Butterfly Counting on Uncertain Bipartite Graphs

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

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

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

ABSTRACT

When considering uncertain bipartite networks, the number of instances of the popular graphlet structure the butterfly may be used as an important metric to quickly gauge information about the network. This Uncertain Butterfly Count has practical usages in a variety of areas such as biomedical/biological fields, E-Commerce and road networks. In this paper we formally define the uncertain butterfly structure (in which the existential probability of the butterfly is greater than or equal to some user-defined threshold t) as well as the Uncertain Butterfly Counting Problem (to determine the number of unique instances of this structure on any uncertain bipartite network). We then examine exact solutions by proposing a non-trivial baseline (*UBFC*) as well as an improved solution (*IUBFC*) which reduces the time complexity and employs heuristics to further reduce the runtime in practice. In addition to exact solutions, we propose two approximate solutions via sampling, *UBS* and *PES*, which can be used to quickly estimate the Uncertain Butterfly Count, a powerful tool when the exact count is unnecessary. Using a range of networks with different edge existential probability distributions, we validate the efficiency and effectiveness of our solutions.

PVLDB Reference Format:
Alexander Zhou, Yue Wang, and Lei Chen. Butterfly Counting on Uncertain Bipartite Graphs. PVLDB, 15(2): 211-223, 2022.
doi:10.14778/3489496.3489502

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/AlexanderTZhou/IUBFC.

1 INTRODUCTION

Uncertain (or Probabilistic) Networks are graphs aimed at modelling real-world networks in which connections between users or entities can only be assumed with differing levels of uncertainty [3, 26]. Recently, these networks have grown to become a highly interesting area of exploration due to the real-world applications they provide [31, 34, 37].

Extending on this concept, uncertain bipartite networks are used to examine uncertain networks in which the nodes are broken cleanly into two separate groups. Notable uncertain bipartite network use cases include uncertain links in biological/biomedical networks [7, 18, 27] or in user-item networks where edges may represent confidence in a recommendation [24] or a trust-prediction based on user ratings [11]. Even delivery problems [25, 35] can be modelled as uncertain bipartite networks based on the likelihood the courier can reach a target within a certain time frame.

On deterministic (or certain) bipartite networks the butterfly (sometimes referred to as a 'rectangle' in the literature), a fully connected subgraph containing exactly two nodes from each partition, is perhaps the most important graphlet structure [22, 28, 29]. Butterflies may be used to indicate closely connected nodes, which are useful in tasks such as community search [22] as well as forming the foundation of key community structures such as the bitruss [30, 40] or biclique [36]. In that respect, butterflies play a very similar role to that of the triangle on a unipartite graph [2, 5, 32].

Whilst butterfly counting has been studied on deterministic bipartite networks [22, 28, 29], they have yet to be extended to uncertain bipartite networks which limits their current applicability. This is unlike the triangle (and triangle counting problem) which has already been extended to uncertain scenarios due to similar reasoning [9, 41]. With the structure holding such importance on bipartite graphs, being able to determine the number of butterflies in uncertain bipartite networks can lead to significant insights into the inherent structure and properties of the network.

Use Case Scenario [Host-Parasite Network]: Host-Parasite networks track which parasites latch onto which hosts in a natural environment. In some work, such uncertain networks have been built which also consider host-parasite combinations which could (but not have provably been shown to) exist in nature [27]. By counting the butterflies on such networks, the potential likelihood and impact of cross-host parasite transaction could be examined before introducing such hosts to the same environment. As a result, action should be taken to minimise the uncertain butterfly count on any such environment.

Use Case Scenario [Recommendation Network]: Another use case example is a recommendation user-item network, where edges can exist without certainty where the existential probability is the likelihood a user would enjoy or purchase an item. These recommendation networks can be formed using multiple well-established techniques such as Collaborative Filtering [24]. The uncertain butterflies and their respective count could be used to recommend items which would result in the emergence or reinforcement of strong communities (i.e. bitruss-based [40]). Additionally, general statistics such as a bipartite clustering coefficient [13] can be gleamed using the uncertain butterfly count. Furthermore, by comparing the resulting counts of two different recommendation strategies, a relatively cheap method of comparing potential impacts of strategies beyond simply an increased number of edges becomes viable. That is, the method resulting in the higher count is more promising for future network health.

Proceedings of the VLDB Endowment, Vol. 15, No. 2 ISSN 2150-8097.
doi:10.14778/3489496.3489502

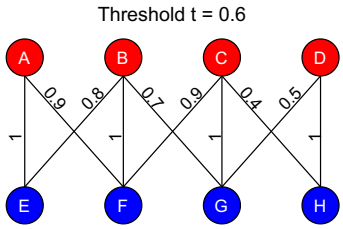

Figure 1: An example uncertain bipartite network with a red and blue partition. Each edge has a corresponding existential probability, with the threshold value $t = 0.6$.

1.1 Our Contributions

In this paper we formally define the Uncertain Butterfly Counting Problem, in which we wish to determine the number of butterflies that exist on an uncertain bipartite network with existential probability greater than or equal to a threshold value t. Figure 1 is an example uncertain bipartite network, where the threshold is set to be $t = 0.6$. The butterflies $B(A, B, E, F)$ and $B(B, C, F, G)$ both have existential probabilities (the product of the existential probability of their four edges) greater than t (0.72 and 0.63 respectively) whilst the butterfly $B(C, D, G, H)$ does not (with probability 0.2). Therefore, the uncertain butterfly count on this network is 2.

Existing butterfly counting techniques on deterministic graphs cannot realistically be transferred directly to this new setting as they treat each edge (and wedge) with the same importance, something that is evidently not true in the uncertain setting. Additionally, in these methods it is not possible to add in a trivial step in which each butterfly is evaluated as either satisfying or failing the threshold as the deterministic method aggregates a batch of butterflies at the same time. It is possible to enumerate all possible worlds, count the number of certain butterflies in each world and aggregate the result but such an approach is extremely unrealistic in practice.

In this paper we propose two exact solutions to the Uncertain Butterfly Counting Problem. Firstly, we create a baseline solution *UBFC* which adapts the current best solution for the Butterfly Counting problem on deterministic graphs and modifies it as necessary such that it is satisfactory to our problem. Then, we propose an improved solution *IUBFC* which reduces the time complexity (from $O(\sum_{e(u,v)\in E} \min\{deg(u)^2, deg(v)^2\})$ to $O(\sum_{e(u,v)\in E} \min\{deg(u) \log(deg(u)), deg(v) \log(deg(v))\})$ without any change in the memory complexity by improving the manner in which wedges are found and combined to find uncertain butterflies. We additionally add heuristics, such as early edge discarding, to further improve the performance of the algorithm.

We also propose two estimation via sampling solutions which trade accuracy in return for significantly improved running times. Minor sacrifices in accuracy in return for major reductions in cost can be highly attractive when a quick examination is needed for recommendation strategies as an example.

Our first algorithm, Uncertain Butterfly Sampling (*UBS*), samples vertices/edges without replacement determining the number of uncertain butterflies each vertex/edge is a part of. By extrapolating and averaging the solution we can derive an estimation of the count with high confidence as the sample size increases. Our second algorithm, Proportion Estimation Sampling (*PES*), estimates the proportion of butterflies with existential probability $\geq t$ and then uses a faster deterministic sampling technique to quickly estimate the number of uncertain butterflies.

Finally we demonstrate the effectiveness of our exact solutions and the efficiency/effectiveness trade-off shown by our sampling approaches by performing experiments on multiple uncertain bipartite networks.

Our contributions are summarised as following:

- We formally define the Uncertain Butterfly and Uncertain Butterfly Counting Problem
- We propose a non-trivial baseline and introduce an improved algorithm which reduces the time complexity of the solution at no increased memory costs
- We propose two fast alternate approximation strategies using sampling
- We validate our algorithms via extensive experiments

To outline the remaining paper, we first examine related works to this topic (Section 2) before formally defining the notation and the problem itself (Section 3). We then introduce a non-trivial baseline exact solution (Section 4) and follow up with key improvements which reduce the time complexity of the solution in combination with powerful heuristics (Section 5). We also propose two sampling solutions which provide approximate results (Section 6). Following this we examine the efficiency and effectiveness of our algorithms (Section 7) before concluding the paper (Section 8).

2 RELATED WORKS

Butterflies: The butterfly (sometimes referred to as a rectangle or 4-cycle) [22, 28, 29] is perhaps the most fundamental structure that exists on bipartite networks [39]. Similar to the role that triangles play on unipartite graphs [2, 5, 32], butterflies are utilised in many areas to either quickly gauge information on the network [22] or help to find more complex structures such as the biclique [15, 19], k/k^*-Partite Clique [20, 38] and bitruss [30, 40].

There has been a recent boom in the area of butterfly counting. In particular, a baseline deterministic solution for counting the number of butterflies on a regular bipartite network [28] with multiple improvements to that algorithm [22] resulting in the current best solution being a vertex priority approach [29] has been studied.

