: Node u points to node v if u dominates v.

Property 1. Points in the parent set of point p, belong to layers preceding the layer to which point p belongs.

$$\forall p' \in \rho_p,\ \lambda(p') < \lambda(p)$$

3.1.1 Skyline Groups

In search of the best sets of points of a dataset, [IP12] and [Zha+14] independently introduced skyline groups as groups of a certain size which are not dominated by any other group of the same size and are denoted by Sky.

Definition 5. (Skyline Groups)[IP12][Zha+14] Given a dataset P and a size g, the skyline groups are defined as follows:

$$G = \{\{p_1, p_2, ..., p_g\} \,|\, p_i \in P, i \in [1, g]\}$$
$$Sky = \{S \in G \,|\, \nexists S' \in G, S' \prec S\}$$

There are a number of definitions introduced for group dominance in the literature. Below we present a number of them. Before getting into that, we give definitions for MIN and MAX representatives of a set of d-dimensional points.

Definition 6. (MIN, MAX, SUM representatives) Given a set of d-dimensional points, P, MIN representative, denoted by P^-, is a d-dimensional point where $P_i^- = min\{p_i \,|\, \forall p \in P\}$ for all $i \in [1, d]$. Similarly, MAX representative is denoted by P^+, which is a d-dimensional point with $P_i^+ = max\{p_i \,|\, \forall p \in P\}$ for

all $i \in [1, d]$. SUM representative is denoted by P^s, which is a d-dimensional point with $P^s_i = \sum_{p \in P} p_i$ for all $i \in [1, d]$.

Example 3. *In Figure 3.1a, MAX representative of group $\{p_4, p_5, p_8\}$ is (7, 70). The MIN representative of the second layer in Figure 3.2, ($\{p_2, p_3, p_9\}$), equals to (2, 20). The SUM representative of group $\{p_1, p_2\}$ is (4, 60).*

The *Aggregation Group Dominance*, suggested in [IP12], uses any monotone function for aggregation, however, [IP12] concentrates specifically on utilising MIN, MAX, and SUM. MIN and MAX aggregate the groups against the direction of optimisation and avoid groups with weak nodes that have high values for one or more attributes (Definition 7 assuming we prefer lower attribute values of points). This group-wise dominance definition is revisited in [Li+15][Li+18] in search of optimal and influential communities.

Definition 7. (Aggregation Group Dominance)[IP12] Let P be a set of points and $U, U' \subseteq P$ of size g. U group-dominates U' iff $MAX(U) \prec MAX(U')$.

Example 4. *Let $U = \{p_1, p_2, p_3\}$ and $U' = \{p_2, p_4, p_6\}$ be groups of size 3 in Figure 3.1b. $MAX(U) = (3, 40)$ and $MAX(U') = (4, 50)$. $MAX(U) \prec MAX(U')$ and thus $U \prec U'$.*

The *Permutation Group Dominance* is introduced by [Liu+15] and proposed by [Zhu+17] and [Zha+19]. This definition is based on one-to-one matching point-wise dominance tests given one specific permutation of a group.

Definition 8. (Permutation Group Dominance)[Liu+15] Let P be a set of points and $U, U' \subseteq P$ of size g. U group-dominates U' iff there is a one-to-one matching dominance, $U = \{u_1, u_2, ..., u_g\}, U' = \{u'_1, u'_2, ..., u'_g\}$, such that $u_i \preceq u'_i$ for all $i \in [1, g]$ and $u_i \prec u'_i$ for at least one i.

Example 5. *Let $U = \{p_1, p_2, p_6\}$ and $U' = \{p_3, p_5, p_8\}$ be groups of size 3 in Figure 3.1b. There exists a one-to-one dominance matching, $\{p_1, p_2, p_6\}$ and $\{p_3, p_8, p_5\}$ where $p_1 \prec p_3$, $p_2 \prec p_8$, $p_6 \prec p_5$. Thus, $U \prec U'$.*

In this study, we use the definition introduced in [Akb22], *Generalised Group Dominance* which favours groups with at least one strong vertex that cannot be dominated.

Definition 9. (Generalised Group Dominance)[Akb22] Let P be a set of points and $U, U' \subseteq P$ of size g. U group-dominates U' iff for all points u' in U' but not in U, exists at least one point u in U but not in U' such that $u \prec u'$.

$$U \prec U' \text{ iff: } U \neq U' \text{ and } \forall u' \in U' \backslash U, \exists u \in U \backslash U', u \prec u'$$

Example 6. *Let $U = \{p_4, p_6, p_8\}$ and $U' = \{p_4, p_5, p_9\}$ be groups of size 3 in Figure 3.1b. $p_6 \prec p_5$ and $p_6 \prec p_9$, i.e., all elememts of $U' \backslash U$ are dominated by at least one element of $U \backslash U'$. Thus, $U \prec U'$. This is depicted in Figure 3.3. Note that there is no one-to-one dominance matching between U and U' and thus U' is not dominated by U under permutation group dominance (Definition 8).*

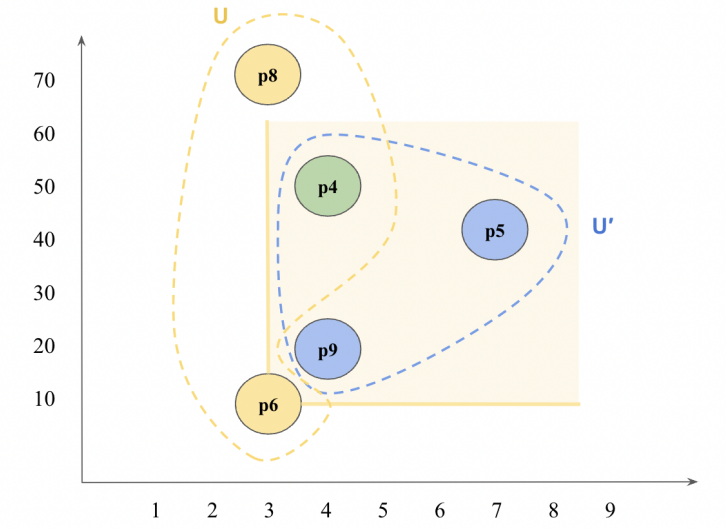


Figure 3.3: An Example of Generalised Group Dominance (Definition 9): U dominates U' since $p_6 \prec p_5$ and $p_6 \prec p_9$.
Note that p_4 is ignored as it is a common element.

Generally, given two groups of the same size, it is more likely that one dominates the other under the generalised group dominance (Definition 9) than the permutation group dominance (Definition 8) and as a result fewer skyline groups are generated. The reason is that, a strong point in a group may dominate all members of another group under generalised definition where this point is limited to a one-to-one matching with only one point in another group having permutation-based definition. Smaller output size is favorable in many applications since a large number of skyline groups may not be helpful with reducing the decision-making space effectively.

We describe below the analysis of [Akb22] on the toy example given in Figure 3.4 to highlight the strengths and weaknesses of each model. This demonstrates why we select the generalised model over the other dominance models. Specifically, [Akb22] compares:

- SUM (Definition 7): The skyline group introduced by [Zha+14; IP12], using the direction of preference for aggregation.
- MAX (Definition 7): The skyline group introduced by [Zha+14; IP12], using the opposite direction of preference for aggregation.
- PERMUTE (Definition 8): The g-skyline introduced by [Liu+15].
- GENERAL (Definition 9): The modification of the g-skyline ([Liu+15]) that [Akb22] presented to improve the selectivity.

[Akb22] uses a toy example to generate skyline 3-Cliques (Figure 3.4). This example includes groups with the following characteristics:

- G1: It contains all skyline points some of which are not connected to one another ($\{$b,c,f$\}$).
- G2: It includes skyline points as well as one weak point (brown points).
- G3: It includes weak points as well as one skyline point (pink points).
- G4: Nodes in this group are all moderately superior to the weakest point of the anticipated skyline group (yellow points).
- G5: The attribute values of all points in this group are slightly better than the corresponding worst values of the anticipated skyline group (orange points).
- G6: This average group includes points with a mixed set of attributes where some are excellent while others are poor (green points).

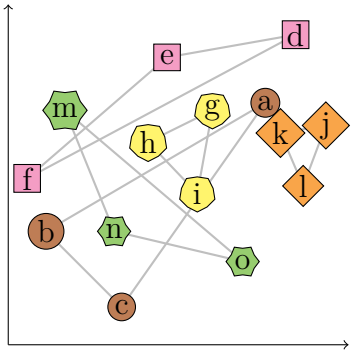
As mentioned by [Akb22], the only groups including skyline points are G2 and G3. Any point in G4 and G5 is dominated by at least one point in G3 and/or G6. G2 and G5 include notably weak points. The examination of the groups produced by each model leads to the following results:

GENERAL: produces G2 and G3. This model prefers groups including at least one strong point. As a result of this, it may generate groups containing weak points (G3).

SUM: produces G3 and G6. It prefers groups including fairly good points and may miss the majority of the best points (G2).

MAX: produces G4 and G5. It avoids groups including weak points and as a result, may not take into account the good points included in a group (G2, G3). In this toy example, it does not output any group including a skyline point.

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



Group Dominance Model	Produced Groups	Excluded Groups
GENERAL	G2, G3	G4, G5, G6
SUM	G3, G6	G2, G4, G5
MAX	G4, G5	G2, G3, G6
PERMUTE	G2, G3, G4, G5, G6	-

Figure 3.4: A Reproduction With Permission of Figure 3.1 From [Akb22] This example is reproduced according to the preference of lower values in this study providing a group-dominance model comparison on a toy dataset.

PERMUTE: produces all groups.

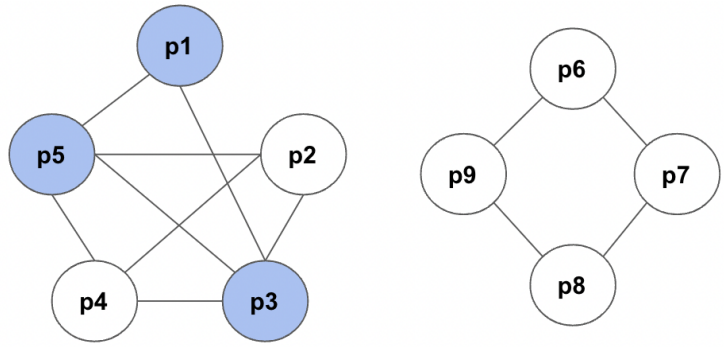
In summary, the SUM model favours fairly good points while the MAX model aims to avoid the weakest points as much as possible. The GENERAL model is designed to identify groups with strong points. The PERMUTE model does not have a particular preference and in this example, it produces the combination of all other models.

3.2 Community Structure in Graph

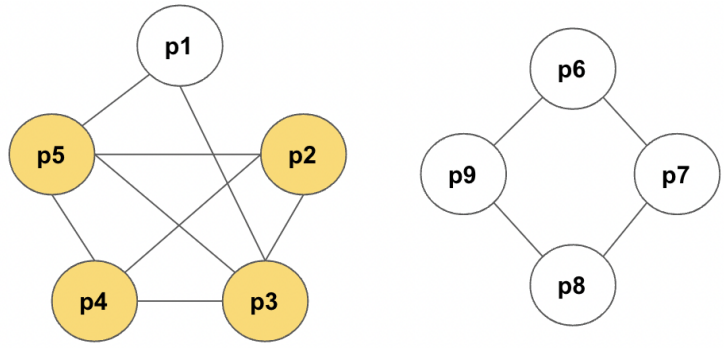
In this work, we study undirected graphs with no self loops. Community structures of a given graph are those vertex-induced subgraphs that are strongly interconnected. A vertex-induced subgraph of a given graph G is denoted by $H_G = (V_H, E_H)$ where $V_H \subseteq V_G$ and $E_H = \{(u,v) \,|\, u \in V_H, v \in V_H, (u,v) \in E_G\}$. *K-Cliques* and *K-Cores* are two widely used definitions for cohesive communities which are described below. δ is used to denote the degree of a node in graph.

Definition 10. (k-Clique) Given a graph $G = (V, E)$, a k-Clique is an induced subgraph $H_G = (V_H, E_H)$ of G such that $|V| = k$ and $\delta_{v \in V_H} = k - 1$. In other words, any two nodes in H are connected.

Examples of k-Cliques are presented in Figure 3.5.



(a) A 3-Clique



(b) A 4-Clique

Figure 3.5: An Example of k-Cliques

Definition 11. (k-Core) Given a graph $G = (V, E)$, a k-Core is an induced subgraph $H_G = (V_H, E_H)$ of G such that $\delta_{v \in V_H} \geq k$.

Note that a k-Clique is equivalent to a (k-1)-Core of size k. In this study, we will be using k-Cores as they are more general. Figure 3.6 illustrates two, green and yellow colored, 2-Core subgraphs. We denote the set of all g-size k-Core subgraphs of a given graph G by $C_{k,g}(G)$:

$$C_{k,g}(G) = \{C \subseteq G \,|\, \forall u \in C,\, \delta_u \geq k,\, |C| = g\}.$$

Definition 12. (Connected k-Core) A k-Core in which exists a path between every two nodes.

