Exact and Approximate Algorithms for the Most Connected Vertex Problem

Cheng Sheng[1], Yufei Tao[1,2], Jianzhong Li[3]

[1]Chinese University of Hong Kong
[2]Korea Advanced Institute of Science and Technology
[3]Harbin Institute of Technology

An *(edge) hidden graph* is a graph whose edges are not explicitly given. Detecting the presence of an edge requires an expensive *edge-probing* query. We consider the *k most connected vertex* (k-MCV) problem on hidden bipartite graphs. Given a bipartite graph G with independent vertex sets B and W, the goal is to find the k vertices in B with the largest degrees using the minimum number of queries. This problem can be regarded as a top-k extension of semi-join, and is encountered in several applications in practice.

If B and W have n and m vertices respectively, the number of queries needed to solve the problem is nm in the worst case. This, however, is a pessimistic estimate on how many queries are necessary on practical data. In fact, on some inputs, the problem may be settled with only $km+n$ queries, which is significantly lower than nm for $k \ll n$. The huge difference between $km + n$ and nm makes it interesting to design an adaptive algorithm that is guaranteed to achieve the best possible performance on every input G. For $k \leq n/2$, we give an algorithm that is *instance optimal* among a broad class of solutions. This means that, for any G, our algorithm can perform more queries than the optimal solution (which is unknown) by only a constant factor, which can be shown to be at most 2.

As a second step, we study an ϵ-approximate version of the k-MCV problem, where ϵ is a parameter satisfying $0 < \epsilon < 1$. The goal is to return k black vertices $b_1, ..., b_k$ such that the degree of b_i ($i \leq k$) can be smaller than t_i by a factor of at most ϵ, where $t_1, ..., t_k$ (in non-ascending order) are the degrees of the k most connected black vertices. We give an efficient randomized algorithm that successfully finds the correct answer with high probability. In particular, for a fixed ϵ and a fixed success probability, our algorithm performs $o(nm)$ queries in expectation for $t_k = \omega(\log n)$. In other words, whenever t_k is greater than $\log n$ by more than a constant, our algorithm beats the $\Omega(nm)$ lower bound for solving the k-MCV problem exactly. All the proposed algorithms, despite the complication of their underlying theory, are simple enough for easy implementation in practice. Extensive experiments have confirmed that their performance in reality agrees with our theoretical findings very well.

Categories and Subject Descriptors: F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms: Theory

Additional Key Words and Phrases: Maximum Degree, Bipartite Graph, Competitive Analysis

Author's address: C. Sheng (*csheng@cse.cuhk.edu.hk*), Department of Computer Science and Engineering, Chinese University of Hong Kong, Sha Tin, Hong Kong; Y. Tao (*taoyf@cse.cuhk.edu.hk*), Affiliation 1: Department of Computer Science and Engineering, Chinese University of Hong Kong, Sha Tin, Hong Kong; Affiliation 2: Division of Web Science and Technology, Korea Advanced Institute of Science and Technology, Korea; J. Li (*lijzh@hit.edu.cn*), School of Computer Science and Technology, Harbin Institute of Technology, China.

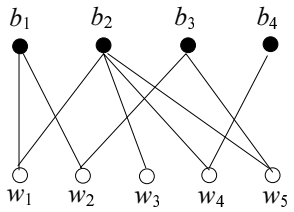


Fig. 1. The 1-MCV result is b_2

1. INTRODUCTION

An *(edge) hidden graph* is a graph whose edges are not explicitly available. Detecting the presence of an edge between two vertices requires an *edge-probing query*, which is an operation that incurs expensive cost. In recent years, *learning hidden graphs* [Goldreich et al. 1998] has attracted considerable attention in the theory community [Alon and Shapira 2008a; Angluin and Chen 2008; Bogdanov et al. 2002; Goldreich et al. 1998]. The main objective of the relevant research is to decide whether the graph has a certain *property*, by probing the least number of edges. The underneath rationale is that, learning only a property of the graph (e.g., whether it is bipartite) is easier than revealing the whole graph. Therefore, the number of edges that need to be probed may be significantly smaller than the total number of edges that may exist.

As will be reviewed in Section 2, the existing research on hidden graphs is mostly motivated by biological and chemical applications. This paper focuses on the database context. We consider the k *most connected vertex* (k-MCV) problem on hidden bipartite graphs. Specifically, given a bipartite graph G between two sets B and W of vertices, the objective is to find the k vertices in B having the maximum degrees. In Figure 1, for example, B has vertices $\{b_1, ..., b_4\}$, and W is $\{w_1, ..., w_5\}$; the output of the 1-MCV problem is b_2. The challenge is to minimize the number of edge-probing queries. Next, we discuss several applications of the k-MCV problem.