In addition, the butterfly counting problem has been tackled in different settings with parallel [28] and cache-aware solutions [29] considered and estimation solutions [22] proposed. Furthermore, butterfly counting in a streaming setting has also been a recent area of research [23]. To the best of our knowledge, we are the first to consider the butterfly (and subsequently the butterfly counting problem) on an uncertain setting and thus we require different solutions to deal with the increased requirements presented by the new problem.

Bipartite Networks: Bipartite networks in particular have recently seen a surge in research popularity due to the real-world applicability of the network type in modelling information [7, 11, 18, 24, 27]. The problems being studied that closely relate to our work is those that regard in the counting, enumeration or search of a particular subgraph structure in the network.

Table 1: Key Notation and Definitions

Notation	Definition
$G = (V = L \cup R, E, P)$	Uncertain bipartite graph
$Pr(.)$	Existential probability
$\mathbb{W}$	Set of all possible worlds
W_i	Possible world i
$W_{backbone}, W_1$	Backbone world/graph
t	Threshold value
B	Deterministic butterfly
B_t	Uncertain butterfly
$\angle$	Deterministic wedge
$\angle_t$	Uncertain wedge
C	Deterministic butterfly count
C_t	Uncertain butterfly count
$p(u)$	Vertex priority of node u
α	C_t/C
$\hat{\alpha}$	An estimated value of α

Such structures include the well-studied biclique [15, 19, 36] in which each node in one partition of the structure must be connected to every node from the other partition. The butterfly is effectively a (2 x 2) biclique (that is a biclique with two nodes in each partition). Another structure is the (α, β)-core [14] which models the popular k-core on the bipartite setting. Perhaps the most important structure to us is the bitruss, a structure in which each edge in the bitruss is a part of k butterflies in the subgraph [30, 40]. Due to the requirement of being able to count butterflies, advancements in butterfly counting can often directly result in improvements to bitruss algorithms.

Uncertain Networks: Uncertain (probabilistic) graphs as a subgenre have received attention from various problems, often by taking traditional graph problems and transferring them into the setting (e.g. k-Nearest Neighbour [21], Core Decomposition [6], Shortest Path [33]). On uncertain graphs, a popular choice is to use Possible World Semantics [1], which breaks the system down into separate instances of the graph with non-zero existential probability. As many problems theoretically require the analysis of all possible worlds, which is expensive, research has been conducted to narrow the scope by finding a good representative possible world [17] or by sampling a set of possible worlds [8, 10, 12] to estimate the result. There does exist one notable work on motif counting (including butterflies) on uncertain (unipartite) networks (LINC) [16] however their problem formulation differs from ours (they wish to output the probability mass function of counts across all possible worlds) as ours finds all butterfly instances whose existential probability is above a specific threshold.

3 PROBLEM DEFINITION

We give the formal definitions of our problem as well as the key notation used in our paper. The key notation used in this paper can be found in Table 1.

3.1 Uncertain Bipartite Networks

Definition 3.1. Uncertain Bipartite Network: An uncertain bipartite network $G = (V = L \cup R, E, P)$ is a network in which any node $u \in L$ may only be connected to node v given that $v \in R$ or vice versa. $P : E \rightarrow (0, 1]$ maps an edge $e_{u,v} \in E$ to a non-zero existential probability $Pr(e_{u,v}) \in (0, 1]$.

On a given uncertain bipartite network G, a possible world $W_i = (V, E_{W_i})$ is a single possible network outcome after fairly and randomly determining if each edge $e \in E$ exists in W_i based on $Pr(e)$ [1]. Like existing work on uncertain networks [1, 9, 16, 41], we assume the existential probability of each edge is independent of each other. Each possible world W_i on G exists with existential probability:

$$Pr(W_i) = \prod_{e \in E_{W_i}} Pr(e) \cdot \prod_{e \in E \setminus E_{W_i}} (1 - Pr(e)) \quad (1)$$

For any G, there exists $2^{|E|}$ possible worlds. The set of all possible worlds is defined as $\mathbb{W} = \{W_1, \ldots, W_{2^{|E|}}\}$ and via the Law of Total Probability $Pr(\mathbb{W}) = \sum_{i=1}^{n} Pr(W_i) = 1$. Additionally, we call the possible world in which all edges are selected to exist the backbone graph of G, denoted by $W_{backbone}$. This may also be thought of as the deterministic variant of the network. Furthermore it can be trivially proven that the total sum of existential probabilities for all worlds a single edge e exists in is equal to $Pr(e)$. That is, given a randomly sampled world W_i the probability $e \in E(W_i)$ is equal to $Pr(e)$.

3.2 Butterfly Counting

In certain bipartite networks, butterflies are considered as one of the key fundamental building block graphlet structures due to the cohesive relationships they represent [22, 28, 29]. On certain graphs, we may define the butterfly as following:

Definition 3.2. Butterfly: A butterfly B consisting of nodes $u_1, u_2 \in L$ and $v_1, v_2 \in R$ exists if and only if there exists edges from u_1 to v_1 and v_2 as well from u_2 to v_1 and v_2. We may use the notation $B(u_1, u_2, v_1, v_2)$ to denote a butterfly containing nodes u_1, u_2, v_1, v_2.

We extend the idea of the butterfly to the uncertain bipartite network setting.

Definition 3.3. Uncertain Butterfly: Given a randomly sampled possible world $W_i \in \mathbb{W}$ and a probability threshold $t (0 \leq t \leq 1)$, an Uncertain Butterfly B_t is a butterfly whose existential probability is greater than or equal to t (i.e. $Pr(B_t) \geq 1$).

The existential probability of a butterfly can be calculated by $Pr(B_t) = \prod_{e \in E_{B_t}} Pr(e)$.

Definition 3.4. Butterfly Count and Uncertain Butterfly Count: The (deterministic) butterfly count C on a certain bipartite graph is the number of unique butterflies on the graph. Similarly, the uncertain butterfly count C_t on an uncertain bipartite graph is the number of unique uncertain butterflies on that network.

Figure 2 illustrates three uncertain butterflies and their respective existential probabilities. For $t = 0.4$, $B1$ and $B2$ would be counted towards the uncertain butterfly count whilst $B3$ would not resulting in a final uncertain butterfly count of 2.

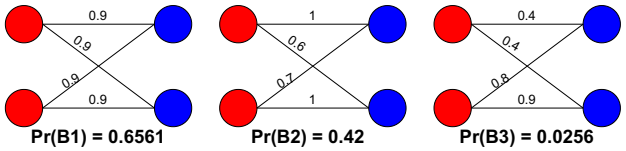

Figure 2: Three uncertain butterflies and their corresponding existential probabilities

Additionally, we need to introduce the idea of the Wedge and Uncertain Wedge.

Definition 3.5. Wedge and Uncertain Wedge: A Wedge $\angle(u, v, w)$ is a two-hop path consisting of two different endpoint vertices u, w from the same partition of the uncertain bipartite network and a middle vertex v from the other partition. An Uncertain Wedge $\angle_t$ is a Wedge that, on a randomly sampled world W_i, exists with $Pr(\angle_t) \geq t \in [0, 1]$.

Notably, any butterfly consists of four wedges which means butterfly counting techniques often use wedges as one of the building blocks in finding butterflies. If a butterfly B contains a wedge $\angle$, we use the notation $\angle \in B$ (and the same for $\angle_t \in B_t$).

We note that there can be an alternate definition of the uncertain butterfly in which each edge in the butterfly must hold an existential probability greater than some threshold. However, in this scenario we may simply prune all edges which fail to meet this threshold and then perform any certain/deterministic butterfly counting technique. Additionally, there could be an alternate problem formulation, in a similar vein to [16], in which instead of a threshold value, the mean and variance of the butterfly count over all possible worlds are derived. Due to the removal of the threshold, the problem would become #P-Hard and as a consequence this hampers the usability of any exact algorithm in a practical setting.

3.3 Problem Statement

We formally define the Uncertain Butterfly Counting Problem as following:

Definition 3.6. Uncertain Butterfly Counting Problem: For an undirected, unweighted bipartite graph $G = (V = L \cup R, E, P)$ and a threshold value t, the Uncertain Butterfly Counting Problem is to determine the count C_t of all uncertain butterflies B_t on G (where $Pr(B_t) \geq t$).

4 EXACT ALGORITHMS - BASELINE

In this section we propose a baseline for exact solutions to the Uncertain Butterfly Counting Problem

Given no baseline exists specifically for the Uncertain Butterfly Counting Problem, the trivial baseline would be to permute all combinations of four edges and check if it is an Uncertain Butterfly satisfying our threshold, taking $O(E^4)$ time. However, given prior work exists on the deterministic variation of our problem, we will instead utilise a baseline that finds butterflies on the backbone graph and examines if each individually satisfies the existential probability threshold environment.

Our baseline is a modified version of the Vertex Priority Butterfly Counting (*BFC-VP*) algorithm provided by Wang et al [29] for the deterministic problem, with the critical difference being how wedges are handled in order to find uncertain butterflies. We call our baseline *UBFC*. A core idea in this algorithm is that of vertex priority.