Example 7. *In Figure 3.6, the subgraphs colored in green and yellow are connected 2-Cores. Note that the yellow subgraph remains a 2-Core if p_1 joins the nodes as well. The larger subgraph including both yellow and green subgraphs presents a non-connected 2-Core.*

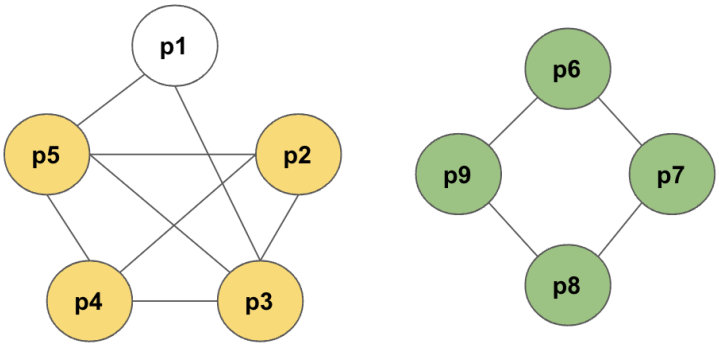


Figure 3.6: An Example of 2-Cores

A non-connected k-core community includes at least 2 connected components each including at least $k+1$ nodes to meet the degree requirements, resulting in a minimum of $2k+2$ nodes in total. This will result in the following theorem.

Theorem 1. Any k-Core of size g where $g \leq 2k+1$ is a connected k-Core.

The definition of the diameter of a graph, which is a useful metric for determining graph density, is as follows.

Definition 13. (Diameter) The diameter of a graph is the length of the longest shortest path between any two nodes.

Note that in a graph with a diameter of less than 2, every pair of nodes are connected and therefore is a Clique.

Any connected k-Core with less than $2k+2$ nodes has a diameter that is equal to or less than 2 if no self loops are allowed. By contradiction suppose that there are nodes u and v in a k-Core of size $2k+1$ or less where the shortest path between u and v is greater than 2. u is connected to a minimum of k nodes ($W = \{w_1, w_2, ..., w_n\}$, $k \leq n$). v must not be connected to any $w \in W$, otherwise the length of the shortest path between u and v would be 2. Therefore, v is not connected to a minimum of $k+2$ nodes (v, u, and any $w \in W$). Thus, there are at most $(2k+1) - (k+2) = k-1$ nodes left with possibility of connections to v. So, the graph cannot be a k-Core as v has a degree less than k.

Property 2. A connected k-Core with a size of $2k+1$ or less and no self loops has a diameter that is no greater than 2.

Definition 14. (n-Plex) Given a graph $G = (V, E)$, a n-Plex is an induced subgraph $H_G = (V_H, E_H)$ of G such that $\delta_{v \in V_H} \geq |V| - n$.

Based on the above definition, any node in a n-Plex can miss at most n connections within the graph. This leads to the following property as presented by [LZY20].

Property 3. A k-Core of size g is equivalent to a (g-k)-Plex of size g.

If no self loops are allowed, then a k-Core of size g is a (g-k-1)-Plex of the same size.

3.3 Problem Definition

Skyline k-cliques were studied in [Zha+19] [Akb22] using dominance Definitions 8 and 9 respectively. K-Core subgraph is a more general concept than k-Clique and therefore less limiting since the size of a k-Clique is directly driven from the degree constraint while in a fixed-size k-Core the connectivity requirement (k) is completely independent of its size.

In this work, we are interested in cohesive sub-communities defined as k-Cores which are optimal with respect to *generalised group dominance* (Definition 9). To our knowledge, there is no other study investigating skyline k-Cores using any of the two mentioned group dominance definitions.

Problem Definition. Given a multi-value vertex-labelled graph $G = (V, E)$, integers k and g, output the set of non-dominated connected k-Core communities of size g under *Generalised Group Dominance* definition where $g - k \leq k + 1$.

$$\text{Skyline Communities} = \{S \in C_{k,g}(G) \mid \nexists S' \in C_{k,g}(G),\ S' \prec S\}$$

We assume an upper-bound on the size of the communities, g, such that this size is no greater than $2k + 1$. We do not need to check if such k-Core is connected since based on Theorem 1, a k-Core subgraph with a size less than or equal to $2k + 1$ is always connected.

Note that in the case of $g - k = 1$, the problem will be discovering non-dominated k-Cliques which is studied in [Zha+19][Akb22] under two different group dominance definitions.

Chapter 4

Proposed Approach

In this chapter, we explain our solution for discovering fixed-size skyline groups given an undirected vertex-labelled graph. Broadly speaking, we iterate over nodes in order of the skyline layers, generate k-Cores, identify those that are skylines, and output them as results. This involves a method of fulfilling the connectivity requirements first and then performing a skyline query within the smaller and refined search area. We present an overview and a detailed description of the proposed approach in Sections 4.1 and 4.2 respectively.

4.1 Overview

We outline the general flow of our solution in Algorithm 1. First, we perform a preprocessing step and reorder nodes based on layer IDs. We follow techniques provided in [Zhu+17] to build the skyline layers. The primary objective of the preprocessing step is to avoid dominance tests between every pair of groups and enable early termination of the algorithm, pruning rules, and optimisations having a certain access order.

We then iterate through nodes and at each step, list k-Cores including the accessed node and compare those to known skylines (results generated at earlier iterations) as well as peers (newly generated groups). Having the comparisons, we identify non-dominated groups and add them to the results. At the end of each iteration, the accessed node is removed to avoid generating duplicate results. In addition, we present a technique for parallelising the primary *for* loop by incorporating synchronisation steps to manage the dependencies between loop iterations. The general process of the solution is illustrated in Figure 4.1 and detailed explanations of all steps are provided in Section 4.2.

Algorithm 1 Skyline-K-Cores

Input: undirected graph G, size k, size g
Output: set of skyline k-Cores

$Sky \leftarrow \{\}$
build skyline layers and sort nodes in G based on layer ID
// sequentially or in parallel, section 4.2.6
for $u \in G$ **do**
    generate k-Cores of size g including u *// section* 4.2.5
    remove k-Cores dominated by known skylines or peers
    add remaining k-Cores to Sky
    remove u from G
return Sky

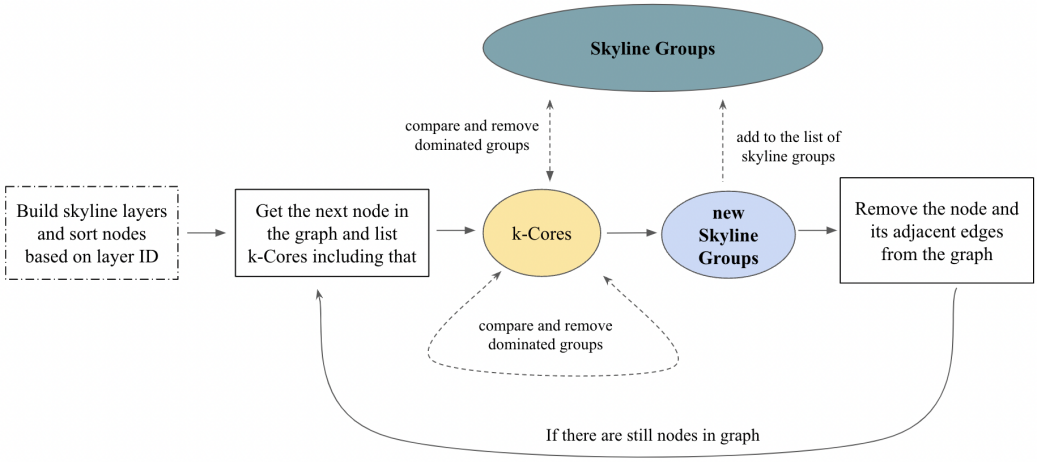


Figure 4.1: An Abstract Representation of the Proposed Solution

4.2 Algorithm Description

In this section, we explain the order in which nodes are accessed as well as the order in which results are produced (Sections 4.2.1 and 4.2.2). The process of ordering serves as an inexpensive examination that assists in bypassing several dominance tests. This is a customisation of a method widely used in skyline literature since [Cho+03].

We describe pruning rules, optimisation techniques, and early termination conditions that have been used earlier in [Akb22][Zha+19] and also introduce new ones used in this work (Sections 4.2.3 and 4.2.4). In the end, we present our algorithm to list k-Cores of a certain size and our progressive algorithm for generating skyline k-Cores under *generalised group dominance* (Definition 9) in

detail (Sections 4.2.5 and 4.2.6). Moreover, we present a parallelised version of this approach to speed up and enhance the efficiency of the original implementation. This parallelised algorithm takes advantage of the parallel computing capabilities of modern computer systems to perform multiple tasks simultaneously which leads to faster execution times as well as improved overall performance (Section 4.2.6).

4.2.1 Access Order

In [Cho+03], it is shown that utilising an access order based on a function that satisfies the condition where a point can only be dominated by those preceding it, can greatly speed up the computation of skyline by skipping half of the point-wise dominance test. Moreover, the sort order gives priority to points that have a higher likelihood of dominating other points.

The use of Manhattan Norm ($L1$ norm) is a commonly employed approach for achieving such ordering. For any pair of d-dimensional points, p_1 and p_2, if $L1(p_1) < L1(p_2)$, there is at least one dimension, i, where $p_1^i < p_2^i$ and thus $p_2 \not\prec p_1$. Inspired by [Zha+19], to list k-Cores, we access nodes in the multi-valued attributed graph following an order derived from the sky layers which holds the same characteristic based on Property 1.

Let u and v be two nodes in graph G where u precedes v in the access order, denoted by $o(u) < o(v)$. Let $C_{k,g}(u)$ denote the set of k-Cores of size g generated when node u is accessed. $C_{k,g}(u)$ are generated such that $u \notin C'$ where $C' \in C_{k,g}(v)$. In other words, when accessing a node, we use that node and only its following nodes to generate k-Cores. This will remove the possibility of generating duplicate groups. The reason is that, when a node is accessed, all possible groups containing that node are generated and then it is safely ignored in the rest of the computations.

Moreover, as a result of this, there is no $C' \in C_{k,g}(v)$ such that $C' \prec C$ where $C \in C_{k,g}(u)$, i.e., given the access order, k-Cores generated at any step do not need to be compared to those generated at prior steps. This is driven from Property 1, as there is no $v' \in C'$ such that $v' \prec u$.

Theorem 2. [Zha+19] Given two nodes $u, v \in G$, suppose $o(u) < o(v)$. For two k-Cores C and C' where $C \in C_{k,g}(u)$, $C' \in C_{k,g}(v)$, $C' \not\prec C$.

Given Theorem 2, we can perform one-sided dominance tests and safely skip half of the group-wise tests. Moreover, the algorithm runs progressively and can be terminated at any step knowing that groups generated until then are skyline groups ([Pap+05]).

4.2.2 Output Order

Let M and M' be two groups of size g and there is at least one node m in M for any node m' in M' such that m precedes m' in the access order ($o(m) < o(m')$). This is denoted by $O(M) < O(M')$ and [Akb22] claims that M cannot be dominated by M'.

Theorem 3. [Akb22] Let M and M' be two groups of size g. If $\exists m \in M$ where $\forall m' \in M'$, $o(m) < o(m')$, then $M' \not\prec M$.

The reason is that based on Property 1, $\nexists m' \in M'$ such that $m' \in \rho_m$. So there exists at least one node in M (m) that it is not dominated by any nodes in M' and as a result based on Definition 9, M' cannot dominate M.

In this study, we generate groups in a way that for any two groups M and M', M is generated before M' if $O(M) < O(M')$, i.e., a group cannot be dominated by any other group generated after that. Based on this, we will be using the following optimisation strategy as in [Akb22].

Optimisation 1. [Akb22] Generate groups such that M precedes M' in the output order if $O(M) < O(M')$ and safely skip the group dominance test of M' against M based on Theorem 3.

This helps us skip half of the dominance tests between new groups generated at each step.

4.2.3 Pruning Rules and Optimisation Strategies

This section describes methods to improve the efficiency of listing k-Cores, through reducing the search space. A set of pruning rules and optimisation techniques are used to do so which are summarised in Algorithm 2. Note that pruning rules involve skipping the processing of a node, while optimisation strategies aim to improve the processing of a node.

Given a graph G, not all nodes in G are qualified to be a part of k-Core subgraphs of G. [Zha+19] used a number of pruning rules to safely skip processing node $u \in G$ if it is not a qualified node as we describe below. First, we present a number of definitions regarding k-Cliques and k-Cores and then present the pruning rules.

