Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Yuzhang Guo April 4, 2019

Deep Learning for Skyline Queries

by

Yuzhang Guo

Li Xiong
Adviser

Department of Computer Science

Li Xiong

Adviser

Vaidy Sunderam

Committee Member

Bree Ettinger

Committee Member

2019

Deep Learning for Skyline Queries

By

Yuzhang Guo

Li Xiong

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Computer Science

2019

Abstract

Deep Learning for Skyline Queries
By Yuzhang Guo

Skyline computation is important in many applications. It identifies a set of skyline points that are not dominated by any other point and is used in multi-criteria data analysis and decision making. In this paper, we propose the first neural architecture for skyline, called neural skyline networks (SkyNet), to learn the spatial knowledge of skyline patterns. Given SkyNet, we can predict the skyline points rather than compute the skyline points using traditional algorithms. SkyNet can predict the skyline points in linear $O(n)$ time with high accuracy, where n is the number of points. To achieve higher accuracy, we further propose two spatial aware loss functions which distinguish the contribution/weights of non-skyline points. Extensive experiments show that our neural skyline networks are capable of high accuracy and are at least one order of magnitude faster than the state-of-the-art algorithms for skyline computation. The main contribution of this paper is to outline and evaluate the potential of a novel approach to compute computational geometry structures (e.g., skyline and convex hull), which complements existing works and arguably opens up an entirely new research direction for a decades-old field.

Deep Learning for Skyline Queries

By

Yuzhang Guo

Li Xiong

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Computer Science

2019

Acknowledgements

I would like to express my sincere gratitude to my thesis advisor Prof. Li Xiong for the continuous support for my study and research. Her guidance, motivation and immense knowledge helped me throughout my research and writing of this thesis.

I would also like to thank the rest of my thesis committee: Prof. Vaidy Sunderam and Dr. Bree Ettinger for their encouragement and insightful comments which incented me to widen my research from various perspectives.

I am grateful to my fellow labmate Dr. Jinfei Liu for enlightening me the first glance of the research and his help throughout the research.

Last but not the least, I would like to thank my parents for giving birth to me and supporting me throughtout my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

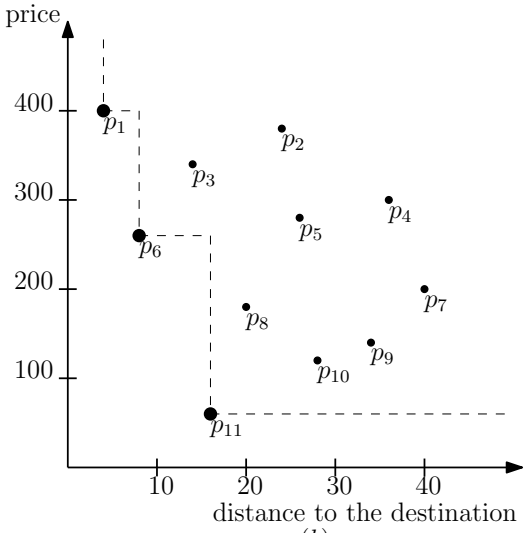
Skyline, also known as *Maxima* in computational geometry or *Pareto* in business management field, consists of the points from a set of multi-dimensional points for which no other point in the set exists that is better in at least one dimension and at least as good in every other dimension. Skyline has many important applications such as decision making and is widely used as a filter to significantly reduce the size of the returned query results.

Assume that we have a dataset P of n points. Each point p in the dataset P has d real-valued attributes and can be represented as a d-dimensional point $(p[1], p[2], ..., p[d]) \in \mathbb{R}^d$, where $p[i]$ is the i-th attribute of p. Given two points $p = (p[1], p[2], ..., p[d])$ and $p' = (p'[1], p'[2], ..., p'[d])$ in $\mathbb{R}^d$, p dominates p' if for every i, $p[i] \leqslant p'[i]$ and for at least one i, $p[i] < p'[i]$ $(1 \leqslant i \leqslant d)$. Given the set of points P, the skyline is defined as the set of points that are not dominated by any other point in P. In other words, the skyline represents the *best points* or Pareto optimal solutions from the dataset since the points within the skyline cannot dominate each other.

Figure 1.1(a) illustrates a dataset $P = \{p_1, p_2, ..., p_{11}\}$. Each records in the

hotel	distance	price
p_1	4	400
p_2	24	380
p_3	14	340
p_4	36	300
p_5	26	280
p_6	8	260
p_7	40	200
p_8	20	180
p_9	34	140
p_{10}	28	120
p_{11}	16	60

(a)



(b)

Figure 1.1: A skyline example of hotels.

dataset represents a hotel with two attributes: the distance to the destination and the price. Figure 1.1(b) shows the corresponding points in two dimensional space, where the x and y coordinates correspond to the two attributes of the hotels, respectively. To give an example of dominance, we can see that $p_5(26, 280)$ is better than $p_4(36, 300)$ in both attributes; thus, p_5 dominates p_4. The skyline of the dataset contains the points that are not dominated by any other points in the dataset, and are p_1, p_6, and p_{11}. Suppose the organizers of a conference need to reserve an hotel considering both distance to the conference destination and the price for participants, the skyline offers a set of best options or Pareto optimal solutions with various tradeoffs between distance and price: p_1 is the nearest to the destination, p_{11} is the cheapest, and p_6 provides a good compromise of the two factors. p_8 will not be considered as p_{11} is better than p_8 in both factors.

Motivation. In two dimensional space, the time complexity for computing skyline points is $O(n \log n)$. If the expected number of skyline points is small, we can employ the even more efficient output-sensitive algorithms [11] [13] to

compute skyline points in $O(n \log v)$ time, where v is the number of skyline points and $v \ll n$ in most cases. Although the traditional algorithms for computing skyline points in two dimensional space is efficient, the time complexity for computing skyline points in high dimensional space is still prohibitively high due to the "curse of dimensionality". In high dimensional space, the state-of-the-art algorithm for computing skyline points in $O(n \log^{d-1} n)$ time was proposed by Bentley in [3]. We can see that the time complexity increases exponentially as the number of dimensions d increases. Therefore, it is desirable to explore a more efficient algorithm for computing skyline.

Can we compute skyline points in linear time? Even in high dimensional space? Deep learning opens up the opportunity to learn a model that reflects the patterns in the dataset and thus to enable the automatic synthesis of specialized skyline structure.

Challenge. This is the first work to explore how to solve skyline problem with neural networks and choosing an appropriate model architecture is a challenge. Considering our skyline problem, the input and the output of skyline computation are sets and the lengths of the input set and the output set are arbitrary. Therefore, it is natural to employ the classic sequence-to-sequence model which uses an encoder (e.g., RNN and LSTM) to map an input sequence to an embedding and another (possible the same) decoder to map the embedding to an output sequence. At the encoding time step i, the encoder takes a vector representing i^{th} point from the database as its input, collects information for the point, and propagates it forward. Then, during the decoding phase, the model outputs one predicted skyline point at each decoding step. The content-based attentional mechanism augments the decoder by propagating extra contextual information from the input and achieves state-of-the-art performances in core

deep learning problems, e.g., translation, parsing, and video captioning.

However, these methods still require the size of the output dictionary to be fixed in advance. Because of this constraint, we cannot directly apply this framework to skyline problem where each input point is a potential candidate for output, so that the number of target classes depends on the length of the input. Pointer Networks can address this limitation by repurposing the attention mechanism to create pointers to input elements. However, each decoding step in pointer networks requires $O(n)$ time. Therefore, the total time for pointer networks is $O(nm)$ where m is the number of decoding steps. We still cannot achieve our goal of computing skyline points in linear time.

Contributions. In this paper, we propose the first neural skyline networks, SkyNet, to predict the skyline points in linear time. Because the skyline points are a subset rather than a sequence, we can predict the skyline points in one decoding step rather than k decoding steps in pointer networks. In SkyNet model, we take k points with the highest weight from n points in one decoding step. Because the k^{th} highest weight can be selected in linear $O(n)$ time by the classic selection algorithm [5], the decoder of SkyNet can be finished in $O(n)$ time. Because the encoder takes $O(n)$ time, the time complexity for SkyNet is $O(n)$ in total. Furthermore, to better capture the skyline structure, we propose two loss function variants, layer-based loss function and Euclidean distance-based loss function. For the layer-based loss function, the point in the lower layer takes more weight. Similarly, the point with near Euclidean distance to the query point takes more weight. Extensive experiments show that our proposed models are capable of high efficiency and accuracy, and both efficiency and accuracy are insensitive to the number of dimensions.

The main purpose/contribution of this paper is to outline and evaluate

the potential of a novel approach to compute computational geometry structures, which complements existing works and arguably opens up an entirely new research direction for a decades-old field. What we want to emphasize is that SkyNet supports a complement for the skyline community rather than a complete replacement of the traditional algorithms.