1.1 Motivation

Application 1 (*semi-join aggregation with complex predicates*). Consider B and W as relational tables, and a join predicate between B and W. An edge-probing query in this scenario examines whether a tuple of B can be joined with a tuple of W. The result of the k-MCV problem is the k tuples in B that can be joined with the most tuples in W, as described by the following pseudo-SQL statement:

```
SELECT  b
FROM    B b, W w
WHERE   [a join predicate about b and w]
GROUP BY  b
HAVING  count(*) ≥ the size of the k-th largest group
```

Notice that, if we remove the GROUP-BY and HAVING clauses, the statement becomes a standard *semi-join*. Hence, k-MCV can be regarded as a *top-k* extension of a semi-join, which returns the k tuples of table B having the strongest joining power with respect to table W. For example, suppose that B is a list of hotels, and W is a list of tour attractions. Setting an edge-probing query to check whether a hotel b and an attraction w are within 1

mile, the above statement is essentially a *top-k spatial join* [Zhu et al. 2005], which finds the k hotels whose 1-mile vicinities cover the largest number of attractions.

The join predicate can be rather unfriendly to relational query optimization. For example, the simple geometric condition given earlier (deciding whether b and w are within 1 mile) is not well supported by a DBMS. This is especially true if the "1 mile" refers to the *road network* distance, in which case evaluating the join predicate may even need to perform a *shortest-path* search on a map. If effective optimization is impossible, the DBMS may execute the statement by first performing a cartesian product between B and W, followed by a group-by and selection of the largest groups. Such a strategy may incur prohibitive cost.

A remedy in the above situation is a fast algorithm for solving the k-MCV problem, which can improve efficiency dramatically by reducing the number of times that the join-predicate is evaluated (i.e., the number of edges probed). Note that, to be incorporated in a relational engine, such an algorithm must be general enough to tackle *any join predicate*, as opposed to only special queries. For this reason, the solutions of [Zhu et al. 2005] are not appropriate for DBMS incorporation.

In fact, the concept of semi-join exists not only in relational databases, but is implicit in the applications of other environments. As detailed below, the k-MCV problem finds use in those applications as well.

Application 2 (*frequent patterns*). Assume that each vertex $b \in B$ represents a candidate pattern, and each vertex $w \in W$ corresponds to a data item. Given a pattern $b \in B$ and a data item $w \in W$, an edge-probing query detects whether b exists in w. In other words, there is an edge in G between b and w if b is observed in w. The k-MCV problem returns the k patterns in B that are most commonly found in the items of W. In some environments, detecting the presence of a pattern can be rather expensive, such that the overall computation time is dominated by the total cost of all queries.

As an example, the pharmaceutical industry has established a novel methodology of discovering new drugs, called *fragment-based drug discovery* [Kapoor 2000]. This is motivated by the frustration that *"finding a new drug is like playing golf, where the target is the pin"* [Kapoor 2000]. The new methodology relieves the frustration by initiating a drug-searching process from a *fragment*, which is a basic chemical compound in the molecular structures of drugs. Hence, an important problem is to identify the k fragments that are most frequently present in a set of drugs. This is a typical k-MCV problem, where B includes all the fragments, and W is the set of drugs under screening. An edge-probing query checks whether a fragment $b \in B$ exists in a drug $w \in W$. Since molecular structures are graphs, the query essentially carries out a *subgraph isomorphism test* [Garey and Johnson 1979], which can be rather costly. Therefore, reducing the number of queries is the key to efficiency.

In general, pattern detection is often achieved by evaluating the distance between a pattern and a data item: a pattern is considered to exist if the distance is sufficiently small. Some distance functions are expensive to evaluate, e.g., *dynamic time warping* [Keogh 2002] and even ℓ_p norms in *ultra-high dimensional spaces* [Houle and Sakuma 2005]. In those cases, the cost of edge-probing queries may dominate the execution time, justifying the need to minimize such queries.

Application 3 (*querying by web service*). Today, many websites provide convenient interfaces to allow the public to query their backend databases. Such services have significantly

increased the amount of data that an ordinary user can access, by removing the need for the user to store gigantic datasets locally. For instance, at *Cinema Freenet* (*www.cinfn.com*), people can input the name of an actor/actress and the title of a movie; then the website will return, among other information, whether the actor/actress played a role in the movie. As another example, using the APIs of *Google Map*, a program is able to obtain the road-network distance between two addresses given in the text format, i.e., the coordinate information of neither address is necessary.