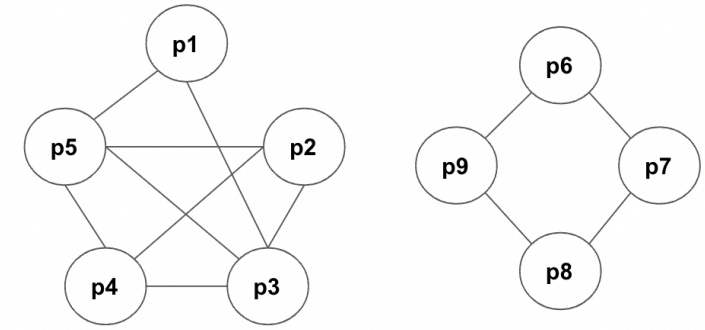
A maximal clique is a clique that cannot have any additional nodes added to it without violating the connectivity requirements. The maximum clique is the largest maximal clique of a given graph. Note that a maximum Clique is always a maximal Clique but the opposite is not necessarily true.

Definition 15. (Maximal Clique) Given a graph G, a maximal clique subgraph of G is a clique that is not a subset of a larger clique in G.

Algorithm 2 Skyline-K-Cores (pruning rules and optimisations included)

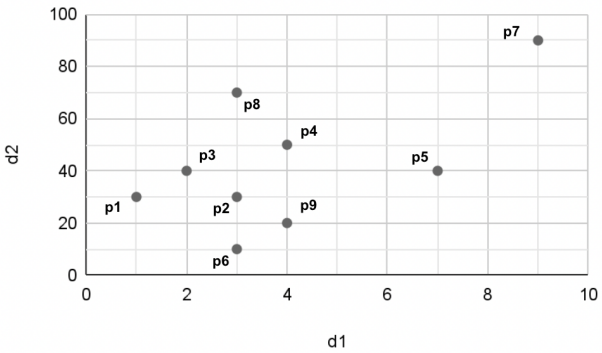
Input: undirected graph G, size k, size g
Output: set of skyline k-Cores

1: $Sky \leftarrow \{\}$
2: build skyline layers and sort nodes in G based on layer ID
3: $K \leftarrow$ maximum k-Core of G // *Pruning Rule 3*
4: **for** $u \in K$ **do**
5: $\tau(u) \leftarrow$ connected component including u
6: generate k-Cores of size g using $\tau(u)$ // *Optimisation 2*
7: remove k-Cores dominated by known skylines or peers
8: add remaining k-Cores to Sky
9: remove u from G and update K as a k-Core
10: return Sky



(a) Undirected Graph

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) Attribute Values of Nodes

Figure 4.2: An Example of an Undirected Multivariate Vertex-labelled Graph

Definition 16. (Maximum Clique) Given a graph G, the maximum clique of G is the maximal clique of G with the largest size.

Example 8. *In Figure 4.2a, $\{p_1, p_3, p_5\}$ and $\{p_2, p_3, p_4, p_5\}$ are maximal cliques of size 3 and 4 respectively. The latter is the maximum Clique of G as well.*

Given a graph G, the maximum k-Core is defined as the largest subgraph of G satisfying k-Core requirements. The core number of each node in G is defined based on the size of the largest k-Core it is included in.

Definition 17. (Maximum k-Core) Given a graph G, the maximum k-Core subgraph of G is the largest subgraph in which every node has a degree of at least k.

Definition 18. (Core Number) Given a graph G, the core number of node $u \in G$, denoted by $core(u)$, is the largest k such that u is in a k-Core subgraph of G.

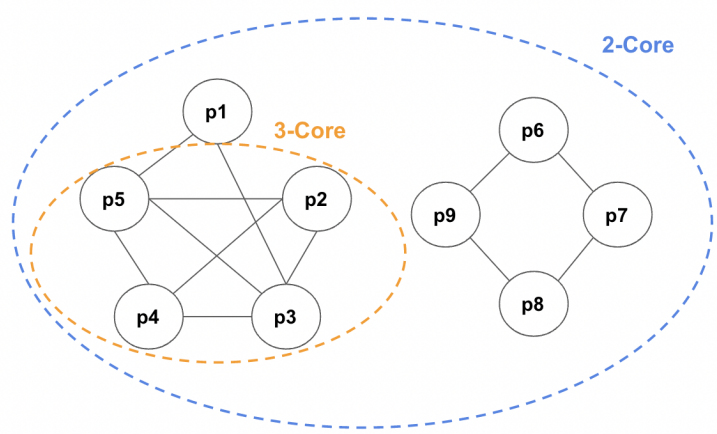


Figure 4.3: An Example of k-Core Decomposition of a Graph

Example 9. *In Figure 4.3, the whole graph is the maximum 2-Core while the subgraph induced on $\{p_2, p_3, p_4, p_5\}$ is the maximum 3-Core. Based on this, core numbers of p_2, p_3, p_4 and p_5 equal 3 while the core number for other nodes is equal to 2.*

Based on Definition 18, a node with a core number of k can only be a part of any k'-core where $k' \leq k$. This will result in the following pruning rule.

Pruning Rule 1. [Zha+19] Given a node $u \in G$, if $core(u) < k$, we can safely skip generating $C_{k,g}(u)$ as this will be an empty set.

When accessing node u, u and its following nodes are used to generate k-Cores. If u does not have at least k neighbours among these nodes, it will not be a part of any k-Core leading to the following pruning rule.

Pruning Rule 2. [Zha+19] Let $\Delta(u)$ denote the neighbours of u following the access order. If $\Delta(u) < k$, we can safely skip generating $C_{k,g}(u)$ as this will be an empty set.

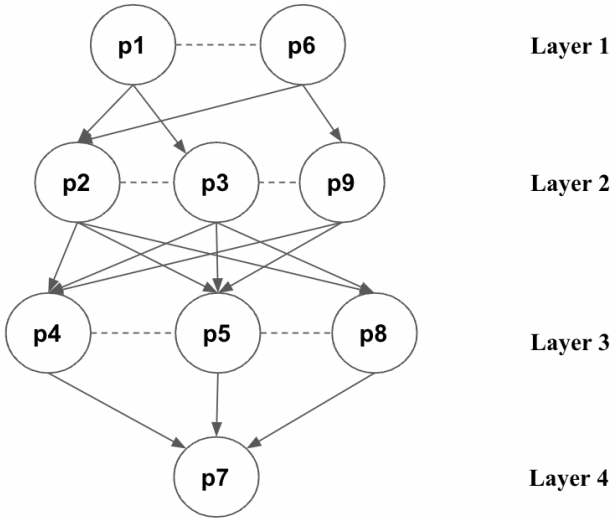


Figure 4.4: Skyline Layers of the Dataset Shown in Figure 4.2b: Node u points to node v if u dominates v.

Example 10. *In Figure 4.2a, suppose we are interested in 3-Cores. It is clear that p_6 cannot be a part of any such subgraph due to insufficient neighbours and we can safely ignore it. Moreover, when accessing node p_3, we use its following nodes in the skyline layers shown in Figure 4.4 ($\{p_9, p_4, p_5, p_8, p_7\}$) to generate groups and it is can be seen that p_3 does not have enough neighbors ($\{p_4, p_5\}$) in this set. Thus, we can safely skip processing it.*

When accessing node u, not all the following nodes are qualified to be a part of a k-Core. Based on this fact, [Zha+19] computes one maximal Clique, containing u, at each step to optimise the process of searching for skyline k-Cliques. The reason is that any group of nodes chosen from this Clique will form a Clique as well. Any node that is not included in this maximal Clique should be investigated separately to see if it can form a clique with u and other available

nodes. This method cannot be applied when listing k-Cores instead of k-Cliques. The reason is that the induced subgraph on a set of nodes chosen from a k-Core graph, will not form a k-Core necessarily.

In the case of searching for k-Cores, one can compute the maximum k-Core on u and its following nodes and use that to generate k-Cores within a smaller search space. This will cover Pruning Rules 1 and 2 as well. In this study, we store a maximum k-Core subgraph, denoted by K. After processing node $u \in G$, u is removed from K and K is updated as a k-Core. This can be done using peeling algorithm (based on [BZ03][MB83]). Note that when accessing node u, we no longer need to process u if it is already removed from K leading to the following pruning rule.

Pruning Rule 3. When accessing a node, we can skip processing it if is not included in the maximum k-Core graph.

We further reduce the search space having the fact that skyline groups should be connected k-Cores. When accessing node u, we extract the connected component of K which includes u, denoted by $\tau(u)$, and use that to generate skyline groups. The reason is that for any node v that is not included in $\tau(u)$ there is no path connecting v to u. As a result, u and v cannot be in the same skyline group as we are looking for groups in which there is a path between any two nodes.

Optimisation 2. When accessing node u, use $\tau(u)$ to generate skyline groups as $\nexists v \in G$, $v \notin \tau(u)$ where $v \in C_{k,g}(u)$.

If $|\tau(u)|$ is less than g for any node $u \in G$, we can skip processing node u.

Pruning Rule 4. If $|\tau(u)| < g$ we can safely skip generating $C_{k,g}(u)$ as this will be an empty set.

Moreover, if $|\tau(u)|$ equals g, there is only one k-Core that can be generated at this step.

Optimisation 3. When accessing node u, if $|\tau(u)| == g$, $C_{k,g}(u)$ is equal to $\{\tau(u)\}$. Safely skip the rest of the group generation computations at this step.

Example 11. *In Figure 4.2a, suppose we are interested in 2-Cores. When accessing node p_6, the connected component including p_6 contains $\{p_6, p_7, p_8, p_9\}$. It is clear that the only 2-Core subgraph including p_6 is $\{p_6, p_7, p_8, p_9\}$.*

4.2.4 Early Termination

The maximum k-Core subgraph, K, includes all remaining nodes that should be used to generate skyline groups. We can safely terminate the algorithm whenever there are not enough nodes left in K.

Early Termination 1. Let K be the maximum k-Core on the remaining nodes. If $|K| < g$, there are not enough nodes left to generate groups of size g and we can safely terminate the algorithm.

Example 12. *In Figure 4.2a, suppose we are interested in 2-Cores of size 4. We process node p_1 and remove it from K. Then we access node p_6 and with removing that, nodes p_7, p_8 and p_9 should also be removed as they no longer meet the degree constraints. Following the skyline layers (Figure 4.4), next node to process is p_2. When this node is processed, it is removed from K. After this step, there are only 3 nodes left ($\{p_3, p_4, p_5\}$) and no groups of size 4 can be generated anymore, so we safely terminate the algorithm.*

Moreover if at some step, $|K|$ equals g, there is only one possible group of size g that can be generated. Note that there will not be enough nodes to generate a group of size g in the next steps. This results in the following pruning rule.

Optimisation 4. When accessing node u, if $|K| == g$, skip the rest of the group generation computations.

[Zha+19] used an early termination condition based on skyline layers which is utilised in this study as well. The set of skyline k-Cores, Sky, is generated progressively following the access order described earlier. Following the approach of [Zha+19], for each $S \in Sky$, we store the MAX representative point, S^+ (Definition 6). Moreover, when building the skyline layers, for each layer L, a MIN representative point of all its objects is stored, denoted by L^- (Definition 6). We can safely terminate the algorithm if exists a S^+ such that S^+ dominates L^-.

Early Termination 2. Let Sky be the set of current skyline groups. When accessing a new layer, L, following the access order, if $\exists\, S \in Sky$ such that $S^+ \prec L^-$, we can safely terminate the algorithm.

4.2.5 Listing K-Cores

Overview

Our k-Core listing algorithm is inspired by [Con+18], where a new method is introduced to list n-Plexes with the three following properties: 1) connected, 2)

with a diameter equal to or less than 2, 3) maximal, i.e. those that are not a subset of any n-Plex satisfying the first two properties. [Con+18] defines this problem as follows: Identify all maximal connected n-Plexes P' that contain a given connected n-Plex P and exclude a given set of nodes $excl$. The problem is approached recursively as follows: for a vertex v that is not in P or $excl$ and such that $P \cup v$ forms a connected n-Plex, the problem is divided into two recursive calls with updated tailset and excluding set. One call finds all solutions that include v by adding v to P, while the other call finds all solutions that exclude v by adding it to $excl$. If the tailset, $cand$, becomes empty, then P can only be considered a maximal connected n-Plex if the set of excluded nodes, $excl$, is also empty, because any node in $excl$ can be used to further expand P. Note that if a node does not satisfy the connectivity requirement, it is added to $excl$, however, it may later be qualified to be included in the tailset. This is because by enlarging an n-Plex, the connectivity constraint may be met and the previously excluded node may become a valid candidate for inclusion in the solution.

Algorithm 3 ListNPlex [Con+18]

Input: graph G, integer n
Output: all diameter-2 n-Plexes of G

for all $u \in G$ **do**
    enum$(G, \{u\}, G \backslash \{u\}, \emptyset)$

function ENUM$(G, P, cand, excl)$
    if $cand \cup excl = \emptyset$ **then** output P and **return**
    for all $c \in cand$ **do**
        $cand'$, $excl' \leftarrow$ update$(G, P \cup \{c\}, excl)$
        enum$(G, P \cup \{c\}, cand', excl')$
        $cand' \leftarrow cand' \backslash \{c\}$
        $excl' \leftarrow excl' \cup \{c\}$

function UPDATE$(G, P, excl)$
    $cand' \leftarrow \{u \in G \backslash (P \cup excl) \mid P \cup \{u\}$ is connected n-Plex $\}$
    $excl' \leftarrow \{u \in excl \mid P \cup \{u\}$ is connected n-Plex $\}$

Given an arbitrary ordering of nodes, [Con+18] breaks down the original problem by listing all n-Plexes that have u as their smallest node for each node u to avoid generating duplicate groups. An overview of this method is presented in Algorithm 3.