We briefly summarize our contributions as follows.

- We propose the first neural skyline networks, SkyNet, to learn skyline patterns. Given SkyNet, we can predict the skyline points in linear $O(n)$ time rather than compute the skyline points using traditional algorithms that have higher time cost.
- For the purpose of integrating the particular structure of skyline, we design two different loss functions. Each loss function has its own advantage.
- We conduct comprehensive experiments on the real and synthetic datasets. The experimental results show that SkyNet is efficient and accurate.

Organization. The rest of the thesis is organized as follows. Chapter 2 presents the related work of skyline and neural networks. The problem definition and preliminaries including several neural networks are given in Chapter 3. We propose the first neural skyline networks and design two different loss functions which capture the particular structure of skyline in Chapter 4. We report the experimental results and findings in Chapter 5. Chapter 6 concludes the paper and discusses the future directions.

Chapter 2

Related Work

In this chapter, we briefly review the related work to the skyline problem and neural networks.

2.1 Skyline

Skyline is a fundamental problem in computational geometry because it is an interesting characterization of the boundary of a set of points. Since the introduction of the skyline operator by Borzsonyi et al. [6], Skyline has been extensively studied in database field. To facilitate the skyline query, skyline diagram was proposed in [15]. To protect data privacy and query privacy, secure skyline query was studied in [14]. [9] studied the skyline in P2P systems. [16, 18,22] studied the skyline on the uncertain dataset. [17] studied the continuous skyline over distributed data streams. [12, 21] generalized the original skyline definition for individual points to permutation group-based skyline for groups.

The theoretical state-of-the-art algorithms for computing skyline points are [13] [11] [3]. In two dimensional space, [13] [11] proposed the efficient $O(n \log k)$ time complexity algorithms based on subtle divide-and-conquer paradigm, where

$k \leqslant n$ is the number of skyline points. [10] proved an $\Omega(n \log n))$ lower bound for the skyline problem, which means that we cannot design an algorithm with better time complexity than $O(n \log n)$. In high dimensional space, the skyline points can be computed in $O(n \log^{d-1} n)$ time based on the empirical cumulative distribution function which was proposed in [3]. Although this algorithm is effective when d is small, the time cost is still prohibitively high when d is large. Therefore, a linear time algorithm is still highly desired. In this paper, our goal is to train the neural skyline networks which can be used to decode the skyline points given any dataset in linear $O(n)$ time while sacrificing a little accuracy.

2.2 Neural Networks

The classic sequence-to-sequence models were proposed by Sutskever et al. [19] and Cho et al. [8]. They have been established as powerful models and have been widely used in many learning tasks such as machine translation and question answering. A sequence-to-sequence model consists of two recurrent neural networks (RNNs) that act as an encoder and a decoder respectively. The encoder compresses the information of an input sequence of variable length to a fixed length vector, and the decoder then generates a variable length target sequence from the vector. The encoder and the decoder are trained together to learn the conditional probability of a target sequence given an input sequence. However, the performance of a basic encoder-decoder structure deteriorates rapidly as the length of an input sentence increases, because of the limited amount of information the fixed length vector that connects the encoder and the decoder can contain. [7]. In order to address this issue, Bahdanau et al.

proposed an attention mechanism [1]. The attention mechanism allows the decoder to adaptively choose a subset of input elements to focus on at each step of decoding. Therefore, instead of solely depending on a fixed-length vector to generate outputs, the decoder is informed that which parts of the input sequence play important roles when decoding. However, these methods require the size of the dictionary to be fixed in advance, which is not suitable for the problems that the size of the output dictionary depends on the length of the input sequence. To address this issue, pointer networks model [20] was proposed. Instead of using attention to blend hidden units of an encoder to a context vector at each decoder step, pointer networks use attention as a pointer to select a member of the input sequence as the output. However, the time complexity for pointer networks is $O(nm)$ which is high, where m is the number of decoding steps.

Chapter 3

Definitions and Preliminaries

In this chapter, we present definitions in Section 3.1 and review the sequence-to-sequence framework [19] and attention models [2] in Section 3.2 and 3.3. In Section 3.4, we present pointer networks [20] which serve as the performance benchmark for our proposed neural skyline networks, SkyNet. We provide the detailed explanation of the architecture of SkyNet in Chapter 4.

3.1 Definitions

We show the formal definition of skyline as follows.

Definition 3.1. (**Skyline**). Given a dataset P of n points in d-dimensional space. Let p and p' be two different points in P, p dominates p', denoted by $p \prec p'$, if for all i, $p[i] \leqslant p'[i]$, and for at least one i, $p[i] < p'[i]$, where $p[i]$ is the i^{th} dimension of p and $1 \leqslant i \leqslant d$. The skyline points are those points that are not dominated by any other point in P.

The theoretical state-of-the-art algorithms for computing skyline points are [13] [11] [3]. In two dimensional space, [13] [11] proposed efficient $O(n \log k)$

time complexity algorithms based on subtle divide-and-conquer paradigm, where k is the number of skyline points. In high dimensional space, the skyline points can be computed in $O(n \log^{d-1} n)$ time based on the empirical cumulative distribution function which was proposed in [3], where d is the number of dimensions. Although those proposed algorithms are effective to some extent, computing skyline points can be time consuming, especially for high dimensional space. In practice, data usually has multiple dimensions. Therefore, a linear time algorithm is desired. However, the lower bound for computing skyline is $O(n \log n)$ even in two dimensional case [10]. In this paper, our goal is to train the neural skyline networks which can be used to predict the skyline points given any dataset in linear $O(n)$ time but sacrificing a little accuracy.

3.2 Sequence-to-sequence Framework

The sequence-to-sequence framework provides an effective approach to map sequences from one domain to sequences in another domain. Consider a training sequence pair (X, Y), where $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_m\}$ are sequences with length n and m respectively. The sequence-to-sequence framework learns to model the conditional probability $P(Y|X)$. Usually, sequence-to-sequence models consist of two main components, an encoder and a decoder, as depicted in Figure 3.1(a). Each component is typically a recurrent neural network (RNN) or a long short-term memory network (LSTM). The encoder processes and encodes the input sequence X into a fixed-dimension representational vector v that contains useful information from X. The representation v is then forwarded to the decoder which generates the output sequence Y conditioned on v.

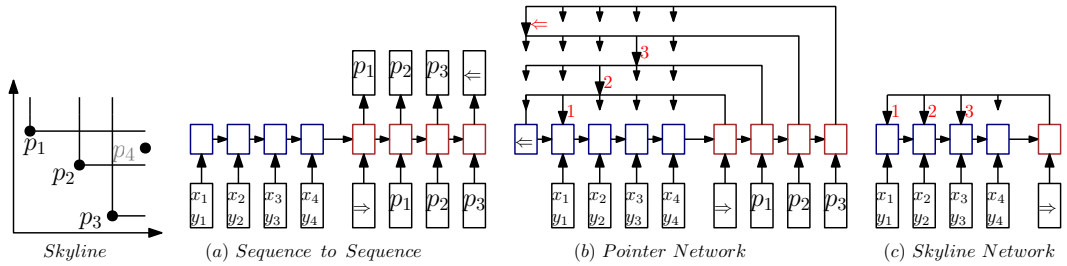


Figure 3.1: Models.

To illustrate how to build the conditional probability $P(Y|X)$, we first decompose it using the chain rule, i.e.,

$$P(Y|X) = \prod_{i=1}^{m} P(y_i|y_1, y_2, ..., y_{i-1}, X)$$

Since the encoder encodes X into the representational vector v, we can replace X with v and have

$$P(y_i|y_{1:i-1}, X) = P(y_i|y_{1:i-1}, v)$$

where $y_{1:i-1}$ represents $y_1, y_2, ..., y_{i-1}$. With an RNN, each conditional probability is modeled as

$$P(y_i|y_{1:i-1}, v) = f(d_i)$$

and

$$d_i = RNN(d_{i-1}, y_{i-1})$$

where $f()$ is a nonlinear function that outputs the probability of y_i based on the previous information and d_i is the hidden state at decoding step i, which encodes the information of v along with y_1,y_2,...,y_{i-1}.

In the training phase, the parameters of the model are learned by minimizing

the cross entropy loss:

$$\mathcal{L} = -\log \prod_{i=1}^{m} P(y_i | y_{1:i-1}, v)$$

Usually, inputs of RNNs are real-valued vectors, so a token embedding layer is always needed to embed non-numeric inputs (e.g. text data in question answering). However, in our problem, the inputs to the network are points in Euclidean space so that embedding layer is not necessary.

Under the assumption that the length of an output sequence is $O(n)$, sequence-to-sequence models have the time complexity of $O(n)$.