These services can be leveraged to solve k-MCV problems in a way we call *querying by web service*. For example, assume that B is a set of actors and actresses, and W is a set of movies. Given an actor/actress $b \in B$ and a movie $w \in W$, an edge-probing query contacts *Cinema Freenet* to verify whether b appeared in W. The k-MCV result is the k actors/actresses that participated in the largest number of movies. In a similar way, *Google Map* can be employed to solve the *top-k spatial join* problem mentioned in Application 1, *without* knowing the coordinates of the hotels and tour attractions at all. Given a hotel $b \in B$ and an attraction $w \in W$, a query connects to *Google Map* to check if the distance from b to w is within 1 mile. Then, the output of k-MCV is the k hotels that have the most attractions within their 1-mile neighborhoods. The performance bottleneck in the above environments is the total network latency of the queries issued. Once again, minimizing the number of queries should be the aim of a k-MCV algorithm.

1.2 Our main results

The first objective of this work is to design a generic algorithm for the k-MCV problem that can be directly used as a black box in all the above applications. If the vertex sets B and W have sizes n and m respectively, in the worst case, solving the problem demands nm edge-probing queries. However, nm is a very pessimistic estimate on how many queries are needed on practical data. As we will see, on some inputs, the problem can be settled with only $km + n$ queries, which is significantly lower than nm for $k \ll n$.

The above discussion suggests that it is a wrong direction to design a *worst-case optimal* algorithm – virtually *any* correct algorithm is worst-case optimal. In fact, the wide spectrum between $km + n$ (good case) and nm (worst case) indicates that we should aim at an *adaptive* algorithm, which is guaranteed to achieve the lowest cost on *every* input. Intuitively, the cost of the algorithm ought to be a function of the difficulty of the input. Namely, when the input is "easy", the algorithm must perform far less than nm queries. As the input's hardness increases, the cost of the algorithm is allowed to grow, but only to the extent enough to tackle the additional difficulty.

This paper presents the first study on the k-MCV problem. For $k \leq n/2$, we propose an algorithm with the properties described earlier, and prove that it is *instance optimal* among a class of solutions (to be defined in the next section). Instance optimality [Fagin et al. 2001] requires that, on *any* data input, our algorithm should be as fast as the optimal solution (which is unknown), up to only a constant factor. We are able to show that the constant is at most 2. In practice, k is usually very small (e.g., 10) compared to the size n of B, such that it can be regarded as a constant. In this case, we prove that our algorithm can be slower than the optimal solution by only a tiny factor of $1 + O(1/n)$.

As a second step, we study an ϵ-approximate version of the k-MCV problem where ϵ is a constant satisfying $0 < \epsilon < 1$. Denote by t_1, ..., t_k (in non-ascending order) the degrees of the k most connected vertices in B. Then, the *ϵ-approximate k-MCV* problem returns k vertices where the i-th ($i \leq k$) vertex has a degree at least $t_i(1 - \epsilon)$, that is,

the degree of this vertex can be lower than t_i by no more than a factor of ϵ. We give a randomized algorithm that returns the correct answer with probability at least $1 - \delta$, and performs $O(\frac{1}{\epsilon^2}\frac{nm}{t_k}\log\frac{n}{\delta})$ queries in expectation. Note that, for fixed ϵ and δ, the cost of our algorithm is bounded by $O(\frac{nm}{t_k}\log n)$, thus beating the lower bound $\Omega(nm)$ of the exact k-MCV problem whenever $t_k = \omega(\log n)$, namely, the degrees of all the result vertices are greater than $\log_2 n$ by more than a constant. In practice, $\log_2 n$ is a small value (e.g., for n being a million, $\log_2 n$ is roughly 20). Hence, the finding suggests that approximate algorithms may have a performance advantage over exact solutions in the *worst case*. For example, our approximate algorithm is more superior in semi-join aggregation (Application 1 of Section 1.1), when at least k black tuples each match, say, at least 1% of the white vertices. In such a case, the approximate solution incurs only $O(n \log n)$ cost, as opposed to the $O(nm)$ cost of the exact algorithm.

The rest of the paper is organized as follows. The next section defines the problem and reviews the previous work related to ours. Then, Section 3 sets the stage for theoretical analysis by defining the algorithm classes, and giving some basic probabilistic facts. Section 4 explains the details of the proposed algorithms for the exact k-MCV problem, whose performance is studied in Section 5. Section 6 is devoted to the ϵ-approximate k-MCV problem, by giving our algorithmic solutions and analyzing their performance. Section 7 experimentally evaluates the efficiency of the proposed techniques. Finally, Section 8 concludes the paper with directions for future work.

2. PROBLEM AND RELATED WORK

We first expand the discussion in Section 1 to formally define the k *most-connected vertex* (k-MCV) problem and its approximate version. Then, we review the existing research on the relevant problems.