In this study, we are interested in k-Cores with no self loops and a size no

greater than $2k+1$. Based on property 2 and Theorem 1, we know that such k-Core is connected and has a diameter less than or equal to 2. Moreover, one can find equivalency between k-Core and n-Plex (property 3) such that a k-Core of size g is a (g-k-1)-Plex.

Without considering skyline properties and requirements, we can use ideas from [Con+18] to design an algorithm to list such k-Cores. However, we need to account for the following differences between computing a maximal n-Plex and a k-Core of a size no greater than $2k+1$: 1) a maximal n-Plex has no size limit and 2) such k-Cores are always connected while this is not necessarily true for a maximal n-Plex. To address these incompatibilities, we prune a branch as soon as its corresponding group includes the desired number of nodes. Also, since we know that the k-Cores we are interested in are always connected, we do not need to check connectivity requirements and keep track of an excluding set as any node that is removed from the tailset will no longer be added to it. Moreover, as we are in search of skyline k-Cores, we use the access order described in Section 4.2.1 to reduce the overhead of dominance tests.

As mentioned in algorithm 1, we list all k-Cores of size g including the accessed node, u, in each iteration. We start with group $\{u\}$ and initialise a tailset. We iterate over all elements of the tailset, add one at a time to $\{u\}$, create a tailset for each new set by removing unqualified nodes, and make recursive calls on the new sets.

Detailed Description

Based on the Optimisation 2, we use $\tau(u)$ to generate a list of k-Cores when accessing node u. We start with group $\{u\}$ and $\tau(u)\backslash\{u\}$ as a tailset. We expand the group by adding each and every node of the tailset one at a time and updating the tailset by removing nodes that are no longer qualified. We continue adding to a group until it includes g nodes and then add it to the set of skyline candidates.

The method is presented in Algorithm 4. Note that on line 2, the tailset is limited to nodes in $u.edges$ as in the case of $g = k-1$ we would be looking for k-Cliques where all nodes are directly connected.

Lines 6 - 19 show the flow for each recursive call. When adding node v to an incomplete group, if v misses more than $g-k-1$ connections within the group, we can safely terminate that branch as the incomplete group will not grow into a k-Core (line 10). Tailset for each incomplete group is updated in two phases. Suppose we expand an incomplete group by adding node v to it. In the first phase, we remove nodes with IDs lower than $v.ID$ from the tailset (line 8). This is done to avoid generating duplicate groups. In the second phase, we use k-Core properties to shorten the tailset. All nodes in a skyline k-Core group of size g,

Algorithm 4 ListKCores

Input: set of nodes N sorted based on access order, size k, size g
Output: set of k-Cores C

1: $T \leftarrow N[1:]$
2: **if** $g - k = 1$ **then** $T \leftarrow T \cap N[0].edges$
3: $C \leftarrow \{\}$
4: UpdateKCores($\{N[0]\}, T, k, g, C$)
5: **return** C

6: **function** UPDATEKCORES(M, T, k, g, C)
7: **while** T is not empty **do**
8: $u \leftarrow T.popfront()$
9: $neighbours \leftarrow E_u \cap M$
10: **if** $M.size - neighbours.size \leq g - 1 - k$ **then**
11: **if** $M.size = g - 1$ **then** add $M \cup u$ to C and **continue**
12: $M' \leftarrow M$, $T' \leftarrow T$
13: **for all** $v \in M' \backslash neighbours$ **do**
14: $v.missed$ += 1
15: **if** $v.missed = g - 1 - k$ **then**
16: $T' \leftarrow T' \cap E_v$
17: **if** $M'.size + T'.size + 1 < g$ **then continue**
18: $u.missed \leftarrow M'.size - neighbours.size$
19: UpdateKCores($M' \cup u, T', k, g, C$)

denoted by S, have at least k neighbors within S, i.e., they can miss a maximum of $g - k - 1$ connections with other members of S assuming no self loops are allowed. We keep track of the number of missed connections of each node in an incomplete group and when this number reaches $g - k - 1$ for any node w, we update the tailset as its intersection with neighbours of w to make sure w misses no more connections. Using this approach we shorten the tailset effectively (lines 11 to 16). Moreover, we prune a branch if its tailset no longer has enough nodes to be expanded to a group of size g (line 17).

Example 13. *Suppose we are interested in 2-Cores of size 4. Figure 4.5 shows how groups are generated when the first node in Figure 4.4, p_1, is accessed. The connected component including p_1, $\{p_1, p_2, p_3, p_4, p_5\}$, is used to list k-Cores. The red branches are pruned as there are not enough nodes left in the tailset to generate groups of size 4. When adding a new node to a group, nodes with access order lower than the new node are removed from the next level tailset. Note that when adding p_2 to $\{p1\}$, p_4 is removed from the*

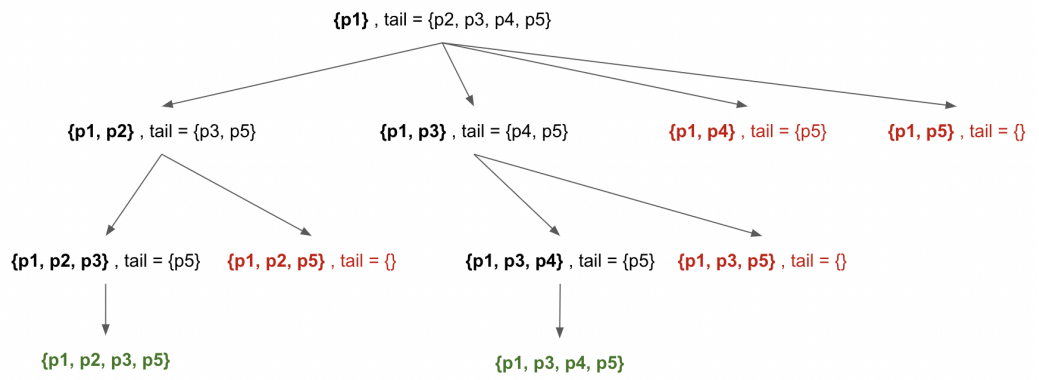


Figure 4.5: Listing k-Cores

tailset. The reason is that each node can miss at most 1 connection within the group. p_1 and p_2 are not connected and therefore any new node added to the group must be connected to both of them.

4.2.6 Generating Skyline k-Cores

Overview

In this section, we introduce the sequential and parallel versions of our progressive algorithm, **SKCore**, that produces all skyline k-Cores of a given vertex-labelled graph. Broadly speaking, we first compute the maximum k-Core graph of the input graph. Then we iterate through nodes following the skyline layers order, generate k-Cores including the accessed node taking advantage of optimisations and pruning rules described in Section 4.2.3, compare them to earlier-generated peers and known skylines (using Algorithm 5), and then add them to the list of skyline groups if not dominated. At the end of each iteration the maximum k-Core is updated with respect to the remaining nodes to narrow down the search space.

Algorithm 5 GroupDominance

Input: groups G, G'
Output: true if $G' \prec G$, false otherwise

1: **for all** $u \in G \backslash G'$ **do**
2: **if** $\nexists v \in G' \backslash G$ such that $v \prec u$ **then return** false
3: **return** true

Detailed Description

The proposed approach performed serially is shown in Algorithm 6. On line 4, the Early Termination 2 is checked. On lines 6 and 8, Pruning Rule 4 and Early Termination 1 are used to terminate early if possible. On line 9, it is checked if the accessed node is qualified to be processed (Pruning Rule 3). If so, then $\tau(u)$ is computed to extract the connected component of K including the accessed node using a DFS method. On line 11, Pruning Rule 3 is checked. On lines 13-17, possible k-Cores are listed using Algorithm 4 and compared to known skylines and earlier-generated peers using Algorithm 5 to identify if they are skyline groups. Finally, the maximum k-Core graph, K, is updated on line 18 at the end of each iteration.

Algorithm 6 Sequential-SK-Core (SKCore)

Input: skyline layers of graph G, size k, size g
Output: set of skyline k-Cores Sky

```
1: Sky ← {}
2: K ← MaxKCore(G)
3: for all layer L do
4:     if ∃ S ∈ Sky such that S+ ≺ L− then return Sky
5:     for all node u ∈ L do
6:         if |K| = g AND ∄S ∈ Sky such that S ≺ K then
7:             add K to Sky
8:         if |K| ≤ g then return Sky
9:         if u ∈ K then
10:            τ(u) ← ConnectedComponent(K, u)
11:            if |τ(u)| = g AND ∄S ∈ Sky such that S ≺ τ(u) then
12:                add τ(u) to Sky
13:            else if τ(u) > g then
14:                kcores ← ListKCores(τ(u), k, g)
15:                for all C ∈ kcores do
16:                    if ∄ S ∈ Sky such that S ≺ C then add C to Sky
17:                    Remove C from kcores
18:            remove u from and update K
19: return Sky
```

Example 14. *Suppose we are interested in 2-Cores of size 4. The maximum k-Core, K, contains all nodes at this step. Following the access order, first p_1 is accessed and as described in Example 13 (Figure 4.5), two groups are generated, $\{p_1, p_2, p_3, p_5\}$ and $\{p_1, p_3, p_4, p_5\}$. $\{p_1, p_3, p_4, p_5\}$ is removed from*

the list as it is dominated by $\{p_1, p_2, p_3, p_5\}$. $\{p_1, p_2, p_3, p_5\}$ *is added to the list of skyline groups and* p_1 *is removed from* K. *Next,* p_6 *is accessed in the connected compoent* $\{p_6, p_7, p_8, p_9\}$. *Based on optimisaton 3,* $\{p_6, p_7, p_8, p_9\}$ *is the only group generated at this step. It is compared to the known skyline groups* ($\{p_1, p_2, p_3, p_5\}$) *and then added to this list as it is not dominated.* p_6 *is removed as well as* p_7, p_8, *and* p_9 *as they no longer meet the degree requirements. At this step,* K *includes only 4 nodes, so there is only one possible group of size 4* ($\{p_2, p_3, p_4, p_5\}$) *based on Optimisation 4. By comparing this to known skylines, it is identified as dominated and removed from the list of candidates. By removing* p_2, *there are no longer enough nodes left in* K *and we safely terminate the algorithm (Early Termination 1).*

The parallelised version, **PKCore**, is designed inspired by [Par+09][Che+15] correctness of which follows from [Che+15] and Theorem 3. In this version (Algorithm 7), instead of processing nodes one at a time, they are processed in batches of size t where t equals the number of available threads. Note that as in [Che+15], each batch is processed only after the generation of skyline groups from all preceding batches is completed and the skyline k-cores are globally known. In each iteration, the next t nodes in K (following the access order described in Section 4.2.1) are assigned to t threads (line 8). A single thread checks the Early Termination 2 for any new layer accessed and set the termination flag to true to skip unnecessary computations in the next round (lines 9-14). Then each thread processes its assigned node, generates k-Cores, compares them to known skylines and earlier-generated peer k-Cores (other k-Cores created by the same thread) taking advantage of output order (Property 3), and removes dominated ones (lines 16-24). Threads are synchronised at this step (line 25) to share their local results and each of them compares its k-Cores to those of threads which are assigned nodes with lower access order (Property 3)(lines 26-27). These are the last comparisons required for identifying potential skyline groups. Next, threads are synchronised once more after all comparisons are performed (line 28) and non-dominated k-Cores are added to the list of skyline groups (line 31). Afterwards, a single thread is responsible for removing all processed nodes from K and updating it as a k-Core (line 32).