Definition 4.1. Vertex Priority [29]: The vertex priority of any node $u \in V$ in comparison to any other node $v \in V$ is defined as $p(u) > p(v)$ if:

(1) $deg(u) > deg(v)$
(2) $id(u) > id(v)$ if $deg(u) = deg(v)$

where $id(u)$ is the vertex ID of u.

Vertex priority is effectively a method that allows for a structured ranking of all nodes in G to avoid the same butterfly being counted multiple times. Additionally, as shown in [29], the usage of a vertex priority approach reduces the running cost of the algorithm.

Algorithm 1: *UBFC*

Input : G: Input Uncertain Bipartite Network
t: Uncertainty Threshold
Output: C_t: Uncertain Butterfly Count

```
1  W_1 ← Extract Backbone Graph;
2  Sort N(u) of each u ∈ V_{W_1} by vertex priority;
3  C_t ← 0;
4  foreach u ∈ V_{W_1} do
5      Create H(w) for each Node w in same partition as u;
6      foreach v ∈ N(u) : p(v) < p(u) do
7          foreach w ∈ N(v) : p(w) < p(u) do
8              H(w).append(v);
9      foreach Node w : |H(w)| > 1 do
10         foreach Nodes v_1, v_2 ∈ H(w), v_1 ≠ v_2 do
11             if Pr(B(u, w, v_1, v_2)) > t then
12                 C_t ← C_t + 1;
```

Algorithm 1 details the baseline solution. We use the backbone graph W_1 (Line 1) in order to extract vertices and edges to find butterflies before checking whether they satisfy the uncertainty threshold. For each node, we sort its neighbours by (increasing) vertex priority (Line 2).

Then, for each node $u \in V_{W_1}$ we perform the baseline uncertain counting algorithm. Firstly, we initialise a Hashmap H which will be used to store all wedges containing u. For each node w, which is a two-hop neighbour of u (thus forming at least one wedge with u), we initialise a list in H which will refer to as $H(w)$ (Line 5). Then, for each nodes $v \in N(u) : p(v) < p(u)$, we find its neighbours $N(v)$. For each node $w \in N(v) : p(w) < p(u)$, we add v to $H(w)$ since there exists a wedge $\angle(u, v, w)$ (Lines 6-8). The vertex priority constraints ensure no redundancy in our algorithm, that is no wedge will be ever added to any list twice.

One important difference between the existing deterministic algorithm and our uncertain baseline is that the deterministic method does not need to store previously discovered wedges as it can quickly determine the number of butterflies at this point via the operation $\binom{n}{2}$ where n is the number of wedges with the same start and end vertices, whereas in the uncertain version each

combination of wedges needs to be examined to check if the resulting butterfly satisfies the probability requirement. As a result the deterministic method does not maintain a list but instead a simple number representing the total number of wedges for each u, w combination. However, given that we need to consider existential probabilities and need to thus know the wedges we keep, our method instead maintains the list $H(w)$. For each combination of two nodes $v_1, v_2 \in H(w), v_1 \neq v_2$, we check if the butterfly $B(u, w, v_1, v_2)$ is an uncertain butterfly B_t. If it is, we increase C_t (Lines 9-12). This management and combining of wedges sets our baseline algorithm for the uncertain butterfly counting problem apart from the deterministic variant.

4.1 Algorithm Complexity

We now examine both the time and space complexity of the baseline algorithm.

Theorem 4.2. *The time complexity of UBFC is* $O(\sum_{e(u,v)\in E} \min\{deg(u)^2, deg(v)^2\})$.

Proof. Wang et. Al proved in their work that the deterministic algorithm runs in $O(\sum_{e(u,v)\in E} \min\{deg(u), deg(v)\})$ time [29]. Notably, their proof examines the cost of wedge discovery (Algorithm 1 Lines 5-10) which makes up the dominating part of their algorithms time complexity.

In the deterministic variant, the remaining cost of the algorithm is done in $O(1)$ time which is different from our uncertain version. In our uncertain version we further have to determine the existential probabilities of the subsequent butterflies given the wedges (Algorithm 1 Lines 9-12).

Due to the Vertex Priority approach, we know that the start vertex of any wedge we discover must have a degree of size greater than or equal to the degree of the mid and end vertex. As a result, the total number of wedges that can be uncovered by our algorithm containing a start vertex u and an end vertex w is $deg(w)$, which means the total combination of two wedges containing u and w is $deg(w)^2$. As a result, the cost of determining all wedge combinations of node u and w is $O(deg(w)^2)$. Thus, the time complexity of $UBFC$ is $O(\sum_{e(u,v)\in E} \min\{deg(u)^2, deg(v)^2\}$. □

Theorem 4.3. *The memory complexity of UBFC is* $O(\sum_{e(u,v)\in E} \min\{deg(u), deg(v)\}$.

Proof. As noted in the proof for Theorem 4.2, the maximum number of wedges discovered for a start vertex u is $deg(u)$. □

5 EXACT ALGORITHMS - IMPROVEMENTS

In this section, we examine methods of improving the exact solution, ultimately reducing the running time as well as proposing heuristic techniques to speed up the algorithm in practice.

5.1 Early Edge/Wedge Discarding

One notable area of wasted operations is when examining or considering wedges that are unlikely to, or simply cannot, be a part of any uncertain butterfly for a given t.

Lemma 1. *If there exists an edge e with existential probability* $Pr(e) < t$, *e cannot be a part of any uncertain butterfly.*

Proof. Given that all probabilities must be ≤ 1, this can be proven trivially. □

Lemma 1 means that we can simply ignore any edge that exists with probability $< t$. Additionally, this same logic can be applied to any graphlet structure (specifically a wedge) with existential probability $< t$ that can be found within a butterfly. Therefore, a simple pruning heuristic we adopt is that if at any point an edge or wedge has an existential probability lower than the threshold, we ignore it for future considerations.

In the case where no edge $e \in E$ exists with $Pr(e) = 1$, we may adjust the pruning threshold to be higher ($t/\max_{e\in E}\{Pr(e)\}$) to account for the higher required existential probabilities to compensate.

5.2 Improved List Management

Another area of improvement over the baseline is in how the list l containing found wedges is handled, noting that currently finding all uncertain butterflies from l is done in $O(|l|^2)$ time, which in turn is a major increase to the runtime of the baseline algorithm acting as one bottleneck.

Lemma 2. *To determine if an uncertain butterfly exists, containing nodes* (u_1, u_2, v_1, v_2), *the only information we need is the existential probability of any two unique wedges sharing the same end points (either* u_1 *and* u_2 *or* v_1 *and* v_2*) and* t.

Proof. Firstly recall that two wedges $\angle_1, \angle_2$ with the same endpoints that are found in the same butterfly B contain all the edges in the butterfly, two from each wedge totalling four unique edges. This is a fundamental truth which serves as the foundation of nearly all butterfly counting techniques [22, 28].

The same extends to the existential probabilities of wedges and butterflies. That is, $Pr(B) = Pr(\angle_1) * Pr(\angle_2)$. With this information and t, we can easily check if an uncertain butterfly exists from these nodes. Given that we are not enumerating, we don't actually need to store any nodes given that the shared end points are assured and wedge probabilities are stored instead. □

Lemma 2 allows us to modify the content of our lists and our wedge combination algorithm. Firstly, instead of l storing the middle node of the wedge we instead have it store the existential probability of the wedge. Secondly, we keep the list sorted upon the insertion of a new wedges information.

Before we fully explain the new Uncertain Butterfly Counting algorithm for the sorted list l, we need to note some properties of the list.

Lemma 3. *If two wedges* $\angle_1, \angle_2 : Pr(\angle_1) > Pr(\angle_2)$ *form an Uncertain Butterfly, then all wedges before* $\angle_1$ *in* l *forms an Uncertain Butterfly with all wedges before* $\angle_2$ *in* l.

Lemma 4. *If two wedges* $\angle_1, \angle_2 : Pr(\angle_1) > Pr(\angle_2)$ *do not form an Uncertain Butterfly, then all wedges after* $\angle_1$ *in* l *do not form an Uncertain Butterfly with all wedges after* $\angle_2$ *in* l.

Proof. As l is a sorted list by existential probability, both Lemma 3 and 4 can trivially be shown with basic probability theory. □

Both Lemma 3 and 4 allow us to quickly validate the existence of Uncertain Butterflies using l without having to actually calculate their existential probabilities. These ideas form the foundation of our improved algorithm for extracting an Uncertain Butterfly count from l.