The k-MCV problem. Let $G = (B, W, E)$ be a bipartite graph, where the set E of edges are between a set B of *black vertices*, and a set W of *white vertices*. G is a *hidden graph*, meaning that none of the edges in E is explicitly given. To find out whether an edge exists between a vertex $b \in B$ and a vertex $w \in W$, we must perform an *edge-probing query* $q(b, w)$, which returns a boolean answer *yes* or *no*. The edges of G that have not been probed are said to be *hidden*. The goal of the k-MCV problem is to find the k black vertices with the largest degrees, by minimizing the number of queries, or equivalently, the number of edges probed.

Two black vertices may have the same degree, namely, a tie. For the sake of fairness, we adopt the policy that the vertices having a tie should receive the same treatment. That is, either they are all reported, or none of them is reported. This means that sometimes the result may have more than k vertices. Formally, denote by $deg(b)$ the degree of a black vertex $b \in B$; the k-MCV result is the *minimal* set R of black vertices satisfying:

(1) $|R| \geq k$, and
(2) $deg(b) > deg(b')$ for any $b \in R$ and $b' \in B \setminus R$

where $|R|$ denotes the size of R, and $B \setminus R$ is the set difference between B and R.

The above definition aims to retrieve vertices with large degrees, whereas a symmetric definition exists for extracting vertices with small degrees. Throughout the paper, we focus on the former version because our solutions can be directly applied to the latter version by working with the complement of G, i.e., a bipartite graph $\bar{G}$ that has B and W as the vertex

sets, and has an edge between $b \in B$ and $w \in W$ if and only if there is no edge between b and w in G.

Denote by n and m the numbers of vertices in B and W, respectively (i.e., G can have between 0 and nm edges). Imagine that we have ranked all the vertices of B in non-ascending order of their degrees, breaking ties arbitrarily. We refer to the i-th ($1 \leq i \leq n$) vertex in the ranked list as the *i-th most connected vertex* in B.

We consider that the value of k is an integer from 1 to $n/2$. In practice, users are usually interested in the *top few* (e.g., 10) black vertices with the maximum or minimum degrees. This implies that ideally a good solution to the k-MCV problem should be especially efficient for $k = O(1)$.

The ϵ-approximate k-MCV problem. Besides the inputs in the exact version of problem, we are given an extra parameter ϵ satisfying $0 < \epsilon < 1$ to control the relative precision. Denote by t_1, ..., t_k the degrees of the k most connected black vertices in G. The ϵ-approximate k-MCV problem aims at returning k black vertices b_1, ..., b_k such that for any $1 \leq i \leq k$:

$$deg(b_i) \geq t_i(1 - \epsilon)$$

namely, the degree of b_i is smaller than t_i by at most a factor of ϵ.

Related work. Although *graph databases* have been extensively studied (see [Angles and Gutiérrez 2008] for a recent survey), we are not aware of any previous work dealing with the k-MCV problem on hidden graphs. Traditionally, the edges of a graph are given explicitly (e.g., in an adjacency matrix/list), so that accessing an edge incurs negligible cost. In that scenario, finding the k vertices with the largest degrees is a trivial task. A distinctive feature of our k-MCV problem is that detecting an edge is costly, such that the number of edge-probing queries determines the overall execution time.

Learning hidden graphs, also known as *graph testing*, was first studied by Goldreich et al. [1998]. At a high level, given a hidden graph G, the objective of learning is to either *confirm* that G has a certain property, or *deny* the existence of the property in G. A fuzzy answer *don't-care* is allowed when G is *close* to having such a property. For example, a property that has been widely studied [Alon and Krivelevich 2002; Bogdanov et al. 2002; Goldreich et al. 1998] is whether G is bipartite. A *don't-care* answer is permitted when G can be converted to a bipartite graph by adding/removing only a small number of edges. The learning of other properties has also been investigated; see, for example, [Alon and Shapira 2008a; 2008b] for a summary.

In the original setup of [Goldreich et al. 1998], an edge-probing query is assumed to detect an edge between only two vertices. In recent years, several authors [Alon et al. 2004; Angluin and Chen 2008; Biedl et al. 2004] have considered *super queries*, each of which detects whether at least an edge exists among a set of vertices in the underlying graph. This is motivated by biological and chemical applications. For example, consider a *reaction graph*, where each vertex is a chemical, and two vertices are connected if the corresponding chemicals react with each other. Then, a super query can be understood as an experiment of mixing several different chemicals, and observing if any reaction happens. If yes, it implies that at least two of the chemicals involved react with each other.

Our k-MCV problem differs from the *graph testing* formulation of [Goldreich et al. 1998]. Specifically, we are not attempting to verify any general property. Instead, we aim at identifying particular vertices in the *given* graph satisfying our degree requirements. This

is analogous to retrieving the items of a dataset qualifying a query condition, as opposed to recognizing which distribution best describes the dataset. To our knowledge, the k-MCV problem has not been addressed in the literature of graph testing.