3.3 Attention Model

In the sequence-to-sequence framework, the information describing an input sequence is compressed into a representational vector v, which is the only information visible to the decoder when generating the output sequence. The potential issue of this approach is that the fixed dimension of v restricts the amount of information flowing through to the decoder RNN. Since input sequences can have arbitrary lengths, v may not hold enough information regarding inputs, especially when the length of an input sequence is large, causing models to suffer from low predicting power. To break the bottleneck of the basic encoder-decoder architecture, [2] proposed an attention mechanism that allows the decoder to focus on parts of the input sequence when generating a target element at each decoding step. Intuitively, the attention mechanism provides a context for the decoder by assigning weights to each encoder hidden state.

Let us denote encoder and decoder hidden states as $(e_1, ..., e_n)$ and $(d_1, ..., d_m(\mathcal{P}))$,

respectively. Mathematically, at decoding step i, the attention mechanism computes

$$u_j^i = g(d_i, e_j)$$

$$a_j^i = softmax(u_j^i) \quad j \in (1, ..., n)$$

$$c_i = \sum_{j=1}^{n} a_j^i e_j$$

where $g()$ is an alignment function that scores how well the j-th element of input sequence matches the output element at position i. The attention vector a^i is then computed by normalizing the vector u^i of length n using the softmax function. Finally, c_i is computed as a weighted sum over the encoder hidden states and serves as the context vector. c_i and d_i are then concatenated to form a new hidden state from which the model makes predictions.

Note that for each step of decoding, the attention model performs n operations to compute an attention vector over the input sequence of length n. If the length of the output sequence is m, the model has the time complexity of $O(nm)$.

3.4 Pointer Networks

The sequence-to-sequence and attention models described in section 3.2 and 3.3 require the output dictionary size to be fixed. However, for skyline problem, the number of potential output at each decoding step is equal to the length of the input, which is variable, because different databases contains different number of records. Thus, we need to train separate models for databases with different size when using sequence-to-sequence and attention models.

We now describe a variation of the attention model, named pointer net-

works proposed in [20]. This model allows us to solve problems where the output dictionary size varies with the number of elements in the input sequence. It is suitable for various combinatorial problems where output is a subset or sub-sequence (permutation) of the input, such as computing convex hulls and Travelling Salesman problem. The Skyline problem, in which the skyline points is a subset of all points in a database, also belongs to this category.

Ptr-Nets propose a novel neural attention mechanism. Instead of using the attention vector to compute the weighed sum over all encoder hidden states, it directly uses attention to model the conditional probability and select a member from the input sequence at each decoding step. For example, in Figure 3.1(b), at the first decoding step, the model outputs an index pointing to the first position of the input sequence. Then, the element corresponding to the output index is fed to the decoder at the next decoding step together with the previous decoder hidden state. Consider a training pair $(\mathcal{X}, \mathcal{C}^{\mathcal{X}})$, where $\mathcal{X} = \{x_1, x_2, .., x_n\}$ is a sequence of points and $\mathcal{C}^{\mathcal{X}} = \{C_1, C_2, ..., C_m\}$ is a sequence of indices, each between 1 and n. Ptr-Nets compute the attention as follows:

$$u_j^i = g(d_i, e_j)$$

$$P(C_i|C_{1:i-1}, v) = softmax(u^i)$$

where $g()$ is an alignment function as described in section 3.3 and d_i and e_j are the i-th decoder and j-th encoder hidden states, respectively. The softmax function normalizes the vector u^i to be an output probability distribution over the dictionary of the input sequence. At each decoding step, the model returns the index of the input element with the highest probability.

Similar to the attention model, for an input sequence of length n and an

output sequence m, the Ptr-Net has the time complexity of $O(nm)$.

Chapter 4

Proposed Models

The accuracy of Ptr-Nets for predicting skyline points is acceptable, but the time complexity $O(nm)$ is high, where n is the number of points and m is the number of skyline points. In this chapter, we propose a novel model, SkyNet, to efficiently predict skyline points. Keeping the accuracy advantage of pointer networks, SkyNet has another desired advantage on time complexity. We can predict skyline points using SkyNet model in linear $O(n)$ time, which is very appealing for big datasets.

4.1 Pointer Networks with an additional skyline layer

We can employ Ptr-Nets to solve the skyline problem directly. Ptr-Nets employ the cross-entropy as the loss function, which measures the performance of a classification model whose output is a probability value between 0 and 1. The cross-entropy increases as the predicted probability diverges from the actual label. We set the labels of skyline points as 1, and the labels for all other

points as 0 in the loss function. Although the time complexity for Ptr-Nets $O(nm)$ is high, it can achieve high prediction accuracy.

Can we achieve a better accuracy? Our observation is that those non-skyline points contribute differently to *Domination Range*. Therefore, we may distinguish those non-skyline points in the training phase. We first show the definition of domination range and then show the definition of skyline coverage as follows.

Definition 4.1. (**Domination Range**). Given a point p in a d dimensional space, the domination range $\mathcal{DR}(p)$ of Point p is the range that if any point p' lies in this range, $p \prec p'$. Given a dataset $P = \{p_1, p_2, ..., p_n\}$ in the d dimensional space, the domination range of P is the union of the domination ranges of Points $p_1, p_2, ..., p_n$, i.e., $\mathcal{DR}(P) = \mathcal{DR}(p_1) \bigcup \mathcal{DR}(p_2) \bigcup ... \bigcup \mathcal{DR}(p_n)$.

For a dataset P with the set of skyline points SKY, it is natural to see that the $\mathcal{DR}(P) = \mathcal{DR}(SKY)$. Therefore, it is reasonable to employ domination range to measure the predicted set of skyline points. We show the definition of Skyline Coverage as follows.

Definition 4.2. (**Skyline Coverage**). Given a point p in a d dimensional space, we use SKY_{pred} to denote the predicted set of skyline points. The skyline coverage for the predicted set of skyline points is

$$\textbf{SkyCov}(SKY_{pred}) = \frac{\mathcal{DR}(SKY_{pred})}{\mathcal{DR}(SKY)}$$

If the predicted skyline points are the exact skyline points, the skyline coverage will be 100%. However, it is difficult to predict all the skyline points exactly. Therefore, we want to maximize the skyline coverage of the predicted skyline points. A natural idea is to choose the points with larger domination

range which usually lie in lower skyline layers. We show the definition of skyline layers as follows.

Skyline Layers [12]. We present a structure representing the points and their dominance relationships based on the notion of skyline layers. A formal definition is presented as follows.

Definition 4.3. (**Skyline Layers**). Given a dataset P of n points in a d-dimensional space. The set of skyline layer $layer_1$ contains the skyline points of P, i.e., $layer_1 = skyline(P)$. The set of $layer_2$ contains the skyline points of $P \backslash layer_1$, i.e., $layer_2 = skyline(P \backslash layer_1)$. Generally, the set of $layer_j$ contains the skyline points of $P \backslash \bigcup_{i=1}^{j-1} layer_i$, i.e., $layer_j = skyline(P \backslash \bigcup_{i=1}^{j-1} layer_i)$. The above process is repeated iteratively until $P \backslash \bigcup_{i=1}^{j-1} layer_i = \varnothing$.

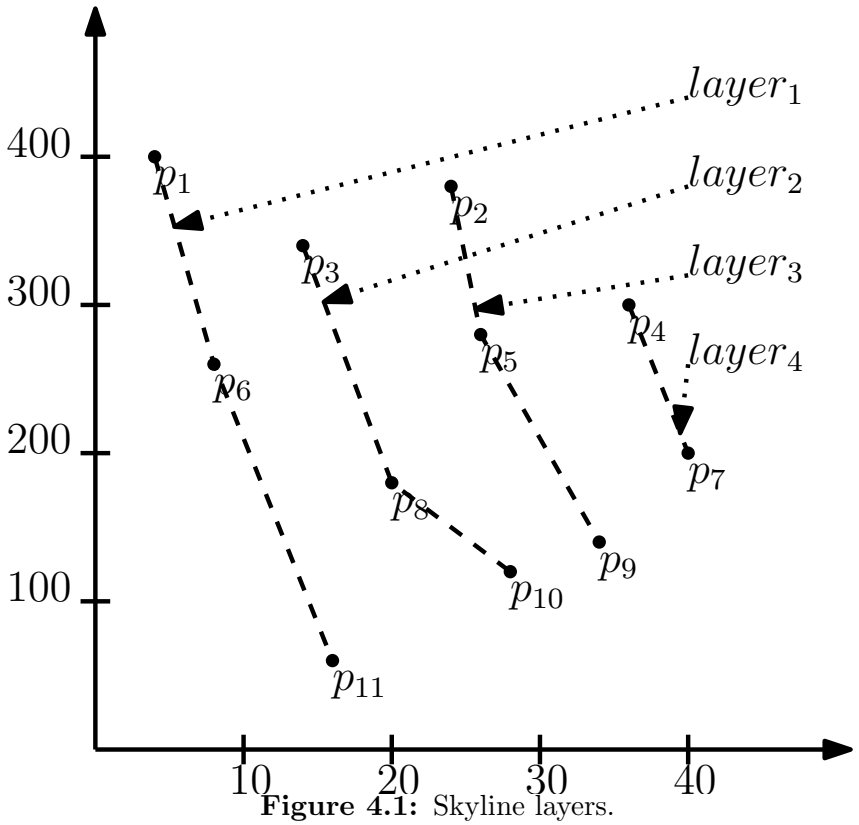


Figure 4.1: Skyline layers.