Algorithm 2: ImprovedListCount

Input : l: Sorted List of Existential Probabilities
t: Uncertainty Threshold
Output: C_t: Uncertain Butterfly Count

1 $C_t \leftarrow 0, i \leftarrow 0, j \leftarrow 1, F \leftarrow false$;
2 **while** $j < |l|$ **do**
3 **if** $l[i] * l[j] < t$ **then**
4 $F \leftarrow true$;
5 $target \leftarrow t/l[j]$;
6 $i \leftarrow GTBinarySearch(l, 0, i-1, target)$;
7 **if** $i < 0$ **then**
8 $Break$;
9 $C_t \leftarrow C_t + i + 1$;
10 $j \leftarrow j + 1$;
11 **if** $F = false$ **then**
12 $i \leftarrow i + 1$;

Algorithm 2 details our improved technique for determining the number of Uncertain Butterflies from the wedges in l. Given that l is a sorted list of existential probabilities, we initialise $i = 0$ and $j = 1$ to be two indexes on the list. The algorithm continues until $j == |l|$ or another break condition is reached.

If $l[i] * l[j] \geq t$ this implies that the current wedges being represented form an uncertain butterfly, which means determining values for i and j are the key in reducing the runtime of the algorithm. As the algorithm begins, as long as $l[i] * l[j] \geq t$ we increment the count by $i + 1$ (due to Lemma 3) and then increment by i and j by 1 (Lines 9-12).

However, after the first time $l[i] * l[j] < t$, we no longer need to continue to increment i due to Lemma 4. This is reflected in our algorithm by a boolean flag F (Lines 1, 4, 11-12). When i and j fail to form an uncertain butterfly, we need to find the value of i which can form an uncertain butterfly algorithm with j done by finding a minimum uncertain probability value $target = t/l[j]$ (Line 5). We then give i the largest index (smallest existential probability) in which $l[i] > target$, done via a variation of Binary Search called $GTBinarySearch(list, leftIndex, rightIndex, target)$ on the existential probabilities in index 0 to $i - 1$ (Line 6). If no such value of i exists, we terminate our algorithm as no further wedges can be found (Line 7-8).

Theorem 5.1. *Algorithm 2 produces the correct uncertain butterfly count.*

Proof. Our algorithm can be thought of as iterating through all values of $0 < j < |l|$ and finding the largest value of $i < j$ which forms an uncertain butterfly. Then, using Lemma 3, we know that there exist $i + 1$ uncertain butterflies in l which satisfy the $i < j$ condition for j. This is done for all j which will result in the exact uncertain butterfly count with no redundancy. Furthermore, whenever $l[i] * l[j] < t$, we know that no future value of i can be greater than the current value of i as j increases due to Lemma 4. □

Theorem 5.2. *Algorithm 2 takes* $O((|l|) + \log(|l|!/\lceil \frac{|l|}{2} \rceil!))$ *time.*

Proof. We iterate j from 1 to $|l| - 1$ assuming no early termination which accounts for the $O(|l|)$ portion of the time complexity. If we were to perform $GTBinarySearch$ $|l| - 1$ times on the entire list l then our time complexity would be $O((|l| - 1)\log(|l|))$. However, since the segment of $|l|$ in which $GTBinarySearch$ ($GTBS$) must operate is capped the first time the function is called and future calls search on an even smaller space due to Lemma 4.

If $GTBS$ is called when $j = |l| - 1$, then the total cost of all $GTBS$ calls is $O(\log(|l| - 1))$. If $GTBS$ is called when $j = |l| - 2$, the total possible cost of all $GTBS$ must be $O(\log(|l| - 2) + \log(|l| - 3))$. The most number of $GTBS$ calls is capped at when $j = \lceil \frac{|l|}{2} \rceil$ with a total possible cost of $O(\log(\lceil \frac{|l|}{2} \rceil) + \log(\lceil \frac{|l|}{2} \rceil - 1) + \cdots + \log(1)) = O(\log(\lceil \frac{|l|}{2} \rceil!))$. The total possible $GTBS$ cost of any smaller value of j must be smaller than this value.

Given this, the total possible cost of $GTBS$ for list l is $O(\max_{\{j=\lceil \frac{|l|}{2} \rceil, \ldots, |l|-1\}} \left\{\log(((\prod_{k=1}^{|l|-j}(j-k))!)\right\}) <$ $O(\log(|l|!/\lceil \frac{|l|}{2} \rceil!))$ time. □

Despite Theorem 5.2 appearing expensive due to the factorial, it is still distinctly smaller than $O((|l| - 1)\log(|l|) = O((|l| - 1) + \log(|l|^{(|l|-1)}))$ time which in turn is obviously smaller than the baseline of $O(|l|^2)$. In most realistic scenarios with the early termination clause and the size of the $GTBS$ search space potentially reducing by much more than a single item, the realistic cost of Algorithm 2 can be significantly smaller than the worst case that is Theorem 5.2. All operations in Algorithm 2 can be done in $O(|l|)$ space, which can be verified trivially.

5.3 Improved Algorithm

Adopting all mentioned improvements, we now present our Improved Uncertain Butterfly Counting Algorithm ($IUBFC$).

Algorithm 3: $IUBFC$

Input : G: Input Uncertain Bipartite Network
t: Uncertainty Threshold
Output: C_t: Uncertain Butterfly Count

1 $W_1 \leftarrow$ Extract Backbone Graph;
2 $RemoveUnusableEdges(W_1, t)$;
3 Sort $N(u)$ of each $u \in V_{W_1}$ by vertex priority;
4 $C_t \leftarrow 0$;
5 **foreach** $u \in V_{W_1}$ **do**
6 Create $H(w)$ for each Node w in same partition as u;
7 **foreach** $v \in N(u) : p(v) < p(u)$ **do**
8 **foreach** $w \in N(v) : p(w) < p(u)$ **do**
9 **if** $Pr(\angle(u, v, w)) \geq t$ **then**
10 $H(w).sortedInsert(Pr(\angle(u, v, w)))$;
11 **foreach** $w : |H(w)| > 1$ **do**
12 $C_t \leftarrow C_t + ImprovedListCount(H(w), t)$;

Algorithm 3 details our improved algorithm, adopting the previously mentioned improvements. The first deviation from the baseline solution is an additional step after extracting the backbone graph where all edges with $Pr(e) < t$ are removed from W_1 for the purposes of wedge (and butterfly) discovery as per Lemma 1 (Line 2).

We sort by vertex priority (Line 3) and then continue using a similar wedge discovery approach as the baseline using the new priority (Lines 5-8). One notable difference from the baseline is the wedge lists store the wedge existential probability value instead of the actual nodes. When a wedge $\angle(u, v, w)$ is discovered it is inserted (if $Pr(\angle(u, v, w)) \geq t$ (Lemma 1)) such that the list at $H(w)$ remains sorted (Lines 9-10).

Finally, our alternate list management strategy allows us to utilise the improved list count strategy detailed in Algorithm 2 (Lines 11-12).

We now examine the changes to space and time complexity of the algorithm compared to our baseline.

Theorem 5.3. *The time complexity of IUBFC is* $O(\sum_{e(u,v)\in E} \min\{deg(u)\log(deg(u)), deg(v)\log(deg(v))\})$.

Proof. Once all wedge existential probabilities in l have been inserted, the cost of having kept it sorted is $O(|l|\log(|l|))$. As any wedge list for start vertex u and end vertex v is capped at size $deg(v) \leq deg(u)$, as shown in the proof for Theorem 4.2, the cost of keeping a wedge list sorted for u and v is $O(deg(v)\log(deg(v))$.

As shown in Theorem 5.2, the time complexity of our new wedge list algorithm is less than $O(deg(v)\log(deg(v))$ time for u and v. □

The memory complexity of *IUBFC* is unchanged from *UBFC* (Theorem 4.3).

5.4 Sorting by Vertex Priority vs. Existential Probability

Vertex Priority is an important tool in the deterministic variation of the problem as it ensures, in total, each edge is only examined one time for each wedge it is a part of. This not only minimises the cost of the algorithm but also reduces redundancy. However, this idea was formed under the assumption that all butterflies (and wedges) are required in the final solution which is evidently not true in our uncertain variant.

Currently, in our improved solution, nodes are sorted by increasing vertex priority (Algorithm 3 Line 3) and a check is made for each wedge to determine if it satisfies the existential probability constraints (Algorithm 3 Lines 7-10). Of course, since the nodes are sorted by priority, we can terminate the iteration through the neighbour list after the first node which fails the priority check in Lines 7 and 8, which serves as an important part of both the baseline and our improved solution in terms of suppressing the cost in this step.

However, we may consider an approach which comes at this problem from an alternate viewpoint. Instead of sorting by (increasing) vertex priority we instead sort by (decreasing) edge existential probability. We can leverage the fact if edges e_1 and e_2 do not form a wedge with existential probability $\geq t$, no edges with existential probability smaller than $Pr(e_2)$ forms a satisfactory wedge with e_1 and vice versa using the same logic as Lemma 4. The same is true for wedge combinations with higher existential probability (following Lemma 3). We will refer to the method in which nodes are sorted by vertex priority as *VP* and the method in which nodes are sorted by edge probability as *EP*.