Finding the vertex with the maximum degree is a basic operation in attacking several problems on bipartite graphs. Our algorithms can be applied as a building brick in those problems, under the circumstances where detecting the presence of edges is expensive. An important example is the *minimum set cover* (MSC) problem. In the context of a bipartite graph between two vertex sets B and W, the MSC problem is to compute the minimum subset $B' \subseteq B$ such that every vertex in W is connected to at least one vertex in B'. The problem is NP-hard but a good approximate solution can be found by a classic greedy algorithm [Cormen et al. 2001], which requires solving multiple 1-MCV problems. Our techniques can be immediately employed.

The concept of instance optimality was introduced by Fagin et al. [2001]. An earlier, similar, concept is *competitive analysis* [Borodin and El-Yaniv 1998], whose differences from instance optimality are nicely explained in [Fagin et al. 2001]. Instance optimal algorithms have been designed for many other problems, for example, manipulating binary search trees [Demaine et al. 2009], approximating the distance from a point to a curve [Baran and Demaine 2005], computing the union/intersection of sorted lists [Demaine et al. 2000], finding the convex hull of polygons [Barbay and Chen 2008], to mention just a few. Recently, a generic framework has been developed in [Afshani et al. 2009] to design instance optimal algorithms for geometric problems.

The k-MCV problem can be regarded as a variant of the *top-k problem*, which has been extensively studied in distributed systems [Fagin et al. 2001], relational databases [Ilyas et al. 2008], uncertain data [Soliman et al. 2008], and so on. However, the solutions in those works are specific to their own contexts, and cannot be adapted for k-MCV. Another related problem in relational databases is *top-k join* [Ilyas et al. 2003; Natsev et al. 2001; Schnaitter and Polyzotis 2008], which returns the top-k tuples from a join with the highest *scores*. The score of a (joined) tuple is calculated from a monotone function based on the tuple's attributes. The ranking criteria in k-MCV, on the other hand, are not based on any attribute, but instead, depend on the *joining power* of a tuple in a participating relation (i.e., it can be joined with how many tuples from the opposite relation).

A preliminary version of this work was published in [Tao et al. 2010]. While that short version studies only the exact k-MCV problem, the current article also provides solutions with theoretical guarantees to the ϵ-approximate k-MCV problem (in Section 6), and accordingly, includes the extra empirical results (Section 7).

3. PRELIMINARIES

This section will first explain the classes of algorithms considered for the exact k-MCV problem. Then, we will elaborate the concept of instance optimality, based on the framework established by [Fagin et al. 2001]. Finally, we will review Chernoff bounds.

Classes of exact algorithms. We aim at designing generic algorithms that do not assume any pre-knowledge of the underlying graph G. In other words, the algorithm obtains information about G only from the problem input (i.e., the vertex sets B and W), and the results of the edge-probing queries already performed. To make our discussion more specific, Figure 2 describes a high-level framework to capture k-MCV algorithms. The framework describes two core operations performed repetitively by an algorithm:

algorithm *MCV*
input: a hidden bipartite graph
output: the k-MCV result
1. **repeat**
2. b = *pick-black*
3. *probe-next*(b)
4. **until** it is safe to return the result

Fig. 2. An algorithmic framework

—*pick-black*, which returns the black vertex b on which the algorithm wants to probe a hidden edge, according to the current status of the algorithm's execution. Different strategies can make a huge difference. This is the key of the algorithm design.

—*probe-next*(b), which reveals an edge of b that is still hidden at this time. Specifically, it selects a white vertex w whose edge with b has not been probed, and performs a query $q(b, w)$.

It would be ideal if we could implement *probe-next*(b) in a way that can *selectively* probe an edge that is likely to be present or absent. This, however, implies that we must know at least something about G, such as the correlations among the edges. Since our objective is to propose a generic algorithm, it appears unjustified to favor a specific application by leveraging its properties, since this will inevitably disfavor another application that does not have such properties. Hence, we focus on two "neutral" versions of *probe-next*(b):

—*Randomized.* A randomized *probe-next*(b), as shown in Figure 3, probes any hidden edge of b with the same probability. This is reasonable when the algorithm cannot predict the nature (i.e., present or not) of any hidden edge.

—*Deterministic.* Assume that the m white vertices in W are arranged into a sequence $(w_1, w_2, ..., w_m)$. A deterministic *probe-next*(b), as shown in Figure 4, probes the next hidden edge of b according to the sequence. This is reasonable in scenarios where the white vertices must be accessed sequentially due to a limitation on access pattern in the underlying application. For example, in an *air index* described in [Imielinski et al. 1997], a server periodically broadcasts the data objects in a round robin fashion, whereas a client receives the objects in the order they appear in the broadcasting sequence. Another advantage of deterministic implementation is that it removes the need of remembering which edges have already been probed (such information must be maintained for the random version of *probe-next*(b)).