An example of skyline layers of Figure 1.1 is shown in Figure 4.1. It is easy

to see from Definition 4.3 that for a point p, if there is no point in $layer_{i-1}$ that can dominate p, p should be in $layer_{i-1}$ or a lower layer.

In the traditional pointer networks, skyline points are labeled as the positive class 1, whereas all other points are labeled as 0. We then train the model to predict the skyline points. However, different points contribute differently during the learning stage of the model, and we think the points in the second layer can further help the model to learn the distribution of points and distinguish skyline points from others. Thus, we label points in both the first layer and second layer as positive class and train a new model. The experiments in Chapter 5 show that pointer networks with an additional skyline layer outperforms pointer networks with only first skyline layer in terms of accuracy.

4.2 Neural Skyline Networks

Although pointer networks with an additional skyline layer can achieve higher accuracy, the time complexity is still $O(nm)$. Due to the massive computation in the model, the running time of Ptr-Nets cannot compete with the state-of-the-art traditional algorithms.

Can we achieve a better efficiency? The pointer network predicts one skyline point at each decoding step and requires m decoding steps to predict all the skyline points, where m is the number of skyline points. For each decoding step, the model needs to compute the attention vector over the input dataset with n points, which is the main reason that slows down the model. For problems with outputs that have an intrinsic order (e.g., finding the correct permutation of the input in Travelling Salesman Problem), we cannot avoid these multiple decoding steps because it is necessary to predict each output element based on

the input and the previous predicted output elements. However, in our skyline problem, the output is a set rather than a sequence. The probability of each output element does not depend on previous predicted output elements, so we can use the attention mechanism to directly model the probability of i^{th} point being a skyline point conditioned solely on the representational vector v, which contains spatial information of all points in the database. Intuitively, we can treat the attention mechanism as a binary classification model taking v and j^{th} encoder hidden state as inputs and compute the probability of j^{th} point being the skyline point. It is computed as follows:

$$u_j = g(v, e_j) \quad j \in (1, ..., n)$$

$$P(Sky_j|v) = sigmoid(u_j)$$

where $g()$ is an alignment function in the attention mechanism that scores how likely j-th point is a skyline point among all points in the database whose information is encoded in v. $P(Sky_j|v)$ denotes the probability of j-th point being skyline given representational vector v. Note that $(P(Sky_1|v), P(Sky_2|v), ..., P(Sky_n|v)$ constitutes the attention vector computed by the attention mechanism at the first decoding step. Therefore, SkyNet is able to solve the skyline problem in one decoding step and has the time complexity of $O(n)$.

Under the setting of our attention mechanism that functions as a binary classifier, the cross entropy loss function is given as follows,

$$\mathcal{L} = -\sum_{j=1}^{n} L_j \log P(Sky_j|v) + (1 - L_j)(\log(1 - P(Sky_j|v)))$$

where $L_j \in \{0, 1\}$ is the label of j-th point in a database with 1 denoting the

point is a skyline and 0 otherwise.

Inspired by the improved performance of the Ptr-Net with an additional skyline layer, we want to explore whether the domain knowledge of the skyline problem can help SkyNet achieve a better performance. We observe that the cross entropy loss function presented above penalizes the output points with equal weight. For example, if a model predicts that p_3 and p_4 in Figure 4.1 have the same probability of being skyline points, the model penalizes them equally. This is not a good penalizing strategy. The point p_3 is closer to the first skyline layer and should be more acceptable than p_4. In section 4.2.1 and 4.2.2, we propose two spatial aware loss functions that are able to encode the spatial information of points and ameliorate the deficiency in the cross entropy loss.

4.2.1 Neural Skyline Networks with layer based loss function

Since points in lower skyline layers are more acceptable than points in higher skyline layers, we propose a layer based loss function, which incorporates the layer knowledge by assigning a weight to each point. The weight of a point is inversely proportional to the layer number of the point, so lower layer the point is in, the higher the weight we assign to it. The layer based loss function is given as follows,

$$\mathcal{L}_{layer} = -\sum_{j=1}^{n} w_j \log P(Sky_j|v) + (1 - w_j)(\log(1 - P(Sky_j|v)))$$

where

$$w_j = \frac{1}{layer_j}$$

is the weight assigned to the j-th point and $layer_j$ denotes the layer number of the j-th point.

In Figure 4.2, we show the assigned probabilities of a set of 100 points being skyline points. The probabilities are predicted by a model trained with the layer based loss function. The darker the color of a point, the higher probability the model predicts. We can see that the loss function enables the model to learn the layer structure of a set of points.

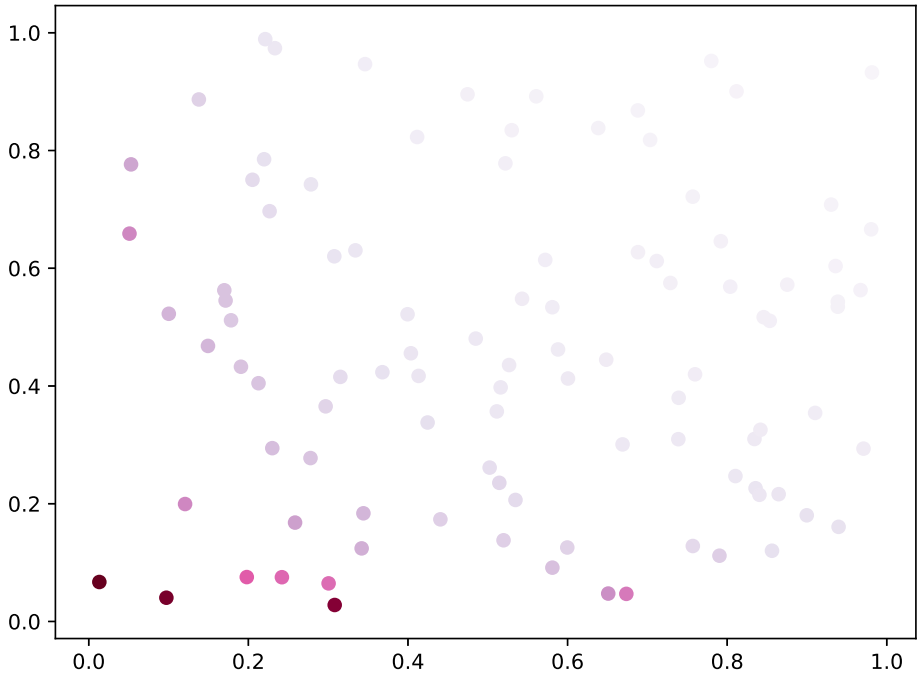


Figure 4.2: Illustration of predicted probabilities over 100 points by a model trained with layer based loss.

4.2.2 Neural Skyline Networks with Euclidean distance based loss function

Generally speaking, points near the origin are more acceptable than points far away from the origin and are more likely to be skyline points. Euclidean

distance based loss function utilizes this knowledge and is given as follows,

$$\mathcal{L}_{layer} = -\sum_{j=1}^{n} w_j \log P(Sky_j|v) + (1 - w_j)(\log(1 - P(Sky_j|v)))$$

The weights for skyline points are 1 and for non-skyline points are

$$w_j = \frac{\exp(1/dist_j)}{\sum_{i=1}^{n} \exp(1/dist_i)}$$

where $dist_j$ is the distance between the j-th point and the origin and n is the number of points in a dataset. Since $1/dist_j$ can have very small value for points near the origin, we use the softmax function to normalize the weights.

In Figure 4.3, we show the assigned probabilities of a set of 100 points being skyline points. The probabilities are predicted by a model trained with the Euclidean distance based loss function. The darker the color of a point, the higher probability the model predicts. We can see that the model assigns high probabilities for skyline points. For non-skyline points, the model assigns higher probabilities for points near the origin, but we cannot distinguish the difference in their color in Figure 4.3. The reason is that after applying softmax function, the weight assigned to each non-skyline point is small comparing to 1, which is the weights assigned to skyline points. Although the color is indistinguishable, each point still contributes differently during the learning process of the model, and the points closer to the origin are actually predicted with a slightly higher probability of being skyline points.