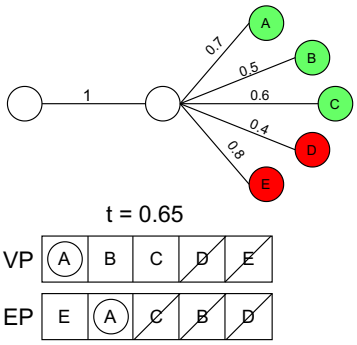

Figure 3: A graph and the order in which the Vertex Priority (*VP*) approach and Edge Probability (*EP*) approach checks nodes. A green node is one in which the vertex priority constrain is satisfied and a red node is one in which it is not. A circle around the node in the order list indicates that the node also satisfied the alternate requirement constraint.

Algorithm 4: *SortByEdgeProb*

```
1 foreach u ∈ V_{W_1} do
2     foreach v ∈ N(u) do
3         foreach w ∈ N(v) do
4             if Pr(∠(u, v, w)) ≥ t then
5                 if p(v) < p(u) && p(w) < p(u) then
6                     H(w).sortedInsert(Pr(∠(u, v, w)));
7             else
8                 break;
```

Algorithm 4 shows the modified method for wedge discovery assuming sorted *EP*. For an edge $e(u, v)$, we examine nodes $w \in N(v)$ and determine if $Pr(e(u, v)) * Pr(e(v, w)) \geq t$ (Line 4). If this probability constrain satisfied, we add the wedge if the priority constraint is also satisfied (Line 5-6). If the probability constraint is not satisfied, we know that there are no more satisfactory wedges containing $e(u, v)$ and we move on in our search (Line 7-8).

Figure 3 illustrates the order in which the two techniques accesses nodes from the given graph. In the graph, a green node indicates the vertex satisfies the priority constraint whilst a red node indicates that the vertex does not satisfy the priority constraint. In the case of *VP*, the nodes are sorted in alphabetical order.

Using *VP*, nodes A, B and C are checked in that order with only node A satisfying the existential probability check. Since node D fails the vertex priority requirement, the wedge discovery process is terminated. When using *EP* on the other hand, nodes E and A are checked in that order with only node A satisfying the vertex priority check. Since node C results in a wedge with lower existential probability than t, the wedge discovery process is terminated.

Both *VP* and *EP* ensures that each wedge is only added to a list once during the entire algorithm so the final result will remain the

same. What the methods do control is the number of times an edge reaches the alternate check (wedge probability for VP and vertex priority for EP) step. In VP, each edge reaches the alternate check step once for each wedge it is a part of. Alternatively, in EP each edge reaches the alternate check step twice for each wedge with existential probability $\geq t$ and zero times otherwise.

Theorem 5.4. *The time complexity of EP is* $O(\sum_{e(u,v)\in E} \min\{deg(u)\log(deg(u)), deg(v)\log(deg(v))\} + \Gamma(e))$ *where* $\Gamma(e)$ *is the number of wedges* $\angle$ *containing* e *where* $Pr(\angle) \geq t$.

Proof. The numbers of wedges that will be passed into a list as well as the size of each list is unchanged from Theorem 5.3. The only change is that for each edge $e \in E$ it will be checked an additional time for each uncertain wedge it is contained in. □

The memory complexity of EP is unchanged compared to VP.

Of course the better method to use between VP and EP is highly graph and threshold value dependant. The break-point in which EP is the superior method is when less than 50% of the wedges in W_1 post pruning have an existential probability $\geq t$. Furthermore, the larger the percentage favours in one direction indicates the corresponding method is more efficient by that margin. When the percentage skews towards 100% VP is significantly better and when it skews towards 0% EP is significantly better. Furthermore, it is possible for EP to become more expensive than even the baseline if there is minimal or no pruning and a large percentage of the wedges have an existential probability $\geq t$ due to the nature of reaching the check step twice as opposed to once for each wedges on G than an edge is apart of in the baseline. Thus, it is recommended for low t values relative to the edge probability distributions to select VP.

It should be noted that whilst both $UBFC$ and $IUBFC$ are both designed for the Uncertain Butterfly Counting problem, with minimal variations to the algorithm both methods can be easily adapted into enumeration algorithms at no increased memory cost, and in the case of $UBFC$ no added time complexity.

6 SAMPLING ALGORITHMS

Whilst exact algorithms are indeed important for determining the exact uncertain butterfly count on a network, approximate solutions may also provide significant insights at a fraction of the time cost. In this section, we introduce two local sampling techniques aimed at solving the Uncertain Butterfly Counting problem. The first technique Uncertain Butterfly Sampling (UBS) operates by local sampling the number of uncertain butterflies each vertex is a part of. The second technique Proportion Estimation Sampling (PES) uses a local sampling approach which determines the number of certain butterflies and estimates the proportion of butterflies on G which are uncertain.

6.1 Uncertain Butterfly Sampling (UBS)

For the first approximate solution we utilise a local sampling approach to determine the number of uncertain butterflies containing a sampled vertex or edge. We then propagate that number outwards accordingly to the entire network and as the number of samples grows the average value converges to the true result. This approach is adapted from a local sampling deterministic solution proposed by Sanei-Mehri [22] with the added task of needing to monitor and check existential probabilities to ensure each uncertain butterfly is valid.

Algorithm 5: *vUBLS*

Input : G: Input Uncertain Bipartite Network
t: Uncertainty Threshold
u: Selected Vertex
Output: $C_t^e(u)$: Extrapolated Uncertain Butterfly Count of u

1 $C_t^e(u) = 0$;
2 Create $H(w)$ for each Node w in same partition as u;
3 **foreach** $v \in N(u)$ **do**
4 **foreach** $w \in N(v)$ **do**
5 **if** $Pr(\angle(u,v,w)) \geq t$ **then**
6 $H(w).sortedInsert(Pr(\angle(u,v,w)))$;
7 **foreach** *Node* $w : |H(w)| > 1$ **do**
8 $C_t(u) \leftarrow C_t(u) + ImprovedListCount(H(w), t)$;
9 $C_t^e(u) \leftarrow \frac{C_t(u)|V|}{4}$;

Algorithm 5 illustrates a local sampling algorithm for the number of uncertain butterflies containing a selected vertex u. Our algorithm follows the deterministic variation [22] with the key changing being the usage of our ImprovedListCount technique introduced by us in Algorithm 3 (Lines 1-8), in the same way our baseline exact algorithm $UBFC$ was modified from the exact deterministic algorithm.

Theorem 6.1. *The time complexity of Algorithm 5 is* $O(d^2\log(d))$ *where* $d = \max_{v\in V}\{deg(v)\}$

Proof. As the idea of vertex priority does not apply in local search, the maximum number of wedges for a shared start and end point is the maximum degree of any vertex ($d = \max_{v\in V}\{deg(v)\}$) on G. As proven in Theorem 5.3, the Improved List Management technique takes $O(d\log(d))$ time thus our final time complexity is $O(d^2\log(d))$ as any start node can only have $\leq d$ end nodes. □

Theorem 6.2. *The memory complexity of Algorithm 5 is* $O(d^2)$ *where* $d = \max_{v\in V}\{deg(v)\}$

Proof. The total number of wedges that any given vertex in G can be a part of is d^2. □

With the known number of uncertain butterflies containing u, $C_t(u)$, we extrapolate that number to the entire network $C_t^e(u)$ (Line 9). We can do this as in there deterministic sampling approach, it is shown that $\mathbb{E}[C^e(u)] = C$ and $Var(C^e(u)) \leq \frac{|V|}{4}(C+p^v)$ where C is the certain butterfly count and p^v is the number of pairs of butterflies in G that share at least a single node [22]. Extending on their proof and excluding butterflies which do not satisfy t, we can trivially show that $\mathbb{E}[C_t^e(u)] = C_t$ (i.e. the estimator is unbiased) and $Var(C_t^e(u)) \leq \frac{|V|}{4}(C_t + p_t^v)$ where p_t^v is the number of pairs of uncertain butterflies in G that share at least a single node. We thus implement this in our algorithm.

Algorithm 6: *vUBS*

Input : G: Input Uncertain Bipartite Network
t: Uncertainty Threshold
$time$: Time Allocation
Output : $\tilde{C_t}$: Estimated Uncertain Butterfly Count

1 $\tilde{C_t} \leftarrow 0$;
2 $i \leftarrow 0$;
3 **while** *time has not elapsed* **do**
4 $u \leftarrow$ Unsampled Vertex from G;
5 $C_t^e(u) \leftarrow vUBLS(G, t, u)$;
6 $\tilde{C_t} \leftarrow \frac{i\tilde{C_t} + C_t^e(u)}{i+1}$;
7 $i \leftarrow i + 1$;

Algorithm 6 details our vertex-centric Uncertain Butterfly Sampling (*vUBS*) method. For a graph G, threshold t and a time allocation/budget, our method estimates the uncertain butterfly count $\tilde{C_t}$. We sample a node u without replacement and determine $C_t^e(u)$ using *vUBLS* (Algorithm 5) (Line 4-5). We then adjust $\tilde{C_t}$ accordingly (Lines 6-7). We continue this process until the time allocation is depleted. Furthermore, as extended from the deterministic version, if the algorithm is run for $O\left(\frac{|V|}{C_t}(1 + \frac{p_t^v}{C_t})\right)$ iterations, it provides an (ε, δ)-estimator [22].