Depending on which version of *probe-next*(b) is adopted, the algorithmic framework of Figure 2 is specialized into two algorithm classes: $\mathcal{A}_{\mathrm{RAN}}$ and $\mathcal{A}_{\mathrm{DET}}$. Specifically, $\mathcal{A}_{\mathrm{RAN}}$, referred to as the *random-probe algorithm class*, includes algorithms that apply the randomized version; $\mathcal{A}_{\mathrm{DET}}$, the *deterministic-probe algorithm class*, contains algorithms that apply the deterministic version. In each class, the algorithms differ in their implementations of *pick-black*.

Instance optimality. In the worst case, nm edge-probing queries are needed to solve the k-MCV problem. To prove this, consider an input G with no edge at all, namely, no black vertex is connected to any white vertex. Any algorithm correctly solving the 1-MCV

algorithm *probe-next*(b)

/* for the *random-probe* class $\mathcal{A}_{\mathrm{RAN}}$; an algorithm of this class probes the edges of a black vertex in random order */

1. **if** b has no more hidden edge **then**
2. **return** NULL
3. w = a random vertex of W whose edge with b has not been probed.
4. **return** $q(b, w)$

Fig. 3. Randomized *probe-next*(b)

algorithm *probe-next*(b)

/* for the *deterministic-probe* class $\mathcal{A}_{\mathrm{DET}}$; for every black vertex, an algorithm of this class probes its edges by the same sequence of white vertices $(w_1, w_2, \ldots, w_m)$ */

1. i = the number of edges of b that have been probed
2. **if** $i = m$ **then return** NULL
3. **return** $q(b, w_{i+1})$

Fig. 4. Deterministic *probe-next*(b)

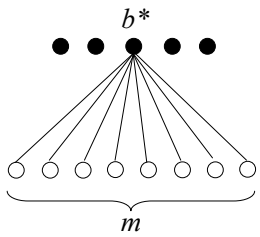


Fig. 5. An easy input to 1-MCV

problem on this graph must probe the edge between *each* pair of black and white vertices, before it can conclude that all black vertices have degree 0. Skipping any edge, say between $b \in B$ and $w \in W$, leaves the risk that b may have a degree of 1.

Worst case analysis often incurs the criticism of being over conservative in practice. In our problem, the previous paragraph indicates that the worst-case cost of solving k-MCV is nm anyway. So by this yardstick, it does not even make sense to study the problem, because all algorithms are equally bad. This, however, is a pessimistic judgment because it is possible to do much better than the worst case on many inputs. To make our argument solid, consider an input G where one vertex $b^\star$ in B has degree m (i.e., $b^\star$ has an edge with every vertex in W), and all the other $n-1$ vertices in B have degree 0 (see Figure 5). It is easy to see that the 1-MCV problem can be solved by issuing less than $m+n$ queries. Specifically, we can probe all the edges of $b^\star$, and only *one* edge for every other black vertex $b \in B, b \neq b^\star$. The total number of queries is $m+n-1$, but this is enough to find out that $b^\star$ has degree m, and that any other black vertex b has degree at most $m-1$. Therefore, $b^\star$ must be the only vertex in the result.

Motivated by this, we turn our attention to designing an algorithm that guarantees the best performance on *every* input. Specifically, on difficult inputs that require nm queries anyway, our algorithm does not achieve any improvement. However, on easier inputs, our algorithm incurs lower cost, actually so low that it is provably as fast as even the optimal algorithm (which remains unknown currently), up to a small factor.

Next, we formalize the above discussion using the concept of *instance optimality* introduced by [Fagin et al. 2001]. This concept requires an algorithm to be optimal on every data input, and is thus stronger than worst-case optimality. In general, let $\mathcal{A}$ be a class of algorithms, and $\mathcal{D}$ a family of datasets. Denote by $cost(A, D)$ the cost of algorithm $A \in \mathcal{A}$ on dataset $D \in \mathcal{D}$. Then, an algorithm $A^\star \in \mathcal{A}$ is *instance optimal* over $\mathcal{A}$ and $\mathcal{D}$ if there is a constant r satisfying

$$cost(A^\star, D) \leq r \cdot cost(A, D) \tag{1}$$

for any $A \in \mathcal{A}$ and any $D \in \mathcal{D}$.