Another way to interpret the spatial aware loss functions is to consider the fundamental meaning of cross entropy. In information theory, cross entropy is commonly used to quantify the difference between two probability distributions. In the basic cross entropy loss function, labels can be reinterpreted

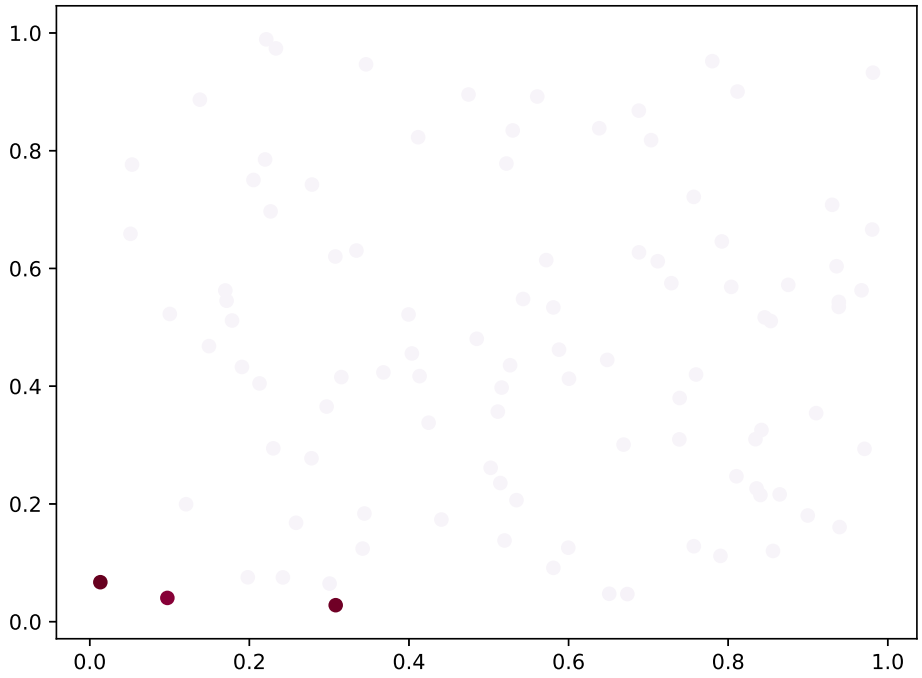


Figure 4.3: Illustration of predicted probabilities over 100 points by a model trained with layer based loss.

as probabilities of being a skyline point but can only take values in the set $\{0, 1\}$. However, in spatial aware loss functions, the range of label value is not restricted to $\{0, 1\}$. Labels can take values in a continuous range $[0, 1]$, and the values are computed by $\frac{1}{layer_j}$ in layer based loss and $\frac{\exp(1/dist_j)}{\sum_{i=1}^{n} \exp(1/dist_i)}$ in Euclidean distance based loss.

4.3 Discussion

If we compare the skyline problem to the task of choosing the k most representative/coverage points from a dataset of n points, the mechanism of pointer networks is a local optimization while the mechanism of neural skyline networks is a global optimization. The pointer network is more like a greedy algorithm that chooses the locally optimal point at each decoding steps based on the in-

put and the previous outputs, and it requires k decoding steps. Neural skyline networks search the global optimal subset with size k in one decoding step.

Chapter 5

Experiments

In this chapter, we present experimental studies evaluating our proposed pointer networks with an additional skyline layer and neural skyline networks.

5.1 Experiment Setup

We ran experiments on a machine with an Intel Core i7-8700K and two NVIDIA GeForce GTX 1080 Ti running Ubuntu with 64GB memory. We implemented the following algorithms in Python 3.5 and TensorFlow 1.9.

- **TRAD:** state-of-the-art traditional skyline computation algorithm [3].
- **PtrNet:** Pointer Networks.
- **PtrNet+:** Pointer Networks with an additional skyline layer.
- **SkyNet:** Neural Skyline Networks with cross entropy loss function.
- **SkyNet-layer:** Neural Skyline Networks with layer based loss function.
- **SkyNet-dist:** Neural Skyline Networks with Euclidean distance based loss function.

We used synthetic datasets and a real NBA dataset in our experiments. We built a dataset[1] that contains 2384 NBA players who are league leaders of playoffs. Each player has five attributes (Points, Rebounds, Assists, Steals, and Blocks) that measure the player's performance.

5.2 Average Number of Skyline Points

Although the expected number of skyline points for independent datasets is $O((\ln n)^{d-1})$ in theory [4], the big O notation is not accurate enough for practical applications. Therefore, we experimentally evaluate the number of skyline points given the number of points n and the number of dimensions d. For each setting of different n and d, we randomly generate 10000 datasets. We then compute the skyline points for the datasets and take the average number of skyline points. The average number of skyline points for different n and different d is shown in Table 5.1. We use the calculated number of skyline points to set the number of decoding steps in pointer networks and the number of points with the largest probability being skyline points in the decoding step of neural skyline networks.

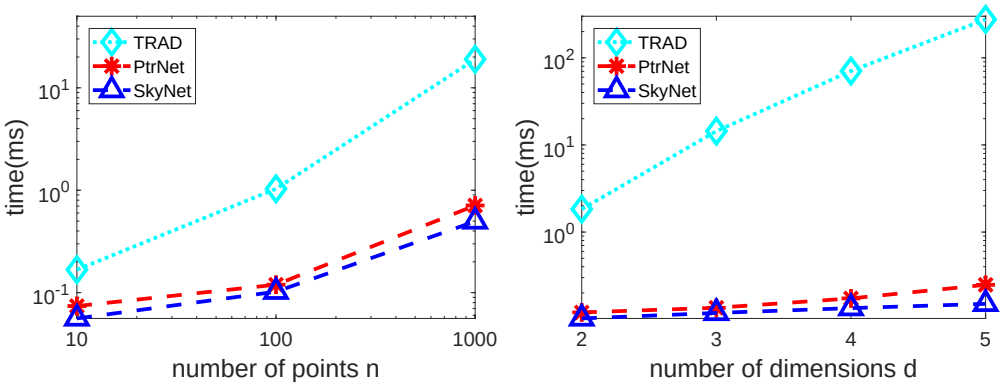
Table 5.1: Average number of skyline points.

number of points	10	100	1000	10k	100k
# of skyline points in 2D	2.9	5.2	7.5	9.5	11.8
# of skyline points in 3D	5.1	14.3	28.7	49.1	73.2
# of skyline points in 4D	6.8	27.9	76.8	165.2	304.2
# of skyline points in 5D	8.1	43.7	158.2	415.2	947.6

[1]Extracted from http://stats.nba.com/leaders/alltime/?ls=iref:nba:gnav on 04/15/2015.

5.3 Prediction Time Cost

Figures 5.1(a)(b) present the computation time for TRAD and the prediction time cost for PtrNet, PtrNet+, SkyNet, SkyNet-layer, and SkyNet-dist. PtrNet and PtrNet+ have the same time cost because they employ the same model. The same to the models of SkyNet, SkyNet-layer, and SkyNet-dist. Therefore, we only report the prediction time cost for PtrNet and SkyNet. Figure 5.1(a) presents the time cost for different number of points n in two dimensional space. Both PtrNet and SkyNet significantly outperform TRAD and almost linearly increase with increasing number of points n, which verifies the efficiency of our proposed models. PtrNet has a good performance because the time complexity for PtrNet is $O(nm)$ and the number of skyline points m is small, i.e., PtrNet does not need too many decoding steps for the small datasets. However, we can see that the difference between PtrNet and SkyNet increases with the increasing n because the number of skyline points increases with the increasing number of points n and PtrNet needs to decode more steps. Figure 5.1(b) present the time cost for different number of dimensions d. Similar to the performance in different number of points n, in different number of dimensions d, both PtrNet and SkyNet significantly outperform TRAD. Furthermore, TRAD exponentially increases with the increasing d because d is in the exponential part of the time complexity $O(n \log^{d-1} n)$. PtrNet almost exponentially increases with the increasing d because the number of skyline points exponentially increases with the increasing d. SkyNet has the best performance, especially when the number of skyline points is large.



(a) $d = 2$ (b) $n = 100$

Figure 5.1: Time cost.