Algorithm 7 Parallel-SK-Core (PKCore)

Input: skyline layers of graph G, size k, size g, size t
Output: set of skyline k-Cores Sky

1: $Sky \leftarrow \{\}$
2: $K \leftarrow MaxKCore(G)$
3: $terminate \leftarrow false$
4: $kcores_i \leftarrow \{\}, \forall i \in [1, t]$
5: **while** !$terminate$ **do**
6: **if** $|K| = g$ AND $\nexists S \in Sky$ such that $S \prec K$ **then** add K to Sky
7: **if** $|K| \leq g$ **then return** Sky
8: assign nodes with lowest IDs in K, $u_1, u_2, ..., u_t$, to threads
9: $layer_{max} \leftarrow layerID_{max} + 1$
10: **for all** new layer L accessed **do**
11: **if** $\exists\ S \in Sky$ such that $S^+ \prec L^-$ **then**
12: $layer_{max} \leftarrow L.ID$
13: $terminate \leftarrow true$
14: **break**
15: // beginning of parallel region, $i \in [1, t]$
16: **if** $u_i.layerID < layer_{max}$ **then**
17: $\tau(u_i) \leftarrow ConnectedComponent(K, u_i)$
18: **if** $|\tau(u_i)| = g$ AND $\nexists S \in Sky$ such that $S \prec \tau(u_i)$ **then**
19: add $\tau(u_i)$ to $kcores_i$
20: **else if** $\tau(u_i) > g$ **then**
21: $kcores_i \leftarrow ListKCores(\tau(u_i), k, g)$
22: **for all** $C \in kcores_i$ **do**
23: **if** $\exists\, S \in Sky \cup kcores_i$ such that $S \prec C$ **then**
24: remove C from $kcores_i$
25: // synchronisation step
26: **for all** $C \in kcores_i$ **do**
27: **if** $\exists\, C' \in kcores_j$, $j < i$ such that $C' \prec C$ **then** flag C for removal
28: // synchronisation step
29: remove any $C \in kcores_i$ flagged for removal
30: // end of parallel region
31: add all $C \in kcores_i$ to Sky and clear $kcores_i$, $i \in [1, t]$
32: remove $u_1, u_2, ..., u_t$ from and update K
33: **return** Sky

Chapter 5

Empirical Validation

We perform comprehensive experiments to assess our proposed approach. We tailor the algorithm introduced in [Zha+19] to be compatible with the group dominance definition of [Akb22] and use that as our baseline to compare against for the special case of k-Cores where the size of the subgraph is equal to $k + 1$. The performance of our algorithm is evaluated based on various input parameters. These include the size of the input graph and skyline groups, the required connectivity level, the relationship between connectivity level and group size, and the concurrency level applied. Additionally, we evaluate how the number of dimensions and the distribution type of node labels affect our solution. We offer thorough explanations and analyses of how these input parameters impact our proposed approach to better understand the behaviour of our algorithm and the baseline as well.

5.1 Datasets and Parameters

We use four real datasets available on the SNAP website, consistent with datasets used in [Zha+19]. Note that for the purpose of this study, all directed edges are replaced with undirected ones and all duplicate edges produced as a result of this are removed. Table 5.1 represents statistics of the datasets. The last column shows the average percentage of nodes a certain node has connections to within the graph ($\delta_{avg}/n * 100$ where n denotes the number of nodes of the graph). This can be used as an indicator of graph density.

We perform our experiments on the previously mentioned datasets over a number of parameters including skyline size g, minimum degree requirements (k in k-Core), vertex label dimension d, and label type which is explained below. Moreover, we compare sequential and parallel versions of SK-Core (Algorithms 6 and 7) and measure the effectiveness of using different numbers of threads (t).

Dataset	Vertices	Edges	δ_{avg}	δ_{max}	Graph Density
Eucore (Eu)	1,005	16,064	32	345	3.1809
WikiVote (Wi)	8,298	100,762	24	1,065	0.2927
Enron (En)	36,692	183,831	10	1,383	0.0273
YouTube (Yo)	1,157,828	2,987,624	5	28,754	0.0004

Table 5.1: Statistics of Datasets

The original datasets do not contain multi-valued vertex labels. We generated d-dimensional labels of three types of distributions following the techniques in [BKS01] as follows:

- Independent (indep): values of the attributes are produced independently from one another utilising a uniform distribution.
- Correlated (corr): values of the attributes are produced such that a node with a good value in one dimension have good values in other dimensions.
- Anti-correlated (anti-corr): values of the attributes are produced such that a node with good value in one dimension have bad values in one or all of the other dimensions.

5.2 Experiment Setup

We apply our proposed solution to a number of scenarios including those used in [Zha+19]. We assess the performance dependency of our algorithm with respect to the graph sparsity and the number of nodes in the graph (dataset), connectivity level requirements (k), the degree of freedom within a group ($g-k$), the positive/negative correlation between attribute values of nodes (attribute distribution), the dimensionality of node attributes (d), running sequentially (seq) or parallel (par), and the concurrency level applied (t).

Table 5.2 shows the details of all experiment setups. Sections 5.4.1 and 5.4.2 follow experiment configurations of [Zha+19] while Sections 5.4.3 and 5.4.4 evaluate aspects that are unique to our work. Experiments are conducted on a machine with *2 x Intel Platinum 8160F Skylake @ 2.1Ghz* and 128GB RAM and all algorithms are implemented in C++ and OpenMP. Note that if the algorithm is out of memory or it cannot finish in 12 hours, the processing time is reported as INF.

Section	Dataset	k	$g-k$	Label Type	d	seq/par(t)
5.4.1	Wi,En	4	1	indep	2-5	seq,par(48)
5.4.1	Eu	2,4,6,8,11	1	indep,corr,anti-corr	2	seq,par(48)
5.4.2	Eu,Wi,En,Yo	2,4,6,8,11	1	indep	2	seq,par(48)
5.4.3	Eu	2,3	1-3	indep	2	seq,par(48)
5.4.4	Eu	3	3	indep	2	par(1-48)
5.4.4	Yo	4	1	indep	2	par(1-48)

Table 5.2: Experiment Setups

Algorithm 8 Baseline (our adaptation for Definition 9 of [Zha+19])

Input: skyline layers of graph G, size k
Output: set of skyline k-Cliques

$Sky \leftarrow \{\}$
for all layer L **do**
  if $\exists\ S \in Sky$ such that $S^+ \prec L^-$ **then return** Sky
  for all node $u \in L$ **do**
    if $|\Delta(u)| \geq k-1$ **then**
      $M \leftarrow$ induced subgraph on $\Delta(u)$
      $kcliques \leftarrow Cliques(M, k-1, \{u\})$
      for all $C \in kcliques$ **do**
        for all $C' \in Sky \cup kcliques, C' \neq C$ **do**
          if $CheckDom(C', C) = true$ **then**
            $kcliques \leftarrow kcliques \backslash C$
            break
      $Sky \leftarrow Sky \cup kcliques$
return Sky

function CheckDom(C', C)
  // returns true if C' dominates C, false otherwise
  if $C'^+ \prec C^-$ **then return** true
  for all $u \in C \backslash C'$ **do**
    if $\nexists v \in C' \backslash C$ such that $v \prec u$ **then return** false
  return true

5.3 Baseline

The algorithm introduced in [Zha+19] is used to discover skyline k-Cliques of a given dataset under permutation group dominance (Definition 8). We replace

this definition with the generalised group dominance (Definition 9) and modify the algorithm accordingly so that it can be used as our baseline for performance studies. Note that this method is developed to discover skyline k-Cliques only (where $g-k=1$) and it cannot be used to search for k-Cores (where $g-k \geq 1$).

Algorithm 8 presents the baseline. Note that $\Delta(u)$ denotes the neighbours of node u following it in the access order. We iterate through nodes following the access order and check Early Termination 2 when accessing a new layer (line 3). Pruning Rule 2 checks if the accessed node can form a k-Clique (line 5). For each accessed node, the subgraph induced on its neighbors is computed and used to generate k-Cliques (line 6). Method *Cliques* on line 7 retrieves all cliques of a certain size from a given graph (using the technique introduced in [DBS18]). All listed k-Cliques are compared to known skylines and peers to identify if they are skyline groups (line 8 - 12). CheckDom method (lines 15 - 20) is the implementation of group dominance check in [Zha+19] excluding pruning rules that are not compatible to generalised group dominance (Definition 8).

Many of the optimisations in the original algorithm of [Zha+19] aim to reduce the cost of dominance checking under the permutation-based model, but the generalised model is much less expensive. Especially in large and sparse graphs, performing dominance tests is not the bottleneck in this problem. Table 5.3 reports the percentage of time taken by three main tasks using the Enron dataset with $k = \{4, 6, 8, 11\}$. These include *listing* cliques, performing dominance *tests* and *reducing* search space (by computing induced subgraphs and connected component in Algorithms 8 and 6, respectively).

	Baseline			SKCore		
k	listing	testing	reducing	listing	testing	reducing
4	9.8%	8.7%	63.4%	21.5%	4.1%	68.6%
6	28.9%	17.8%	40.8%	69.2%	7.6%	19.0%
8	42.0%	16.7%	29.0%	85.7%	5.4%	6.5%
11	61.7%	4.6%	25.8%	94.9%	1.4%	3.2%

Table 5.3: Breakdown of Time Spent in Major Components

Given this observation, we removed some of the optimisations from [Zha+19] under the belief that, with this dominance model, they introduce more overhead than they save. For example, we omit the maintenance and use of an R-tree for storing MAX and MIN representatives to check Early Termination 2 on line 3 and the repeated calculation and iteration of maximal cliques inside the loop of line 4. We concede that, if we are mistaken here, the baseline could be up

to 17.8% faster. On the other hand, we retain in the baseline the low-overhead cheap test on line 17, even though on the Eucore dataset, for example, at $k = 2$ and $g - k = 2$, the test fails more than 99% of the time, because it is inexpensive. Still, we do not adopt this idea for our proposed approach (Algorithm 6).

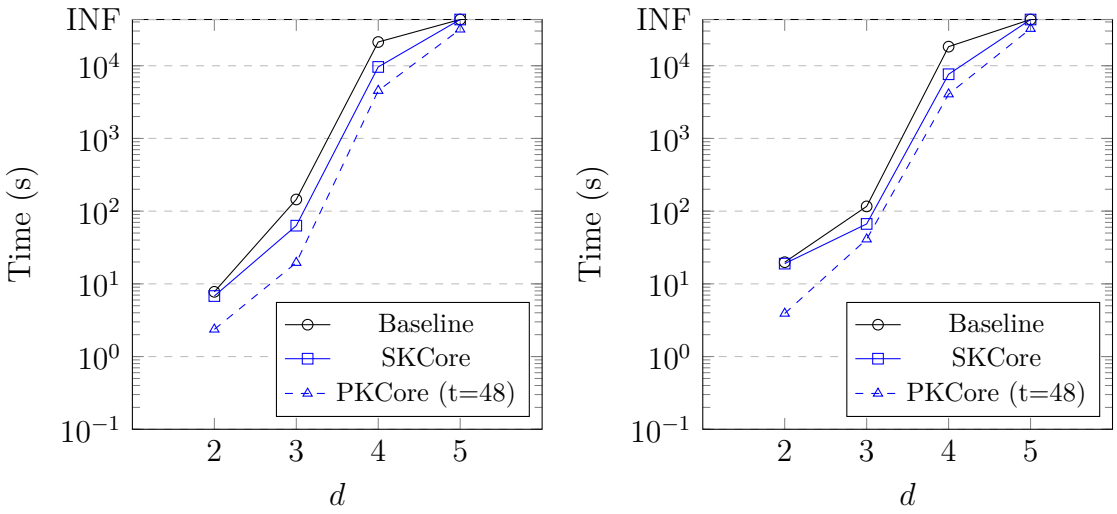
Broadly, we observe that our baseline is quite a bit faster than the algorithm in [Zha+19] as dominance tests are less expensive here.

5.4 Performance Studies

This section provides a comprehensive summary and analysis of the results of all experiment setups listed in Table 5.2. Note that SKCore and PKCore in the legend of plots denote the sequential and parallel versions of our proposed solution respectively.

5.4.1 Dimension Size and Attribute Distribution

To evaluate the effect of the number of dimensions on our proposed solution, we perform experiments on WikiVote and Enron datasets with independent attribute distribution changing d from 2 to 5. We use k and g values of 4 and 5 respectively ($g - k = 1$) and as a result, the findings could be compared to the baseline. The results are outlined in Figure 5.1.