Whilst we have only detailed the vertex-centric algorithm, a similar algorithm which is edge-centric is also implemented for the purposes of experimentation. This method holds the same properties and utilises the same algorithm as the vertex-centric approach except with references of vertices replaced with edges.

6.2 Proportion Estimation Sampling (PES)

Our second sampling approach utilises a different problem formulation as its basis. One notable difference between local certain and uncertain butterfly counting techniques for a selected vertex or edge is that the certain solution is theoretically less expensive in both time and space requirements. As we are interested in a sampling, and thus approximate, solution perhaps an approach in which we leverage the speed of certain sampling and correctly adapt those values to our problem is of interest.

To accomplish this, let us think of another way of formulating the Uncertain Butterfly Counting problem. Suppose we have the deterministic butterfly count on the Backbone Graph C. Then the uncertain butterfly count can be formulated as $C_t = \alpha C$ where α is the proportion of butterflies with existential probability $\geq t$.

Using this logic, if we can find α we can use deterministic local sampling approaches to quickly estimate C and thus estimate C_t. Essentially, we can think of each butterfly as a Bernoulli trial $Ber(\alpha)$ in regards to succeeding if the existential probability satisfies the threshold t.

Of course determining α exactly is extremely costly and unfeasible as it requires the discovery of all butterflies, which in itself already contains the solution to our problem. Instead, if we can find a way to estimate α we can utilise the deterministic sampling solution to quickly approach the uncertain butterfly count. Our estimation method relies on the following observation.

Lemma 5. *The PDF of Butterfly Existential Probabilities (B-PDF) can be thought of as a "random" sample of the PDF of the product of existential probabilities of in G for all permutations of four edges (E4-PDF).*

Proof. Of course, since a butterfly is a structure which contains exactly four edges, the set of all butterflies is located inside the set of all permutations of four non-identical edges. Noting that edge probabilities are independent, we can think of *B-PDF* as a sample of size C from *E4-PDF*. Notably, *E4-PFD* can be 'known' in that all relevant information can be computed in $O(E^4)$ time. That is, no assumptions need to be made about the model distribution. □

Given Lemma 5, using the Margin of Error (in percentage points) statistic for a set of Bernoulli trials [4] we can determine how much the proportion from *B-PDF* can deviate from the proportion from $E4 - PDF$ with a set confidence (i.e. a confidence interval). The Margin of Error (M) equation is as follows:

$$M = z\sqrt{\frac{\alpha(1-\alpha)}{n}} \tag{2}$$

where z is the z-table confidence interval value (e.g. 2.576 for 99% confidence) and n is the size of the sample set (i.e. C). This essentially means the estimated value of α is within the range $(\alpha - M, \alpha + M)$ 99% (or other chosen confidence) of the time for a sample of size n.

With this we can calculate, with a confidence interval, the likelihood of a sample set of size C being with a certain Margin of Error using the proportion $\hat{\alpha}$ of the $E4 - PDF$. Of course we cannot know C with certainty but we can estimate it $\tilde{C}$ via a deterministic sampling technique. A quirk of the Margin of Error equation is that small changes in the sample size provides negligible results. It is also known that the as the sample size increases, the relative change in the reduction M decreases.

We determine $\tilde{C}$ using a vertex-centric deterministic butterfly count and extrapolation sampling technique (*vBFC*, Algorithm 7) devised by Sanei et. al[22]. In this counting technique, the hashmap hold only a single integer for each end-node and once wedges are discovered the butterfly counting step can be done in $O(1)$ time. In the same work, it is also shown that $\mathbb{E}[C^e(u)] = C$ (i.e. unbiased), $Var(eC(u)) \leq \frac{V}{4}(C + p_v)$ where p_v is the pairs of butterflies that share one vertex.

Algorithm 7: *vBFC*

Input : G: Input Uncertain Bipartite Network
u: Selected Vertex
Output : $C^e(u)$: Extrapolated Deterministic Butterfly Count of u

1 $C(u) \leftarrow 0$;
2 Create $H(w) \leftarrow 0$ for each Node w in same partition as u;
3 **foreach** $v \in N(u)$ **do**
4 **foreach** $w \in N(v)$ **do**
5 $H(w) \leftarrow H(w) + 1$;
6 **foreach** *Node* $w : |H(w)| > 1$ **do**
7 $C(u) \leftarrow C(u) + \binom{H(w)}{2}$;
8 $C^e(u) \leftarrow \frac{C(u)|V|}{4}$;

From this, we can estimate the uncertain butterfly count as $\tilde{C_t} = \hat{\alpha}\tilde{C}$.

Algorithm 8: *vPES*

Input : G: Input Uncertain Bipartite Network
t: Uncertainty Threshold
$time$: Time Allocation
Output : $\tilde{C_t}$: Estimated Uncertain Butterfly Count

1 $\hat{\alpha} \leftarrow E4PropCalculation(G, t)$;
2 $\tilde{C_t}, \tilde{C} \leftarrow 0$;
3 $i \leftarrow 0$;
4 **while** *time has not elapsed* **do**
5 $u \leftarrow$ Unsampled Vertex from G;
6 $C^e(u) \leftarrow vBFC(G, u)$;
7 $\tilde{C} \leftarrow \frac{i\tilde{C}+C^e(U)}{i+1}$;
8 $i \leftarrow i + 1$;
9 $\tilde{C_t} \leftarrow \hat{\alpha}\tilde{C}$;

Our vertex-centric Proportion Estimation Sampling (*vPES*) algorithm is detailed in Algorithm 8. We first calculate $\hat{\alpha}$ by determining the proportion of the *E4-PDF* with existential probability greater than t in $O(|E|^4)$ time (Line 1). Then, for was long as our time allocation allows, we sample vertices without replacement and estimate the deterministic butterfly count $\tilde{C}$ (Lines 4-8). Finally, we output our estimated uncertain butterfly count $\tilde{C_t} = \hat{\alpha}\tilde{C}$ (Line 9). We can then calculate the Margin of Error $\hat{\alpha}$ using $\alpha = \hat{\alpha}, n = \tilde{C}$, with a given confidence value (Equation 2) then extrapolating this to confidence interval to $\tilde{C_t}$. Following other Monte-Carlo sampling approaches, this method is unbiased.

Once again, a similar edge-centric algorithm exists with minimal adjustment to Algorithms 7 and 8 whose detail we have opted to exclude in interest of space.

6.2.1 *Further Proportion Estimation Sampling.*

Of course, whilst gains have been made using *vPES* compared to *vUBS* in regards to the cost of each sample allowing for *vPES* to sample more nodes in a shorter timeframe, the trade-off of a $O(|E|^4)$ proportion estimation step is too much in reality without a dedicated system which stores the *E4-PDF*. In that regard, if we can further estimate α we can adopt the cheaper sampling process with less associated overhead.

We instead estimate the proportion by sampling from the *E4-PDF*, which results in a trade-off of an increased margin of error for a set confidence in return for a potentially significant decrease in the cost of the process.

Our new $\hat{\alpha}$ estimation is simple. For a given time budget, we randomly select four edges and determine if the product of their existential probabilities is less $\geq t$. After the time budget depletes, having conducted m samples, we have estimated $\hat{\alpha}$. We can also determine the Margin of Error M_S of this sampling process using Equation 2. Aside from this modification to Line 2, Algorithm 8 proceeds unchanged. Alternatively, given user-defined M_S and z values, it is possible to rearrange Equation 2 to determine the number of samples n. Each sample trivially takes $O(1)$ time and space. It should be noted this estimator is unbiased, where the proof is available in our implementation repository.

With the M_S and also the Margin of Error estimated from the *B-PDF* as described previously (now denoted as M_B). The Margin of Error of our final result can be calculated as $M = \sqrt{{M_S}^2 + {M_B}^2}$ assuming the same confidence value was selected for both M_S and M_B. It should be noted that via the Law of Large Numbers the number of samples required is not linear to the number of edges in the graph.

7 EXPERIMENTATION

In this section we examine the efficiency and effectiveness of our proposed algorithms on a diverse range of graphs with different edge probability distributions and threshold values. When appropriate, we will preface an algorithm with v if it is vertex-centric or e if it is edge-centric (e.g. *vUBS* is vertex-centric *UBS*).

7.1 Experiment Settings

Table 2 holds the details of all datasets used in this section. In the Edge Prob. column, if the value is a distribution this means we assigned a synthetic weight from that distribution to each edge based on the popular Normal and Uniform distributions. The parameters for normally distributed edge probabilities vary between datasets to add variety to each individual network. Alternatively, for values that say "Collaborative Filtering" we utilised existing ratings on the network to estimate the likelihood a user would to buy another item using a Collaborative Filtering Item Recommender [24]. For these networks (where $|E|$ and AvgDeg are denoted with *), we append the number of edges whose probability was estimated to the end for each experiment. For example, if for the CD dataset our experiment utilised a million edges we would label the dataset 'CD1M'. All raw datasets can be found from the Konect project [1]

Our code[2] is written in standard C++11 and compiled using g++. Our experimental environment is an Intel Xeon Gold 6240R CPU @ 2.40GHz with 1007GB of memory.