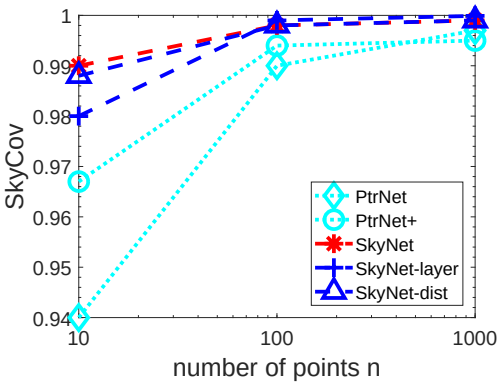
5.4 Prediction Accuracy

In this subsection, we evaluate the prediction accuracy of our proposed algorithms by the SkyCov, the Recall, and the Precision. Even we choose the points randomly rather than compute the skyline points from the given datasets, those chosen points also have the SkyCov named as Basic SkyCov, i.e., the SkyCov accuracy we can achieve even we randomly choose points. To better illustrate the accuracy gain of our proposed models, we compute the basic SkyCov for the datasets with size 10, 100, $1k$, $10k$, $100k$ by randomly choosing the expected number of skyline points as shown in Table 5.1. For example, for ten points in two dimensional space, we randomly generate 10000 datasets with 10 points. For each dataset, we randomly choose $2.9 \approx 3$ points as a set S_{random}, and then compute $\mathcal{DR}(S_{random})$ for this set; we also compute the set of skyline points as $S_{skyline}$, and then compute $\mathcal{DR}(S_{skyline})$. The basic SkyCov equals to $\frac{\mathcal{DR}(S_{random})}{\mathcal{DR}(S_{skyline})}$. For each size, we compute 10000 times and take the average score. The basic SkyCov is shown in Table 5.2.

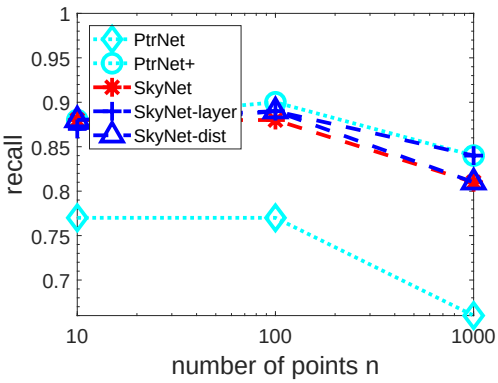
Skyline is widely used as a filter. Therefore, it is reasonable/acceptable to

Table 5.2: Basic SkyCov of random points.

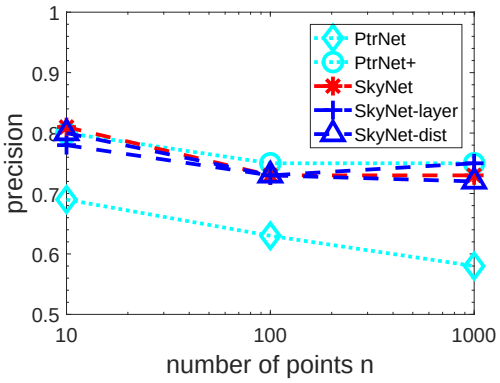
number of points	10	100	1000	10k	100k
SkyCov in 2D	0.66	0.67	0.69	0.73	0.76
SkyCov in 3D	0.72	0.69	0.73	0.78	0.83
SkyCov in 4D	0.81	0.72	0.74	0.79	0.85
SkyCov in 5D	0.86	0.76	0.76	0.80	0.86



(a) SkyCov



(b) Recall

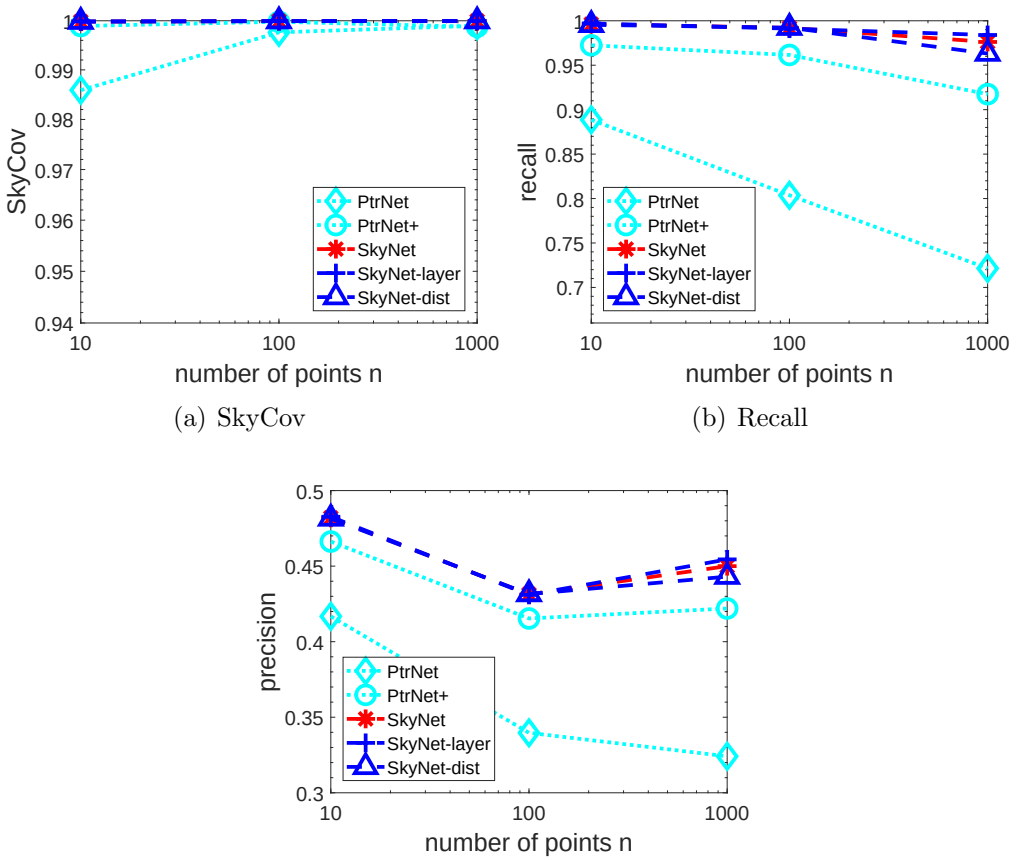


(c) Precision

Figure 5.2: The impact of number of points n on the accuracy.

returned more points than the number of skyline points and the recall is more important than the precision for predicting skyline points. Figures 5.3(a)(b)(c) present the SkyCov, the Recall, and the Precision with varying number of points n by returning twice the average number of skyline points given any datasets. For example, our models output 3 points given a dataset with 10 points in Figure 5.2. In this subsection, our models output 6 points. It is natural to

see that the recall by returning twice the average number of skyline points is much higher than the recall by returning the average number of skyline points. For example, both SkyNet and its variants achieve around 0.97 recall, which is highly desirable in practice.



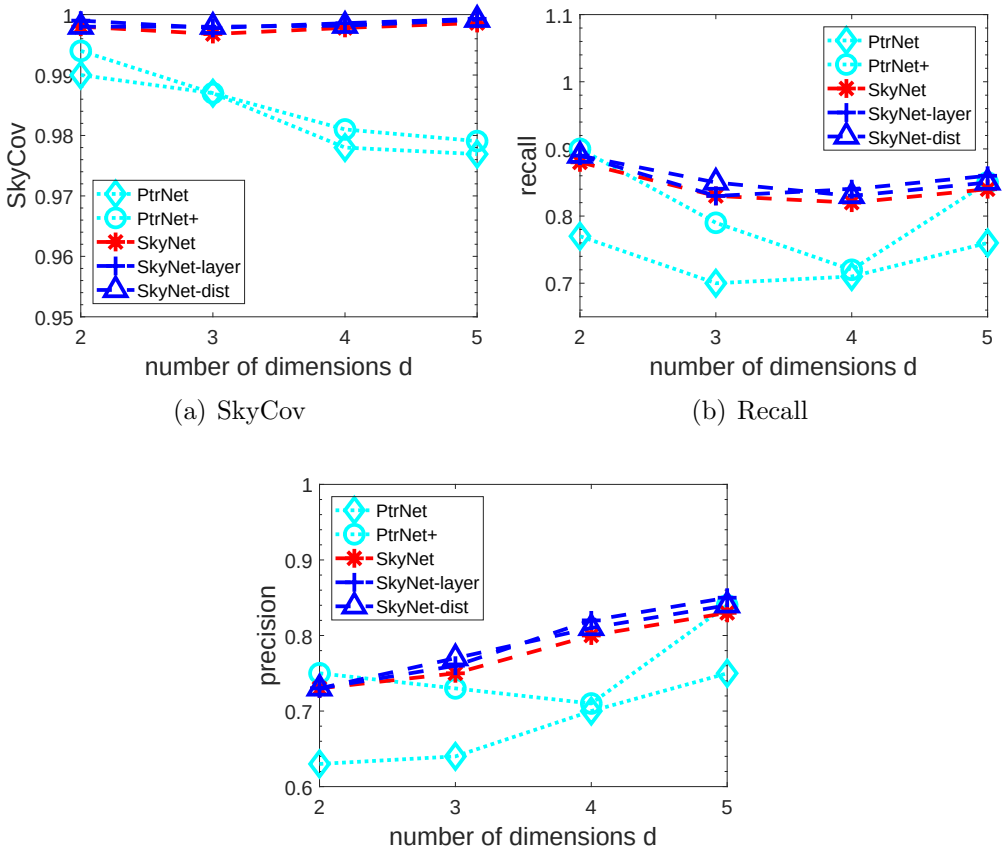
(a) SkyCov (b) Recall

(c) Precision

Figure 5.3: The impact of number of points n on the accuracy.