(a) WikiVote

(b) Enron

Figure 5.1: Processing Time (Varying d)
Independent Label Distribution, $k = 4$, $g - k = 1$

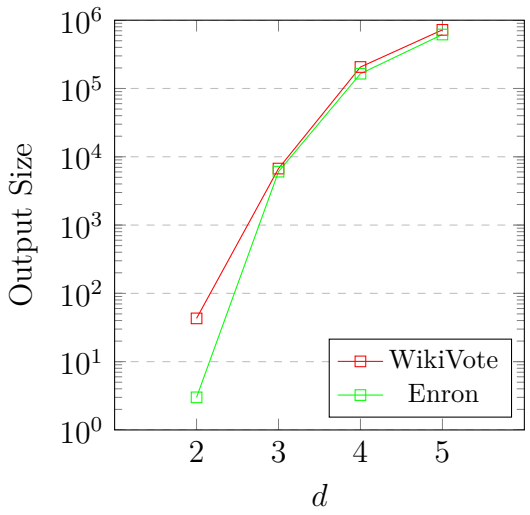


Figure 5.2: Output Size (Varying d)
Independent Label Distribution, $k = 4$, $g - k = 1$

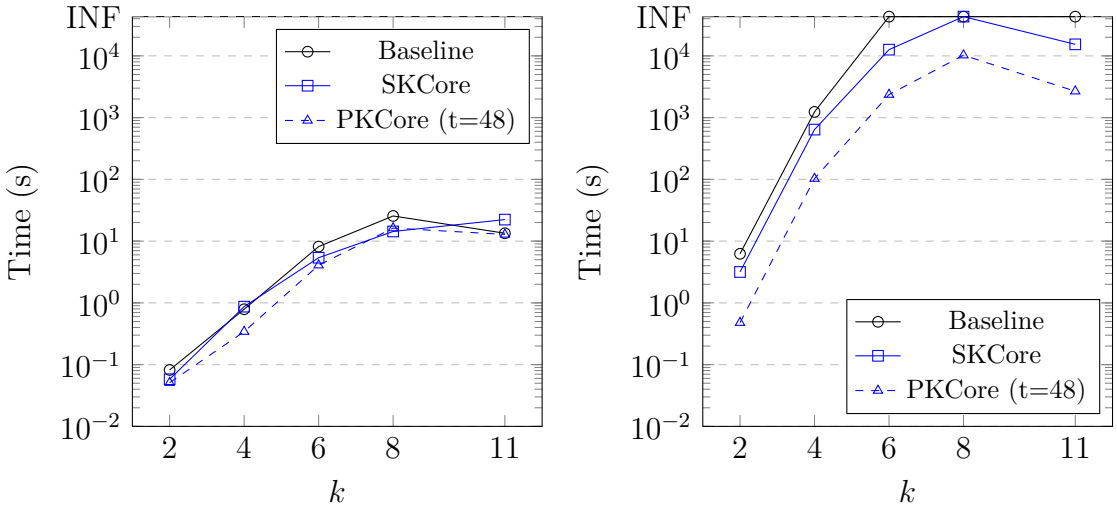
The plots demonstrate that a rise in the value of d results in an exponential increase in the required processing time. As the dimensionality increases, the number of k-Cores with skyline property grows as well. This necessitates an increased amount of dominance tests that need to be performed, since groups are less likely to be dominated. The same pattern can be seen in Figure 5.2 showing the output size of the corresponding scenarios in Figure 5.1. In other words, the exponential growth of the execution time is the result of the significant expansion of the solution space. It can be seen that the processing time of the parallel version follows the same trend with increasing number of dimensions.

Moreover, it can be seen that the increase in the execution time is sharper than that of the output size. That is because any new skyline group added to the solution space should be tested against many other groups to be identified as skyline. As a result, any increase in the number of skyline groups leads to a significant growth in the number of dominance tests required to be performed.

The baseline exhibits a similar pattern and it can be seen that our proposed solution (SKCore) runs faster than the baseline in all scenarios depicted in Figure 5.1 with up to 2.4x acceleration rate.

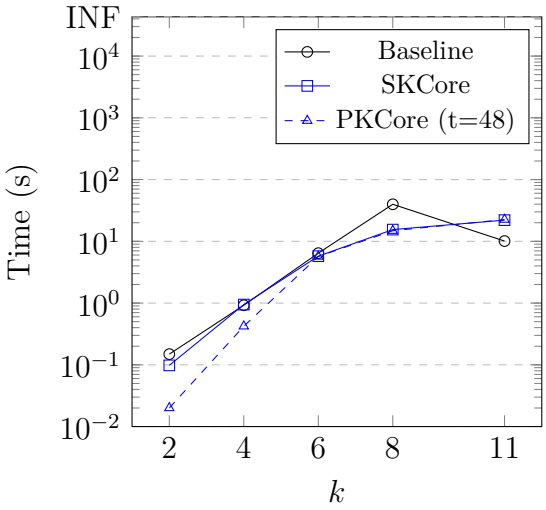
Note that for $d = 5$ using both datasets, the execution time of the baseline and sequential version of our method exceeded 12 hours and therefore they are specified as INF while the parallel version of our algorithm finished in less than 9 hours. It can be seen that at a certain value of d (4 in this case), increasing d will no longer cause an exponential increase in the output size and processing time. The reason is that, with increasing d, more dominance tests fail leading to

an increase in the number of skyline groups. At a certain point, the number of failed dominance tests reaches a level that increasing the dimension size causes fewer new tests to fail and the output size grows less sharply.



(a) Correlated

(b) Anti-Correlated



(c) Independent

Figure 5.3: Processing Time (Varying Label Distribution Type)
Eucore dataset, $d = 2$, $g - k = 1$

We also assess the influence of using different attribute distribution types and provide a visual representation of the results of the corresponding experiment setup in Figure 5.3. For this experiment, we used the Eucore dataset with 2-

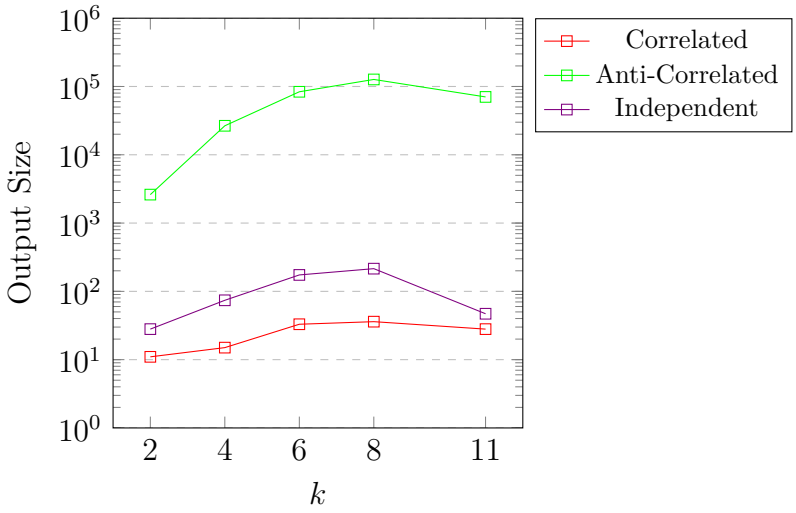


Figure 5.4: Output Size (Varying Label Distribution Type)
Eucore dataset, $d = 2$, $g - k = 1$

dimensional labels as in [Zha+19].

In general, the execution time for nodes with correlated attribute values is much less than that of independent and anti-correlated ones. This happens as a result of a significant decrease in the output size (Figure 5.4). Intuitively, with correlated and anti-correlated attributes, the number of skyline results is the smallest and largest among the three distribution types respectively. The reason is that having correlated attribute values leads to strong points that are more likely to dominate other points and fewer skyline groups are generated. This causes a decrease in the number of dominance tests required to identify skyline groups. The opposite happens when having anti-correlated attribute values and independent attribute values stands in between these two extremes. This trend holds true for baseline as well as sequential and parallel versions of our proposed method.

Note that the trend of the processing time with respect to k is similar for all three distribution types. Generally, as the value of k increases, the number of k-Cores also grows, causing longer execution times for listing and comparing groups. However, at some value of k, the number of possible combinations decreases, leading to a decrease in execution time. This is the reason that with increasing k the processing time also increases and then at a certain point, the trend levels off, and starts to drop. This can be seen clearly for anti-correlated distribution. In the case of correlated and independent distributions, this peak occurs at a lower value of k in baseline (8) than our proposed solution and as a result the baseline runs faster for $k = 11$ for these two distributions. This

pattern is discussed with more depth in section 5.4.2. For most of the scenarios depicted in Figure 5.3, the sequential version of our algorithm, SKCore, has faster processing time than the baseline (up to 1.98x speedup).

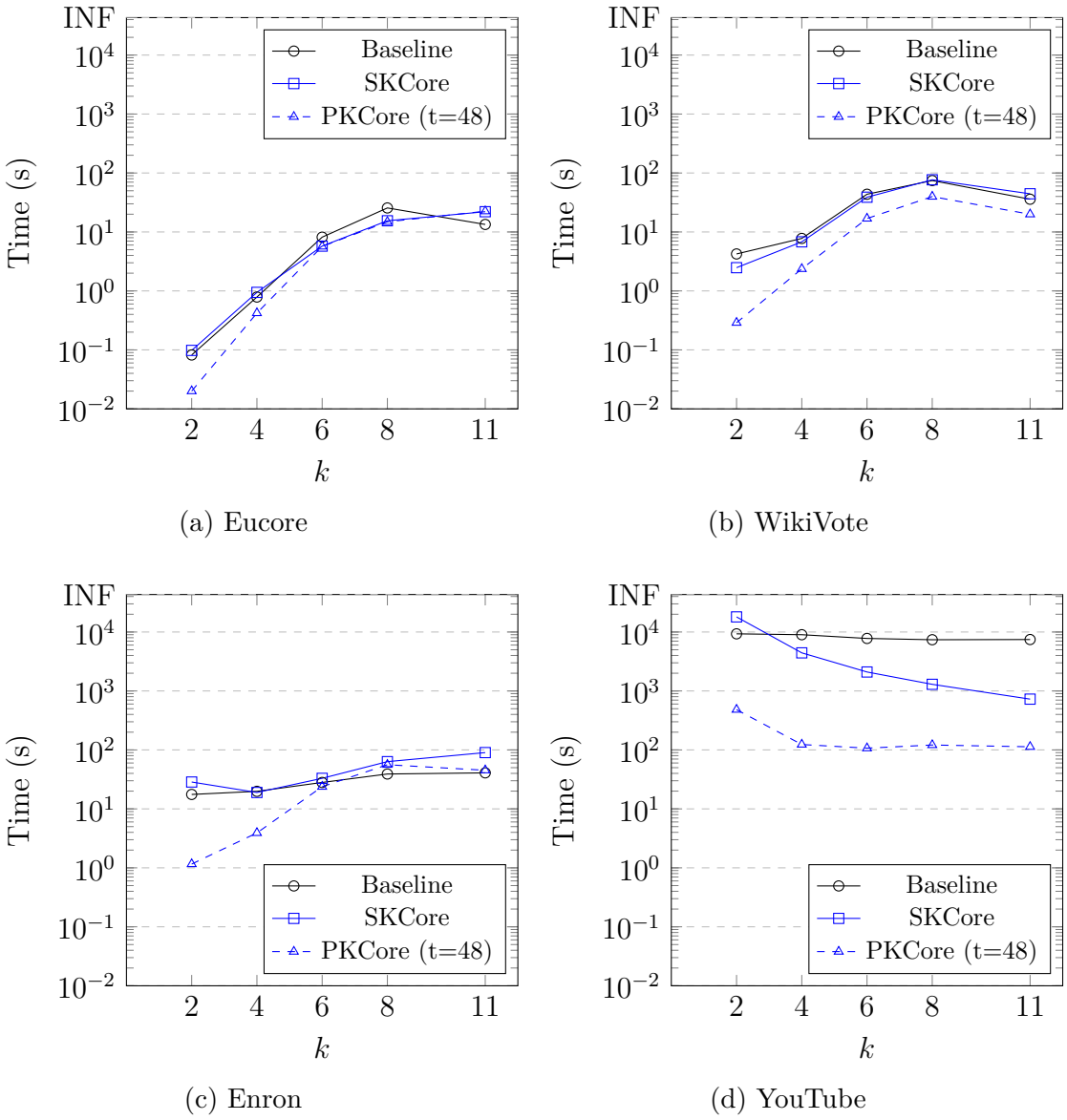
5.4.2 Connectivity Strength (k)

In this section, we analyse how the connectivity requirements influence the performance of our solution. We use all four datasets in Table 5.1 and 2-dimensional attributes and independent distribution for values with a changing k in the range of 2 to 11. We fixed $g - k$ as 1 so that the results could be compared to those of [Zha+19]. The findings of this experiment setup are reported in Figure 5.5.

Note that the higher the number of possible combinations of size g meeting degree requirements, the longer it takes to list them and compare them to one another for identifying skyline groups. Intuitively, with increasing the size of the input graph, the number of possible combinations grows. The effect of this can be seen in Figure 5.5. The execution time for the largest dataset (YouTube) is longer than that of other datasets. This is true in general, when comparing the results of any two datasets with the same value of k having Table 5.1.

Moreover, generally, for any given dataset, with increasing k, the amount of time required for execution also increases. However, at a certain point, the trend levels off and begins to decrease. This occurs because as the value of k increases, the number of potential groups also increases, which leads to longer execution times for tasks such as listing k-Cores and performing dominance tests. Nevertheless, as k continues to increase, the number of possible combinations eventually decreases, resulting in a corresponding decrease in execution time. As an example, it can be seen for the WikiVote dataset (Figure 5.5b) that as k grows from 2 to 6, the execution time of SKCore increases from 2.478s to 38.87s (slower processing time by 15.69x). The execution time rises to a high point and peaks in $k = 8$, reaching 76.69s and then starts to drop hitting 44.406s when the k grows to 11. Note that the peak is not clear for the Eucore dataset (Figure 5.5a) since it may happen for a k value equal to or greater than 11. As Figure 5.5 shows, the processing time of the baseline and PKCore follows the same rule with respect to k.