In our context, $\mathcal{A}$ is either $\mathcal{A}_{\mathrm{RAN}}$ or $\mathcal{A}_{\mathrm{DET}}$, and $\mathcal{D}$ includes all the bipartite graphs. Note that while all the algorithms in $\mathcal{A}_{\mathrm{RAN}}$ must be randomized, those in $\mathcal{A}_{\mathrm{DET}}$ can be either randomized or deterministic, depending on their implementations of *pick-black*. In any case, we define $cost(A, G)$ to be the *expected cost* of an algorithm A (in $\mathcal{A}_{\mathrm{RAN}}$ or $\mathcal{A}_{\mathrm{DET}}$) on the input graph $G \in \mathcal{D}$, where cost is measured by the number of edge-probing queries performed by A. This definition trivially applies to a deterministic A, whose $cost(A, G)$ is simply its single-execution cost on G.

Our objective is to find an $A^\star$ in each algorithm class that makes (1) hold. Furthermore, it is important to keep the constant r as small as possible. In particular, a much stronger result is obtained if r can be shown to *decrease* with the size of the input. For example, if an algorithm achieves an $r = 1 + 1/n$, then the algorithm is not only instance optimal (notice that $1 + 1/n$ is at most 2), but is nearly optimal in the absolute sense for large n (in which case r is very close to 1).

Chernoff bounds. The essence of Chernoff bounds is that the summation of independent random variables often does not deviate much from the summation of their respective expectations. Actually, we need only a special case, where all those variables follow the Bernoulli distribution. Specifically, let X_1, ..., X_s be independent Bernoulli variables, all with success probability p. In other words, X_i equals 1 with probability p, and 0 with probability $1 - p$. Note that the sum of X_1, ..., X_s equals sp in expectation. The standard Chernoff bounds [Hagerup and Rub 1990] state that, for any $\alpha > 0$:

$$\Pr\left[\sum_{i=1}^{s} X_i \geq (1+\alpha)sp\right] \leq \left(\frac{e^\alpha}{(1+\alpha)^{(1+\alpha)}}\right)^{sp} \tag{2}$$

and for α satisfying $0 < \alpha < 1$:

$$\Pr\left[\sum_{i=1}^{s} X_i \leq (1-\alpha)sp\right] \leq \left(\frac{e^\alpha}{(1+\alpha)^{(1+\alpha)}}\right)^{sp} \tag{3}$$

The above inequalities are a bit complex, and may not be convenient to apply. The proposition below gives some simpler but weaker alternatives.

PROPOSITION 1. *Let $X_1, ..., X_s$ be s independent Bernoulli variables with success*

probability p. It holds that:

$$\Pr\left[\sum_{i=1}^{s} X_i \geq (1+\alpha)sp\right] \leq \exp\left(\frac{-sp\alpha^2}{3}\right), \qquad \textit{when } 0<\alpha<1 \qquad (4)$$

$$\Pr\left[\sum_{i=1}^{s} X_i \geq (1+\alpha)sp\right] \leq \exp\left(\frac{-(1+\alpha)sp}{6}\right), \qquad \textit{when } \alpha \geq 1 \qquad (5)$$

$$\Pr\left[\sum_{i=1}^{s} X_i \geq (1+\alpha)sp\right] \leq \left(\frac{e}{1+\alpha}\right)^{(1+\alpha)sp}, \qquad \textit{when } \alpha > 0 \qquad (6)$$

$$\Pr\left[\sum_{i=1}^{s} X_i \leq (1-\alpha)sp\right] \leq \exp\left(\frac{-sp\alpha^2}{3}\right), \qquad \textit{when } 0<\alpha<1 \qquad (7)$$

PROOF. The proofs of (4) and (7) can be found in [Hagerup and Rub 1990]. To prove (5) and (6), first notice that

$$\left(\frac{e^{\alpha}}{(1+\alpha)^{(1+\alpha)}}\right)^{sp} = \left(\frac{e^{\alpha/(1+\alpha)}}{1+\alpha}\right)^{(1+\alpha)sp}.$$

Thus, (6) follows immediately from (2) and the fact that $e^{\alpha/(1+\alpha)} < e$. Now, define:

$$f(\alpha) = \frac{e^{\alpha/(1+\alpha)}}{1+\alpha}$$

which is monotonically decreasing, because $\frac{\mathrm{d}}{\mathrm{d}\alpha}(\ln f) = (1+\alpha)^{-2} - (1+\alpha)^{-1} < 0$. As a result, $f(\alpha) \leq f(1) \approx 0.824 < e^{-1/6}$ when $\alpha \geq 1$. This, together with (2), establishes (5). □

The inequalities of the above proposition are useful in establishing the theoretical guarantees of the proposed solutions to the approximate k-MCV problem. As discussed in Section 6, our algorithms probe the edges of the input graph in a random fashion. As far as a black vertex is concerned, if we randomly pick one of its edges, the event that the edge is solid happens with a fixed probability, namely, the event can be described by a Bernoulli random variable. The Chernoff bound will then be used to estimate the number of solid edges among its edges that have been probed.