Figures 5.2(a)(b)(c) present the SkyCov, the Recall, and the Precision with varying number of points n for PtrNet, PtrNet+, SkyNet, SkyNet-layer, and SkyNet-dist in two dimensional space. All our proposed models outperform the basic pointer networks model in all three accuracy metrics. For the skyline coverage metric, all the models including PtrNet perform very well, e.g., for two dimensional case, PtrNet has the lowest SkyCov 0.94 while the basic SkyCov

is only around 0.68. Furthermore, all our proposed models outperform PtrNet. For example, PtrNet+ has the lowest SkyCov 0.968 which is much higher than 0.94 of PtrNet. Comparing different models, both SkyNet and its variants outperform the pointer networks model, which verifies the advantages of our proposed neural skyline networks. For both the recall and the precision, our proposed models significantly outperform the basic pointer networks model. Both the recall and the precision are higher than 0.75 for all our proposed models.



(a) SkyCov

(b) Recall

(c) Precision

Figure 5.4: The impact of number of dimensions d on the accuracy.

Figures 5.4(a)(b)(c) present the SkyCov, the Recall, and the Precision with varying number of dimensions d for PtrNet, PtrNet+, SkyNet, SkyNet-layer,

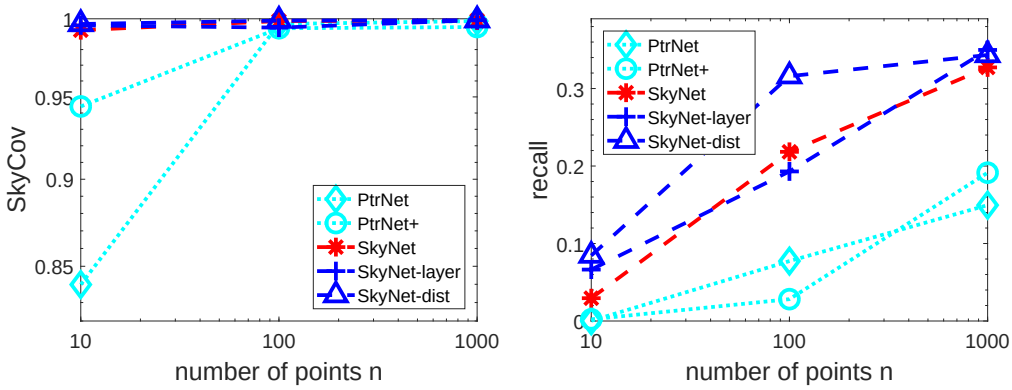
and SkyNet-dist when $n = 100$. For all our proposed models, the number of dimensions d does not have a significant impact on all the three metrics. The reason is that we embed each point in one encoding/decoding step and the number of dimensions d cannot affect the model very much. Both SkyNet and its variants outperform PtrNet and PtrNet, which verifies the advantages of our proposed neural skyline networks model. Both SkyNet-layer and SkyNet-dist outperform SkyNet, which verifies the efficiency of our proposed loss functions.

5.5 Model Transformability

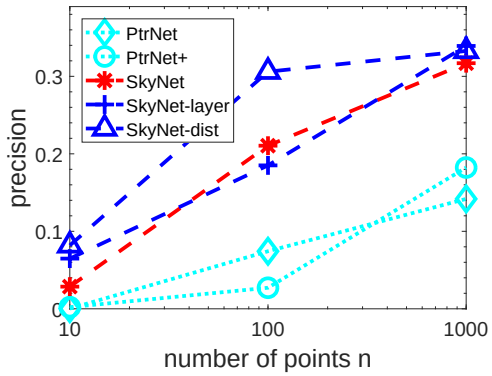
In this subsection, we evaluate the model transformabilities of our proposed models, i.e., we can use a model trained on datasets with a points to predict the skyline points of a dataset with b points, where $a \neq b$.

Figures 5.5(a)(b)(c) present the SkyCov, the Recall, and the Precision on the datasets of 100000 points predicted by the models that are trained by the datasets with 10, 100, and 1000 points, respectively. For the SkyCov metric, all the models including PtrNet significantly higher than the basic SkyCov 0.76, which verifies the efficiency of those neural network models. All our proposed models outperform PtrNet, which verifies the high efficiency of our propose models. For our proposed SkyNet and its variants, the SkyCov is almost 1 and the basic SkyCov is 0.66, 0.67, and 0.68, respectively, which verifies the high transformabilities of our problem model. Even for the models trained by the datasets with 10 points, the SkyCov for predicting the datasets with 100000 points is almost 1, which is significantly desirable. For the recall and the precision, the scores increase with the increasing number of points trained by the proposed models. The reason is that the trained models with

more points capture more spatial knowledge of skyline patterns. Comparing different models, both SkyNet and its variants significantly outperform both PtrNet and PtrNet+, which verifies the gain of our proposed neural skyline networks. Furthermore, SkyNet-dist outperforms SkyNet, which verifies the gain of our proposed loss function for skyline. Although the SkyCov of our proposed models is very high, both the recall and the precision are not so high. The reason is that although the predicted points are not exactly the skyline points, they are very closely to the skyline points.



(a) SkyCov (b) Recall

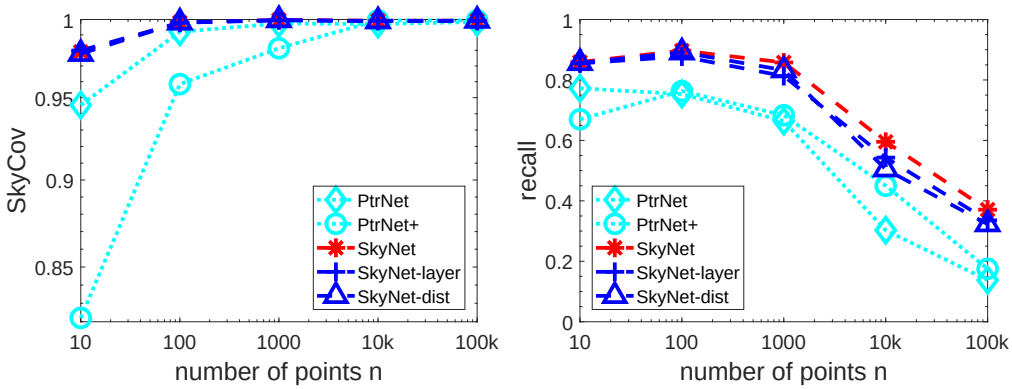


(c) Precision

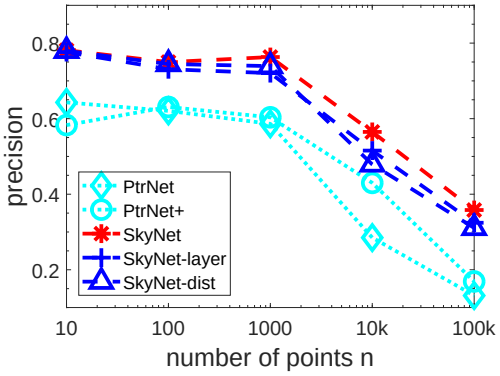
Figure 5.5: Model transformabilities of different n.

Figures 5.5(a)(b)(c) show that our proposed models have good transformabilities. We would like to know if we can enhance the transformability by

training the models with datasets with various number of points. We randomly generate 10000 datasets that contain various number of points from 10 to 1000 in two dimensional space. We train all the models by the 10000 datasets and use those trained models to predict the skyline points of the datasets with 10, 100, 1000, 10000, and 100000, respectively.



(a) SkyCov (b) Recall



(c) Precision

Figure 5.6: Transformability by training the models with datasets that contain various number of points.