7.2 Exact Algorithms Efficiency

In this section we examine the efficiency of both variants of *IUBFC* (*IUBFC-VP* and *IUBFC-EP* in Section 5) and compare their runtimes with our proposed baseline *UBFC* (Section 4). Since they are exact solutions the effectiveness need not be compared. If an algorithm did not conclude after 100 hours of being run, we terminated it. All instances of this will be noted as required.

The runtime of each algorithm for a set of datasets is visible in Figure 4 (where $t = 0.8$). Clearly both our improved algorithms are faster than the baseline (*UBFC* for LJ and OK both did not finish after 100 hours) due to our improvements. It is also notable that average degree has a much greater impact on runtime than the number of nodes or edges (as highlighted most notably with the comparison between FL and DBLP). Our exact algorithms struggle on the larger datasets, indicating the need for our exact methods.

Figure 5 illustrates that as t approaches 1, the runtime of both *IUBFC-VP* and *IUBFC-EP* both become significantly smaller than the baseline. Notably the runtime of *IUBFC-EP* starts higher but

[1] http://konect.cc/
[2] Available at https://github.com/AlexanderTZhou/IUBFC

Table 2: Dataset Information (* indicates variable number)

Dataset	Edge Prob.	$\|L\|$	$\|R\|$	$\|E\|$	AvgDeg
YouTube (YT)	Uniform	94,238	30,087	293,360	4.719
Teams (TM)	Normal$\tilde{}$(0.8, 0.2)	901,166	34,461	1,366,466	2.921
IMDB	Normal$\tilde{}$(0.6, 0.3)	303,617	896,302	3,782,463	6.229
Flickr (FL)	Uniform	395,979	103,631	8,545,307	34.208
DBLP	Normal$\tilde{}$(0.7, 0.1)	1,953,085	5,624,219	12,282,059	3.241
LiveJournal (LJ)	Normal$\tilde{}$(0.5, 0.2)	3,201,203	7,489,073	112,307,385	21.011
Orkut (OR)	Normal$\tilde{}$(0.5, 0.25)	2,783,196	8,730,857	327,037,487	56.807
CiaoDVD (CD)	Collaborative Filtering	21,019	71,633	*	*
BookCrossing (BC)	Collaborative Filtering	77,802	185,955	*	*

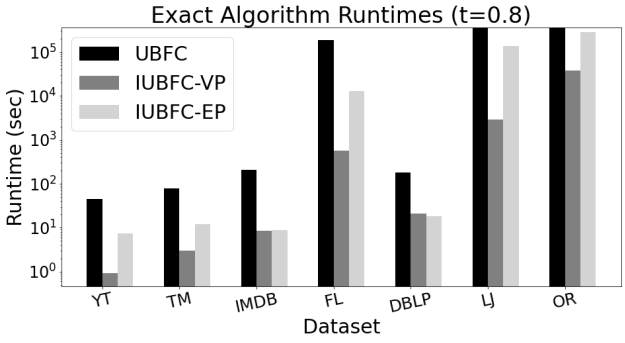

Figure 4: Runtime Comparison of exact algorithms on multiple datasets ($t = 0.8$).

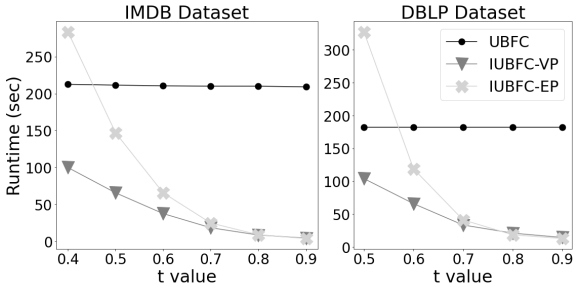

Figure 5: The change in runtime as the threshold value t changes.

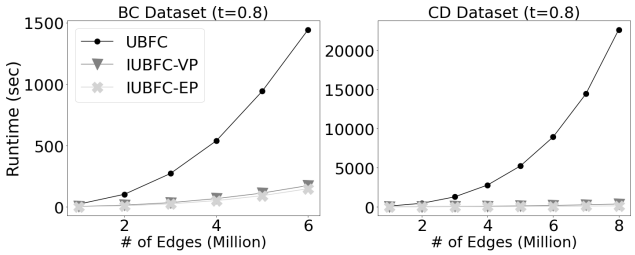

Figure 6: The change in runtime against the number of edges

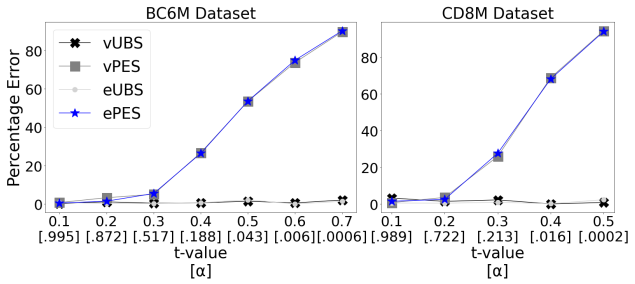

Figure 7: The average percentage error after 100 samples as t increases. Corresponding α values are also included.

drops in a march sharper manner and ultimately becomes faster than *IUBFC-VP*, as the number of uncertain wedges rapidly diminishes reducing the added costs on *IUBFC-EP* that do not exist on *IUBFC-VP* whilst maintaining the benefits (see Section 5.4).

Figure 6 examines the change in runtime as the number of edges (and subsequently average degree) changes on our Collaborative Filtering datasets. It is evident that both *IUBFC-VP* and *IUBFC-EP* are both much more scalable than the the baseline.

7.3 Sampling Algorithms Effectiveness

We examine the effectiveness of our two sampling algorithms *UBS* (Section 6.1) and *PES* (Section 6.2) using the Further Proportion Estimation method (Section 6.2.1). For all *PES* experiments, we set the desired Margin of Error M_S for Further Proportion Estimation Sampling at 0.01 and z value at 2.56.

Figure 7 details the change in percentage error as the value of t increases (and subsequently the proportion α decreases). It can be seen that all variants of *UBS* are highly effective, whilst variants of *PES* at lower t values are also efficient.There is no notable difference in effectiveness between vertex and edge-centric algorithms.

One noticeable effect is that as t increases and α grows smaller, the error of *PES* ultimately increases. This is due to the Further Proportion Estimation technique having a set Margin of Error (M_S) on $\hat{\alpha}$, which in our experimentation was set to 0.01. This of course means that our sampled proportion is allowed to deviate from the true proportion by up to that much, which is not an issue for most α values. However, for small α values this can allow for a large relative difference in acceptable $\hat{\alpha}$ values. For example if the true

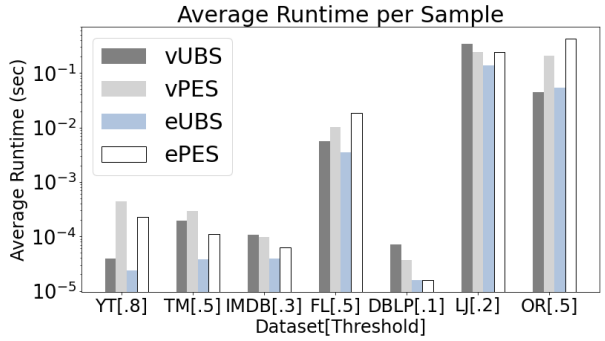

Figure 8: Average running time of a single sample for a dataset with the input threshold in square brackets.

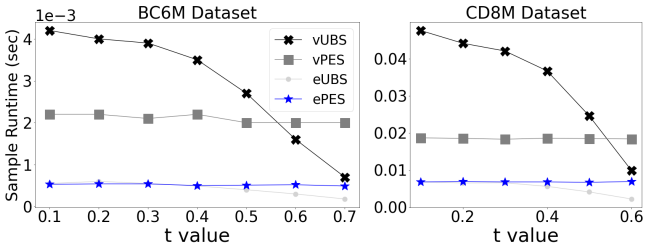

Figure 9: The change in runtime per sample as the threshold value t changes.

α value is 0.02, an $\hat{\alpha}$ value between 0.01 and 0.03 is within our allowed range despite being able to create up to a 50% change in the estimated uncertain butterfly count. As a result *UBS* should be favoured when the α value becomes extremely small. It should be noted that a very high α value will not effect the Error % of the final count in the same way. Both algorithms are similarly effective for larger α values.

7.4 Sampling Algorithms Efficiency

We now investigate the efficiency of our sampling algorithm and consider situations in which one may be preferred over the other.