The peak of curves shown in Figure 5.5 relies on the graph sparsity. In a sparse graph, the number of possible k-Cores is lower than that of a dense graph. Moreover, in such graphs, nodes are less likely to meet degree requirements and the number of components is high. As a result, pruning rules described in Section 4.2.3 are more effective and Early Termination 1 holds earlier in comparison to a dense graph. This can be recognized by comparing each curve to the corresponding graph density indicator in Table 5.1. Generally, the peak and overall trend of the processing time of PKCore is close to its corresponding



(a) Eucore

(b) WikiVote

(c) Enron

(d) YouTube

Figure 5.5: Processing Time (Varying k)
Independent Label Distribution, $d = 2$, $g - k = 1$

SKCore version however, the peak for the baseline occurs at a lower value of k than for SKCore and PKCore when using the Eucore dataset.

In most of the scenarios depicted in Figure 5.5, the processing time of the baseline and SKCore are close and neither is consistently faster or slower than the other. However, for the YouTube dataset, SKCore has notably faster processing time than the baseline for values of k in the range of 4 to 11 and is able to

Dataset \ k	2	4	6	8	11
Eucore	4.90	2.24	1.01	1.04	0.98
WikiVote	8.57	2.87	2.30	1.93	2.22
Enron	24.63	4.85	1.39	1.13	1.99
YouTube	37.41	35.86	19.59	10.70	6.43

Table 5.4: Accelerations Rates for Varing k Using 48 Threads (Figure 5.6)

achieve up to 10.19x speedup.

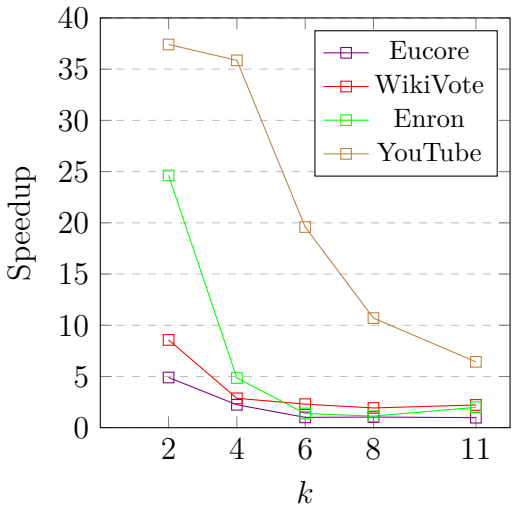


Figure 5.6: Execution Time Speedup Over SKCore (Varying k)
Independent Label Distribution, $d = 2$, $g - k = 1$, $t = 48$
Exact Coordinates Available in Table 5.4

As illustrated in Figure 5.6, the parallel version of the algorithm (using 48 threads) achieves up to 37x speedup. However, the performance is not consistent and depends on the dataset used and value of k. Note that the acceleration rates are reported over the sequential program. In the sequential version, we process nodes one by one and at each iteration we list k-Cores and compare them to identify skylines. In general, as we move forward to the end of the list of nodes, the processing time of each iteration tends to decrease as fewer nodes are available. We designed a coarse-grained parallel algorithm that each thread is responsible for one of these iterations at a given time and the processing time of threads are not necessarily the same. With increasing k, the connectivity requirement tends to be more limiting. As a result, it is more likely that nodes are removed from the maximum k-Core and the execution time of iterations decreases more quickly and the gaps become larger leading to less balanced work

distribution between threads. This is the reason that decreasing value of k results in better performance of the parallel algorithm.

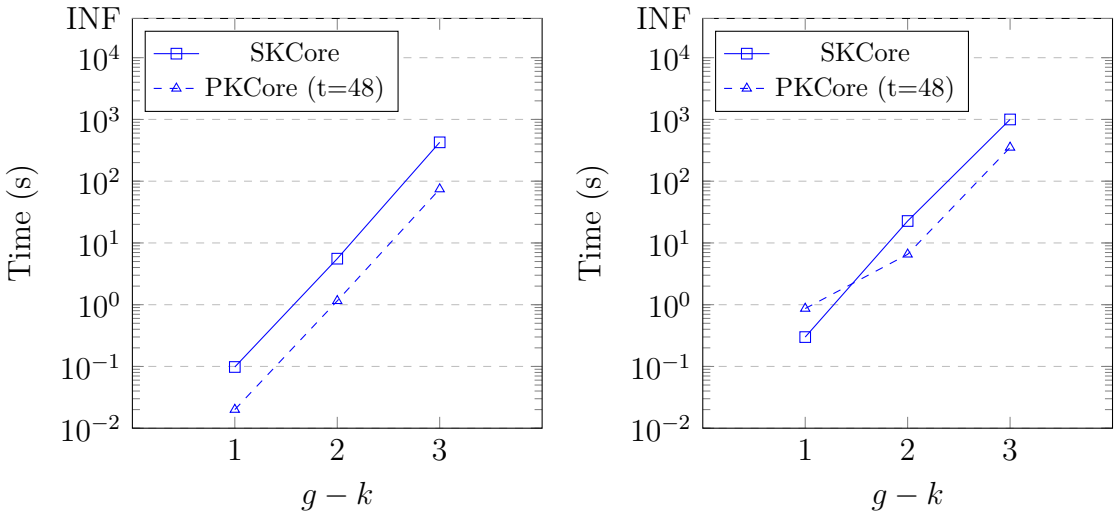
Figure 5.6 also shows that the parallel version has a better performance on the YouTube dataset. At each iteration of the algorithm, we retrieve the connected component the accessed node is included in and use that to list k-Cores. If two consecutive nodes are not in the same connected component, the processing time of these two nodes will be independent and removing the first node has no effect on the connected component including the second node. In a sparse graph, the number of components is high and it is less likely that two consecutive nodes are in the same component. As a result, two threads are more likely to use two separate sets of nodes and their workload relies on the size of the component they process. Thus, the workload of the second thread is not necessarily less than that of the first thread causing a more balanced workload in sparse graphs than in dense ones. According to Table 5.1, the YouTube dataset is a sparser graph than all other ones and as a result, the parallel version has a better performance on this dataset.

5.4.3 The K-Core Freedom ($g - k$)

When performing experiments with different g and k values, the size of the group, g, cannot be used independently to analyse the performance of the algorithm. The reason is that the relation between the group size and the connectivity requirement determines the complexity of the problem and not the group size. This is confirmed by the fact that discovering skyline k-Cliques is a more specific and less complicated problem than searching for skyline k-Cores.

In this section, we analyse the performance of our novel algorithm discovering skyline k-Cores and we discuss how changing the level of freedom within a k-Core (the maximum allowed number of missed connections) affects the execution time of our algorithm. We perform our experiments on the Eucore dataset and 2-dimensional attributes with independent distribution for the values. We use values of 2 and 3 for k and analyse the influence of changing $g - k$ from 1 to 3 on the performance of our proposed approach. Note that baseline is not included in this section as it is only able to discover skyline k-Cliques (equivalent to skyline k-Cores in which $g - k = 1$) and cannot be generalised to the search for k-Cores.

As depicted in Figure 5.7, increasing the difference between the group size and the degree requirements within the group results in exponential growth of the execution time. This is expected since with increasing $g - k$, exponentially more combinations of nodes will satisfy k-Core requirements leading to longer time needed to list k-Cores and make comparisons between candidates. The reason is that having k fixed, it is more likely for a certain node to have the minimum



(a) $k = 2$ (b) $k = 3$

Figure 5.7: Processing Time (Varying $g - k$)
Eucore Dataset, Independent Label Distribution, $d = 2$
Baseline does not support k-Core skylines where $g - k > 1$
and is not included in this experiment setup.

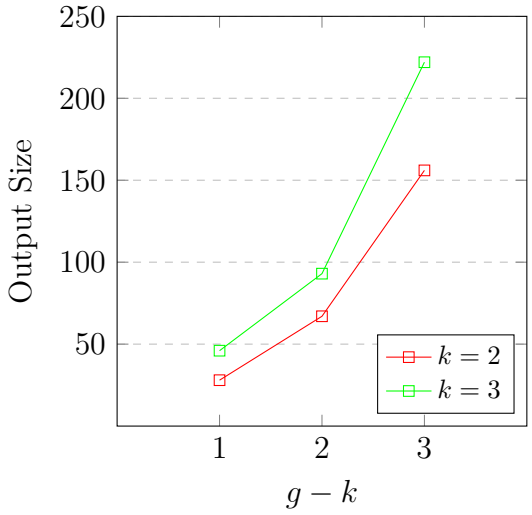


Figure 5.8: Output Size (Varying $g - k$)
Eucore Dataset, Independent Label Distribution, $d = 2$
Baseline does not support k-Core skylines where $g - k > 1$
and is not included in this experiment setup.

number of connections required to other group members within a larger group. This is confirmed by Figure 5.8 showing the output size of the corresponding scenarios in Figure 5.7 which follows the same exponential trend. Note that a linear trend in logarithmic scale (Figure 5.7) is equivalent to exponential trend in linear scale (Figure 5.8).

Figure 5.7 also shows that increasing k from 2 to 3 increases the execution time, the pattern described earlier in Section 5.4.2 in detail. This is because, in the case of $k = 3$, there are more possible combinations satisfying degree requirements and as a result listing and comparing candidates takes more time.

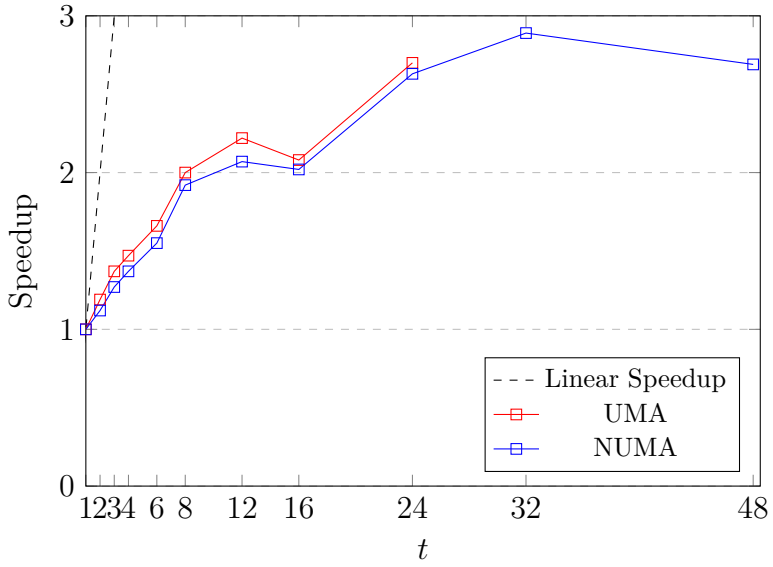
5.4.4 Parallel Scalability

In this section we analyse the parallel scalability of our algorithm. We use 2-dimensional attribute values with independent distribution and conduct the experiments in two scenarios in which our parallelised algorithm has different behaviors: a) Eucore dataset with $k = 3$, $g - k = 3$, b) YouTube dataset with $k = 4$, $g - k = 1$. Note that the baseline is not included in this experiment as it is not designed for parallel computing.

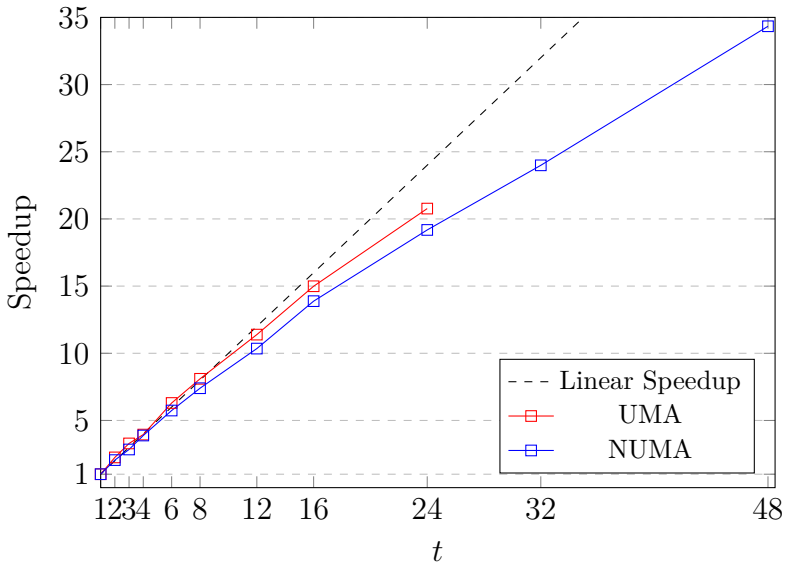
The speedup achieved by the parallel version is reported in Figure 5.9 with respect to the parallel version using a single thread. The performance of the parallelised algorithm is evaluated under both UMA (Uniform Memory Access) and NUMA (Non-Uniform Memory Access) architecture. When conducting experiments under UMA design, a single processor is used with 24 cores. In contrast, when using NUMA architecture, two processors with 24 cores each are employed, and an equal number of cores from each processor are utilised in each experiment.

Uniform Memory Access (UMA) is a type of shared memory architecture design that is utilised in parallel computing. With UMA, all processors within the system share the physical memory in a uniform manner. Therefore, the access time for a particular memory location remains constant regardless of which processor initiates the request or which particular memory location is accessed to retrieve the data. RAM and PRAM models explicitly assume UMA where all memory accesses are of unit cost. In contrast, in a NUMA architecture, a processor retrieves data from its own local memory more quickly than non-local memory, either on a different processor or shared between multiple processors. As a result, a multi-threaded algorithm is expected to run faster on a UMA architecture. Also, with a poor algorithm design, the gap is expected to grow with increasing number of threads as more communications are required between NUMA nodes.