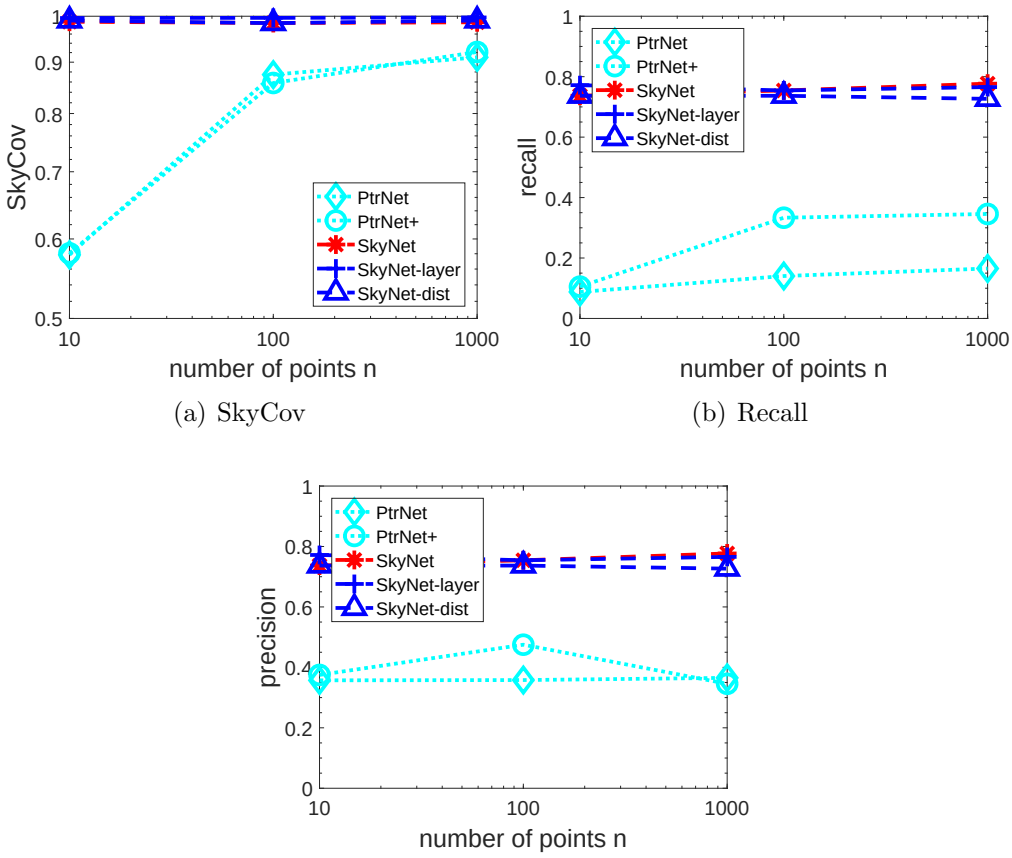
Figures 5.6(a)(b)(c) present the SkyCov, the Recall, and the Precision on the datasets of 10, 100, 1000, 10000, 100000 points predicted by the models that are trained by the datasets with various number of points from 10 to 1000. For the ease of presentation, we use $model_a$ to denote the model that is trained

by the datasets with a points and use $model_{a-b}$ to denote the model that is trained by the datasets with various number of points from a to b. We show the transformabilities of predicting the skyline points of the datasets with the length that the models already learned by comparing to Figure 5.2. To predict the skyline points of the datasets with size 10, 100, and 1000, our $models_{10-1000}$ have the similar performance with $models_{10}$, $models_{100}$, and $models_{1000}$, respectively. We show the transformabilities of predicting the skyline points of the datasets with the length that the models did not learn by comparing to Figure 5.5. To predict the skyline points of the datasets with size 100000, our $models_{10-1000}$ achieve around 0.4 for both the recall and the precision, which are higher than the score 0.3 achieved by $models_{1000}$. Therefore, we can see that the models trained by the datasets with various number of points have higher transformability than the models trained by the datasets with the fixed number of points.

5.6 Real Dataset

In this subsection, we evaluate the prediction accuracy of our proposed algorithms on the real NBA dataset. Because there are 57 skyline points in the real NBA dataset, we choose the top 57 points with highest weights in our neural skyline networks.

Figures 5.7(a)(b)(c) present the SkyCov, the Recall, and the Precision on the real NBA dataset predicted by the models that are trained by the datasets with 10, 100, and 1000 points, respectively. For the SkyCov metric, both SkyNet and its variants significantly higher than the basic SkyCov 0.76, which verifies the efficiency of our proposed neural skyline networks. Interestingly, the



(a) SkyCov (b) Recall

(c) Precision

Figure 5.7: Model transformabilities on real data.

SkyCov predicted by both PtrNet and PtrNet+ is lower than the basic SkyCov for 1000 points in five dimensional space, 0.76. The reason is that both PtrNet and PtrNet+ predicted much less skyline points than 158 (the average number of skyline points for 1000 points in five dimensional space) and less points usually lead to a lower SkyCov. For the recall and the precision scores, they almost do not change with the increasing number of points trained by the proposed models. The reason is that even for the models trained by 10 points, they are already good to learn the spatial knowledge for 2384 points. Comparing different models, both SkyNet and its variants have around 0.8 recall and precision, and they significantly outperform both PtrNet and PtrNet+, which

verifies the gain of our proposed neural skyline networks. The recall and the precision for both SkyNet and SkyNet+ are exactly the same. The reason is that we choose the top 57 points with highest weights, the denominators for both the recall and the precision are the same.

5.7 Summary

All our proposed models can achieve high accuracy. In practice, we can return twice the average number of skyline points to achieve higher recall. All our proposed models have very high transformabilities. For example, the models trained by the datasets with 1000 points can predict the skyline points for a dataset of 100000 points with a high accuracy. The models trained by the datasets with various number of points have higher transformability than the models trained by the datasets with fixed number of points. Therefore, in practice, we can train the model by the datasets with various number of points to achieve higher accuracy.

Chapter 6

Conclusion and Future Work

In this paper, we propose the first neural skyline networks, SkyNet, to predict the skyline points in linear time given any datasets. Instead of computing the skyline points using traditional algorithms, SkyNet predicts the skyline points based on the trained models. To better capture the spatial information of a set of points and help models to achieve higher accuracy, we propose two loss function variants, layer based loss function and Euclidean distance based loss function. Experimental results show that our proposed models are capable of high accuracy and high efficiency. Our proposed models are at least one order of magnitude faster than the traditional state-of-the-art skyline algorithms. Furthermore, our models' efficiency is insensitive to the number of dimensions, whereas the time complexity of the traditional algorithms increases exponentially as the number of dimensions increases. The main contribution of this paper is to outline and evaluate the potential of a novel approach to compute computational geometry structures (e.g., skyline and convex hull), which complements existing works and arguably opens up an entirely new research direction for a decades-old field. SkyNet serves as a complement for the skyline

community rather than a complete replacement of the traditional algorithms. One potential issue of our models is that during the encoding phase, the models use an RNN and treat input points as a sequence, but the input should be a set of points for the skyline problem. In the future, we will investigate into this issue and test whether different orders of the same set of points can affect the performance of our models. We want to modify our proposed network structures to achieve insensitivity to the order of input points, which requires the encoder to generate same representational vector v regardless of the order of inputs. In the future, we will also explore the applications of neural networks on solving other computational geometry problems.

Bibliography

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[3] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[4] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.

[5] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[7] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Pro-*

ceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014, pages 103–111, 2014.

[8] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734, 2014.

[9] B. Cui, L. Chen, L. Xu, H. Lu, G. Song, and Q. Xu. Efficient skyline computation in structured peer-to-peer systems. *IEEE Trans. Knowl. Data Eng.*, 21(7):1059–1072, 2009.

[10] S. Kapoor and P. V. Ramanan. Lower bounds for maximal and convex layers problems. *Algorithmica*, 4(4):447–459, 1989.

[11] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Symposium on Computational Geometry*, pages 89–96, 1985.

[12] J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang. Finding pareto optimal groups: Group-based skyline. *PVLDB*, 8(13):2086–2097, 2015.

[13] J. Liu, L. Xiong, and X. Xu. Faster output-sensitive skyline computation algorithm. *Inf. Process. Lett.*, 114(12):710–713, 2014.

[14] J. Liu, J. Yang, L. Xiong, and J. Pei. Secure skyline queries on cloud platform. In *ICDE*, pages 633–644, 2017.

[15] J. Liu, J. Yang, L. Xiong, J. Pei, and J. Luo. Skyline diagram: Finding the voronoi counterpart fro skyline queries. In *ICDE*, 2018.

[16] J. Liu, H. Zhang, L. Xiong, H. Li, and J. Luo. Finding probabilistic k-skyline sets on uncertain data. In *CIKM*, pages 1511–1520, 2015.

[17] H. Lu, Y. Zhou, and J. Haustad. Continuous skyline monitoring over distributed data streams. In *SSDBM*, pages 565–583, 2010.

[18] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[19] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

[20] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700, 2015.

[21] W. Yu, Z. Qin, J. Liu, L. Xiong, X. Chen, and H. Zhang. Fast algorithms for pareto optimal group-based skyline. In *CIKM*, pages 417–426, 2017.

[22] W. Zhang, A. Li, M. A. Cheema, Y. Zhang, and L. Chang. Probabilistic n-of-n skyline computation over uncertain data streams. *World Wide Web*, 18(5):1331–1350, 2015.