The average running cost of single sample of each algorithm on a variety of different datasets and t values is visible on Figure 8. In general, the cost of a sample for all algorithms scales with the average degree of the network and not the total number of vertices or edges which means it is highly scalable for most real-world systems. It can also be observed that for each dataset, either *UBS* or *PES* is faster for both the vertex-centric approach and the edge-centric approach (we explore this below). Additionally, in general, the edge-centric algorithm is similar or faster than the vertex-centric approach. This is unsurprising as sampling an edge is equivalent to sampling two vertices as opposed to one. However, as shown in 'OR[.5]', there do exist datasets in which it is possible that the edge-centric approach can be slower. This largely occurs when a majority of edges are in a dense subgraph with a relatively smaller proportion of vertices.

Figure 9 illustrates the effect that an increasing t value (by increments of 0.1) has on the runtime of our sampling algorithms. Both *PES* algorithms have a stable runtime for all t values which is unsurprising given how both methods look at all butterflies regardless of existential probabilities. However, as $t \longrightarrow 1$, *UBS* (both vertex and edge variations) becomes the faster option due to our improved list management method as well as not adding in any useless edges into any lists. This indicates a strong preference towards *UBS* algorithms for high t values.

From our sampling experiments, we can infer that on larger t values *UBS* should be favoured due to its improved list management and edge/wedge discarding techniques resulting in a faster runtime than *PES*. On the other hand, *PES* may be favoured for lower t values (with an appropriate subsequent α value) as the runtime is stable for all values of t and can be cheaper than *UBS* as deterministic butterfly counting is less expensive than uncertain butterfly counting even after factoring in the cost associated with Further Proportion Estimation. Additionally, for lower t values *PES* does not encounter the problems associated with a very small α value which means the subsequent estimated Uncertain Butterfly Count is similar in accuracy to *UBS*.

8 CONCLUSION

In this paper we have examined the Uncertain Butterfly Counting Problem on uncertain bipartite networks. We formally defined both the uncertain butterfly and uncertain wedge as well as proposed a non-trivial baseline for the exact solution based on the state-of-the-art solution for the deterministic variant of the problem. We then proposed two improved algorithms which can drastically reduce the runtime of the algorithm. Additionally, when an efficiency for effectiveness trade-off is desired we propose two alternate sampling solutions which quickly converge near the true value. Our experiments examine all our algorithms and we also detail in which scenarios one of the methods may be preferred to the other.

As this is the first work on butterfly counting on uncertain bipartite networks there exists multiple directions to take this and similar problems, such as parallel or cache-friendly approaches. A robust heuristic which can decide which exact or approximate method to use after quickly examining a graph may be useful to avoid additional human decision making. Additionally, the core idea behind Proportion Estimation may be of interest in other uncertain problems with cheaper certain variations.

ACKNOWLEDGMENTS

Yue Wang (Corresponding Author) is partially supported by China NSFC No. 62002235 and Guangdong Basic and Applied Basic Research Foundation No. 2019A1515110473. Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2018AAA0101100, the Hong Kong RGC GRF Project 16202218, CRF Project C6030-18G, C1031-18G, C5026-18G, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, HKUST-NAVER/LINE AI Lab, HKUST-Webank joint research lab grants.

REFERENCES

[1] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *SIGMOD*, pages 34–48, 1987.

[2] M. Al Hasan and V. S. Dave. Triangle counting in large networks: a review. *WIREs DMKD*, 8(2):e1226, 2018.

[3] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting protein complex membership using probabilistic network reliability. *Genome Res.*, 14(6):1170 – 1175, 2004.

[4] J. E. Bartlett, J. W. Kortlik, and C. C. Higgins. Organizational research: Determining appropriate sample size in survey research. *ITLPJ*, 19(1):43–50, 2001.

[5] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. *SIGKDD*, page 16–24, 2008.

[6] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. *SIGKDD*, pages 1316–1325, 2014.

[7] T. Dallas, A. W. Park, and J. M. Drake. Predicting cryptic links in host-parasite networks. *PLOS Computational Biology*, 13:1–15, 05 2017.

[8] K. Han, F. Gui, X. Xiao, J. Tang, Y. He, Z. Cao, and H. Huang. Efficient and effective algorithms for clustering uncertain graphs. *PVLDB*, 12(6):667–680, 2019.

[9] X. Huang, W. Lu, and L. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. *SIGMOD*, pages 77–90, 06 2016.

[10] A. Khan and L. Chen. On uncertain graphs modelling and queries. *PVLDB*, 8(12):2042–2043, 2015.

[11] N. Korovaiko and A. Thomo. Trust prediction from user-item ratings. *Soc. Netw. Anal. Min.*, 3:749–759, 2013.

[12] R.-H. Li, J. X. Yu, R. Mao, and T. Jin. Efficient and accurate query evaluation on uncertain graphs via recursive stratified sampling. *ICDE*, pages 892–903, 2014.

[13] P. Lind, M. C. Gonzalez, and H. Herrmann. Cycles and clustering in bipartite networks. *Phys, review. E.*, 72:056127, 12 2005.

[14] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient (α, β)-core computation in bipartite graphs. *VLDBJ*, 29:1075–1099, 2020.

[15] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou. Maximum biclique search at billion scale. *PVLDB*, 13(9):1359–1372, May 2020.

[16] C. Ma, R. Cheng, L. V. S. Lakshamanan, T. Grubernmannm, Y. Fang, and X. Li. Linc: A motif counting algorithm for uncertain graphs. *PVLDB*, 13(2):155–168, 2019.

[17] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. The pursuit of a good possible world: Extracting representative instances of uncertain graphs. *SIGMOD*, pages 967–978, 2014.

[18] G. A. Pavlopoulos, P. I. Kontou, A. Pavlopoulou, C. Bouyioukos, E. Markou, and P. G. Bagos. Bipartite graphs in systems biology and medicine: a survey of methods and applications. *GigaScience*, 7(4), 02 2018.

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

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

[21] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-nearest neighbors in uncertain graphs. *PVLDB*, 3(1):997–1008, 2010.

[22] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura. Butterfly counting in bipartite networks. *SIGKDD*, 24:2150–2160, 2018.

[23] S.-V. Sanei-Mehri, Y. Zhang, A. E. Sariyüce, and S. Tirthapura. Fleet: Butterfly estimation from a bipartite graph stream. *CIKM*, page 1201–1210, 2019.

[24] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. *WWW*, pages 285–295, 2010.

[25] I. Sungur, Y. Ren, F. Ordóñez, M. Dessouky, and H. Zhong. A model and algorithm for the courier delivery problem with uncertainty. *Transportation Science*, 44(2):193–205, 2010.

[26] S. Suthram, T. Shlomi, E. Ruppin, R. Sharan, and T. Ideker. A direct comparison of protein interaction confidence assignment schemes. *BMC Bioinformatics*, 7:360 – 370, 2006.

[27] J. G. Walker, M. Plein, E. R. Morgan, and P. A. Vesk. Uncertain links in host–parasite networks: lessons for parasite transmission in a multi-host system. *Philos. Trans. R. Soc. B*, 372, 2017.

[28] J. Wang, A. W. Fu, and J. Cheng. Rectangle counting in large bipartite graphs. *BigData*, pages 17–24, 2014.

[29] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *PVLDB*, 12(10):1139–1152, 2019.

[30] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Efficient bitruss decomposition for large-scale bipartite graphs. *ICDE*, pages 661–672, 2020.

[31] B. Wilder, A. Yadav, N. Immorlica, E. Rice, and M. Tambe. Uncharted but not uninfluenced: Influence maximization with an uncertain network. *AAMAS*, 16:1305–1313, 2017.

[32] M. M. Wolf, M. Deveci, J. W. Berry, S. D. Hammond, and S. Rajamanickam. Fast linear algebra-based triangle counting with kokkoskernels. *HPEC*, pages 1–7, 2017.

[33] Y. Yuan, L. Chen, and G. Wang. "efficiently answering probability threshold-based shortest path queries over uncertain graphs. *DASFAA*, pages 155–170, 2010.

[34] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient keyword search on uncertain graph data. *TKDE*, 25(12):2767–2779, 2013.

[35] Y. Zeng, Y. Tong, and L. Chen. Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees. *PVLDB*, 13(3):320–333, 2020.

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

[37] B. Zhao, J. Wang, M. Li, F. Wu, and Y. Pan. Detecting protein complexes based on uncertain graph model. *IEEE/ACM Trans. Comput. Biol. Bioinform*, 11(3):486–497, 2014.

[38] A. Zhou, Y. Wang, and L. Chen. Finding large diverse communities on networks: The edge maximum k*-partite clique. *PVLDB*, 13(12):2576–2589, July 2020.

[39] T. Zhou, J. Ren, M. c. v. Medo, and Y.-C. Zhang. Bipartite network projection and personal recommendation. *Phys. Rev. E*, 76:046115, Oct 2007.

[40] Z. Zou. Bitruss decomposition of bipartite graphs. *DASFAA*, page 218–233, 2016.

[41] Z. Zou and R. Zhu. Truss decomposition of uncertain graphs. *Knowl. Inf. Syst.*, 50(1):197–230, 2017.