Figure 5.9 clearly shows the effect of UMA and NUMA architecture on our parallelised algorithm. For both Enron and YouTube datasets, given a certain number of threads, the speedup of the parallel program under NUMA architecture



(a) Enron, $k = 3$, $g - k = 3$



(b) YouTube, $k = 4$, $g - k = 1$
Exact Coordinates Available in Table 5.5

Figure 5.9: Execution Time Speedup (Varying t)
Independent Label Distribution, $d = 2$
Baseline does not support parallel computing
and is not included in this experiment setup.

Design \ t	1	2	3	4	6	8	12	16	24	32	48
UMA	1	2.25	3.29	3.95	6.3	8.10	11.39	14.99	20.77	-	-
NUMA	1	2.05	2.84	3.88	5.74	7.4	10.35	13.88	19.18	23.99	34.34

Table 5.5: Accelerations Rates for the YouTube Dataset
Independent Label Distribution, $d = 2$, $k = 4$, $g - k = 1$ (Figure 5.9)

is slightly less than that of UMA design and the difference between these two remains almost constant with increasing number of threads. Moreover, it can be seen that the trend of the acceleration rate over increasing number of threads with NUMA system closely follows the corresponding pattern under UMA architecture.

Generally, the acceleration rate increases as the thread count grows from 1 to 48 as illustrated in Figure 5.9. The speedup drop at $t = 16$ in Figure 5.9a is believed to be data dependent since the same trend does not happen with the YouTube dataset in Figure 5.9b.

Figure 5.9 also shows that the parallel algorithm has a better performance on the YouTube dataset than Enron. The serial algorithm processes nodes one by one, listing k-Cores and comparing them to identify skylines at each iteration. Generally, as we progress through the list of nodes, the processing time per iteration decreases because there are fewer nodes left to be used for generating groups. We developed a coarse-grained parallel algorithm in which each thread handles one iteration at a time, but the processing time of each thread may not be the same. During each iteration of the algorithm, we obtain the connected component that contains the accessed node and use that to generate a list of k-Cores. Thus, the processing time of a node depends on the size of the connected component it belongs to and if two consecutive nodes are not part of the same connected component, the workload of the second thread is not necessarily less than that of the first thread resulting in a more balanced workload distribution between threads in sparse graphs. Table 5.1 shows that the YouTube dataset is sparser than the Enron dataset and therefore the parallelised version performs better on this dataset.

According to the findings presented in Figure 5.9b, our suggested parallel solution performs better with multiple threads, achieving super-linear speedup rates of 2.25 and 3.29 with 2 and 3 threads respectively. The rate of acceleration remains almost linear when using up to 16 threads. However, as the number of threads increases, the difference between the speedup curve and linear speedup becomes more significant.

5.4.5 Summary

In this section, we present a summary of our analysis of the experiment results. There are a number of input parameters affecting the performance of the algorithm including the size of the dataset and its sparsity, the number of dimensions of node attributes and the distribution of the attribute values, the connectivity requirements, and its difference with the group size.

Generally, having a larger dataset leads to longer processing time as more nodes should be processed. Moreover, in a sparse graph, the pruning rules and optimisations will be more effective leading to less execution time than in dense graphs. Also, the parallelised solution is more scalable in a sparse graph as a result of a more balanced workload between threads. In this case, the parallel algorithm is able to achieve super-linear speedup results with a low number of threads, 2 and 3, and almost linear acceleration rates using up to 16 threads.

A large number of dimensions needs more processing time since more skyline groups are generated necessitating more dominance tests to be performed. Using anti-correlated distribution for attribute values has the same effect.

Increasing the number of connections required (assuming the degree of freedom, $g - k$, is fixed) will cause an increase in the execution time as more combinations of the desired size satisfy the connectivity constraint. However, at a certain level of connectivity, the number of possible groups decreases and the processing time follows the same trend. Increasing the level of freedom within a group (the difference between the group size and minimum connections required for each node) also leads to slower processing time as more groups meet k-Core requirements and it takes more time to list and compare candidates.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this study, we introduced a novel algorithm for discovering skyline k-Cores under the generalised group-wise dominance relationship. The proposed approach is a progressive algorithm with the ability of early termination that takes advantage of a number of pruning rules and optimisation techniques to reduce the search space effectively and speed up the execution time. We also presented a parallelised version of the proposed solution to accelerate the processing time of skyline communities discovery.

We performed comprehensive experiments and analysed the behaviour of our algorithm with respect to the number of attribute dimensions, graph sparsity, graph size, and more. In general, increasing the number of nodes in the dataset or the number of attribute dimensions causes an increase in the processing time since more nodes should be processed and more dominance tests need to be performed. We also observe that sparse graphs benefit from effective pruning rules and optimisations, resulting in less execution time. Our algorithm is able to achieve an acceleration rate as high as 10x over the baseline on a large sparse graph.

We also evaluated the parallel scalability of our method which is able to achieve super-linear and linear acceleration rates depending specifically on the dataset sparsity and thread count. Our proposed solution attains up to 34x speedup over the single-threaded algorithm on a realistic sparse graph when utilising 48 cores.

6.2 Future work

This work can be expanded in a number of directions. First and foremost, more pruning rules and optimisation strategies could be presented based on properties of skyline groups, community structures, or combination of both. This can speedup listing and comparing skyline communities further. Moreover, we imposed an upper bound on the size of communities based on k such that all produced subgraphs are connected. This limitation can be lifted and our proposed solution can be modified accordingly to discover skyline communities of any size. Additionally, our proposed solution can be used as an inspiration for discovering skyline communities other than k-Cliques and k-Cores.

Bibliography

[Akb22] Akber, M. A. Efficient Skyline Community Discovery in Large Networks. Master's Thesis. University of Victoria, 2022. eprint: http://hdl.handle.net/1828/14157.

[BCA15] Bøgh, K. S., Chester, S., AND Assent, I. Work-Efficient Parallel Skyline Computation for the GPU. *Proc. VLDB Endow.* 8, 9 (May 2015), 962–973. ISSN: 2150-8097. DOI: 10.14778/2777598.2777605. URL: https://doi.org/10.14778/2777598.2777605.

[BKS01] Borzsony, S., Kossmann, D., AND Stocker, K. The Skyline operator. In: *Proceedings 17th International Conference on Data Engineering*. 2001, 421–430. DOI: 10.1109/ICDE.2001.914855.

[BZ03] Batagelj, V., AND Zaveršnik, M. An O(m) Algorithm for Cores Decomposition of Networks. *CoRR* cs.DS/0310049 (Oct. 2003).

[Cha+00] Chang, Y.-C., ET AL. The Onion Technique: Indexing for Linear Optimization Queries. *SIGMOD Rec.* 29, 2 (May 2000), 391–402. ISSN: 0163-5808. DOI: 10.1145/335191.335433. URL: https://doi.org/10.1145/335191.335433.

[Che+15] Chester, S., ET AL. Scalable parallelization of skyline computation for multi-core processors. In: *2015 IEEE 31st International Conference on Data Engineering*. 2015, 1083–1094. DOI: 10.1109/ICDE.2015.7113358.

[Cho+03] Chomicki, J., ET AL. Skyline with presorting. In: Apr. 2003, 717–719. ISBN: 0-7803-7665-X. DOI: 10.1109/ICDE.2003.1260846.

[Cho+10] Cho, S.-R., ET AL. VSkyline: Vectorization for Efficient Skyline Computation. *SIGMOD Rec.* 39, 2 (Dec. 2010), 19–26. ISSN: 0163-5808. DOI: 10.1145/1893173.1893176. URL: https://doi.org/10.1145/1893173.1893176.

[Con+18] Conte, A., ET AL. D2K: Scalable Community Detection in Massive Networks via Small-Diameter k-Plexes. In: July 2018, 1272–1281. DOI: 10.1145/3219819.3220093.

[DBS18] Danisch, M., Balalau, O., AND Sozio, M. Listing K-Cliques in Sparse Real-World Graphs*. In: *Proceedings of the 2018 World Wide Web Conference*. WWW '18. International World Wide Web Conferences Steering Committee, Lyon, France, 2018, 589–598. ISBN: 9781450356398. DOI: 10.1145/3178876.3186125. URL: https://doi.org/10.1145/3178876.3186125.

[IP12] Im, H., AND Park, S. Group skyline computation. *Information Sciences* 188 (2012), 151–169. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2011.11.014. URL: https://www.sciencedirect.com/science/article/pii/S0020025511005998.

[KLP75] Kung, H. T., Luccio, F., AND Preparata, F. P. On Finding the Maxima of a Set of Vectors. *J. ACM* 22, 4 (Oct. 1975), 469–476. ISSN: 0004-5411. DOI: 10.1145/321906.321910. URL: https://doi.org/10.1145/321906.321910.

[LH09] Lee, J., AND Hwang, S.-w. SkyTree: Scalable Skyline Computation for Sensor Data. In: *Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data*. SensorKDD '09. Association for Computing Machinery, Paris, France, 2009, 114–123. ISBN: 9781605586687. DOI: 10.1145/1601966.1601985. URL: https://doi.org/10.1145/1601966.1601985.

[LH10] Lee, J., AND Hwang, S.-w. BSkyTree: Scalable Skyline Computation Using a Balanced Pivot Selection. In: *Proceedings of the 13th International Conference on Extending Database Technology*. EDBT '10. Association for Computing Machinery, Lausanne, Switzerland, 2010, 195–206. ISBN: 9781605589459. DOI: 10.1145/1739041.1739067. URL: https://doi.org/10.1145/1739041.1739067.

[Li+15] Li, R.-H., ET AL. Influential Community Search in Large Networks. *Proc. VLDB Endow.* 8, 5 (Jan. 2015), 509–520. ISSN: 2150-8097. DOI: 10.14778/2735479.2735484. URL: https://doi.org/10.14778/2735479.2735484.

[Li+18] Li, R.-H., ET AL. Skyline Community Search in Multi-Valued Networks. In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. Association for Computing Machinery, Houston, TX, USA, 2018, 457–472. ISBN: 9781450347037. DOI: 10.1145/3183713.3183736. URL: https://doi.org/10.1145/3183713.3183736.

[Liu+15] Liu, J., ET AL. Finding Pareto Optimal Groups: Group-Based Skyline. *Proc. VLDB Endow.* 8, 13 (Sept. 2015), 2086–2097. ISSN: 2150-8097. DOI: 10.14778/2831360.2831363. URL: https://doi.org/10.14778/2831360.2831363.

[LZY20] Li, Q., Zhu, Y., AND Yu, J. X. Skyline Cohesive Group Queries in Large Road-social Networks. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, 397–408. DOI: 10.1109/ICDE48307.2020.00041.

[MB83] Matula, D. W., AND Beck, L. L. Smallest-Last Ordering and Clustering and Graph Coloring Algorithms. *J. ACM* 30, 3 (July 1983), 417–427. ISSN: 0004-5411. DOI: 10.1145/2402.322385. URL: https://doi.org/10.1145/2402.322385.

[Pap+05] Papadias, D., ET AL. Progressive Skyline Computation in Database Systems. *ACM Trans. Database Syst.* 30, 1 (Mar. 2005), 41–82. ISSN: 0362-5915. DOI: 10.1145/1061318.1061320. URL: https://doi.org/10.1145/1061318.1061320.

[Par+09] Park, S., ET AL. Parallel Skyline Computation on Multicore Architectures. In: *2009 IEEE 25th International Conference on Data Engineering*. 2009, 760–771. DOI: 10.1109/ICDE.2009.42.

[Zha+14] Zhang, N., ET AL. On Skyline Groups. *Knowledge and Data Engineering, IEEE Transactions on* 26 (Apr. 2014), 942–956. DOI: 10.1109/TKDE.2013.119.

[Zha+19] Zhang, C., ET AL. Selecting the Optimal Groups: Efficiently Computing Skyline k-Cliques. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Association for Computing Machinery, Beijing, China, 2019, 1211–1220. ISBN: 9781450369763. DOI: 10.1145/3357384.3357991. URL: https://doi.org/10.1145/3357384.3357991.

[Zhu+17] Zhu, H., ET AL. Parallelization of group-based skyline computation for multi-core processors. *Concurrency and Computation: Practice and Experience* 29, 18 (2017). e4195 cpe.4195, e4195. DOI: https://doi.org/10.1002/cpe.4195. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4195. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4195.

[ZMC09] Zhang, S., Mamoulis, N., AND Cheung, D. W. Scalable Skyline Computation Using Object-Based Space Partitioning. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. SIGMOD '09. Association for Computing Machinery, Providence, Rhode Island, USA, 2009, 483–494. ISBN: 9781605585512. DOI: 10.1145/1559845.1559897. URL: https://doi.org/10.1145/1559845.1559897.