4. EXACT ALGORITHMS

In this section, we give two algorithms for solving the (exact) k-MCV problem. The first one, called *sample-sort*, is based on a simple sampling idea. It is included because, in general, it is good practice to *disprove* the efficiency of straightforward solutions, before moving to more complex methods. Indeed, we give an argument in the next section showing that *sample-sort* fails to be instance optimal. Our second algorithm, called *switch-on-empty*, is less intuitive, but turns out to be instance optimal.

Notations and basic strategy. Let us first introduce some key notations and explain a basic bounding strategy. Recall that, $deg(b)$ denotes the degree of a black vertex $b \in B$. Let $R \subseteq B$ be the set of black vertices that an algorithm A decides to return. As mentioned in Section 2, A must have evidence showing:

$$\text{for any } b \in R \text{ and } b' \notin R,\ deg(b) > deg(b').$$

algorithm *sample-sort*(s)

/* for each $b \in B$, $solid(b)$ and $empty(b)$ are dynamically maintained throughout the algorithm */

1. **for** each black vertex b
2. call *probe-next*(b) s times
3. sort all black vertices b by $solid(b)$ in descending order, breaking ties randomly; let L be the sorted order
4. maintain t = the k-th largest $solid(b)$ of all $b \in B$ in the rest of the algorithm
5. **for** each black vertex b by the ordering in L
6. **repeat**
7. *probe-next*(b)
8. **until** all edges of b have been probed **or** $empty(b) \geq m - t + 1$
9. **return** the k black vertices with the largest degrees (handle ties if necessary)

Fig. 6. Algorithm *sample-sort*

This, however, does not imply that the algorithm needs to have the exact $deg(b)$ and $deg(b')$. It suffices to show that a lower bound of $deg(b)$ is greater than an upper bound of $deg(b')$.

If $b \in B$ does not have an edge with $w \in W$ in G, we say that b has an ***empty edge*** with w; otherwise, b has a ***solid edge*** with w. Hence, $deg(b)$ equals the number of solid edges of b. Moreover, the total number of empty and solid edges of b equals m ($= |W|$). Each time when an edge-probing query is performed, the outcome reveals that the edge is either empty or solid. Denote by $empty(b)$ the number of empty edges of b that have been probed, and similarly, let $solid(b)$ be the number of its solid edges probed. It immediately follows that:

$$solid(b) \leq deg(b) \leq m - empty(b). \tag{8}$$

For each $b \in B$, algorithm A maintains, at all times, an upper bound $m - empty(b)$ of $deg(b)$, as well as a lower bound $solid(b)$. It terminates as soon as it is able to conclude on the final result R based on these bounds, in the way explained earlier.

Algorithm *sample-sort* (SS). Next, we explain our first algorithm. It aims at quickly discovering k black vertices with large degrees. After this is done, let x be the smallest degree of the vertices identified. Then, we can prune any black vertex b once $m - x + 1$ of its empty edges have been found. Apparently, a higher x gives stronger pruning power.

But how do we know which vertices are likely to have large degrees? The idea of sampling naturally kicks in. Specifically, algorithm SS has two phases. The first *sampling phase* randomly probes s edges of every black vertex, where s is a parameter of the algorithm. At the end of this phase, all the black vertices b are sorted in descending order of $solid(b)$. Denote the sorted list as L. As $\frac{m}{s} solid(b)$ is an unbiased estimate of $deg(b)$, L essentially ranks all black vertices in descending order of their estimated degrees.

The second, *refinement phase*, processes the black vertices by their sorted order in L. For each black vertex b, SS keeps probing its hidden edges until all of its edges have been probed (at which point, the exact $deg(b)$ is available) or b can be pruned. To enable pruning, at all times, the algorithm maintains a threshold t, which equals the k-th largest $solid(b')$ of all $b' \in B$ (t may change continuously as more edges are probed). Thus, b is pruned

once $empty(b) \geq m - t + 1$.

The overall algorithm is presented in Figure 6. Its main drawback is the reliance on parameter s, for which careful tuning is needed to obtain good efficiency. This motivates the next algorithm, which does not require any parameter.

Algorithm *switch-on-empty* (SOE). The algorithm works in *rounds*, where each round finds exactly *one* empty edge for every black vertex. Rounds continue until the algorithm is able to conclude the result set R of black vertices. Each round works as follows. For every black vertex b, we keep probing its hidden edges, and stop (i) *as soon as* an empty edge of b is found, or (ii) when b has no more edge to probe. In either case, we switch to another black vertex (hence the name *switch-on-empty*), and repeat the same. The round finishes when all the black vertices in B have been processed like this.

Before starting the next round, the algorithm checks whether some black vertices can be safely put into the result R and thus removed from B. Specifically, a vertex $b \in B$ is added to R if it satisfies two conditions:

(1) All its m edges have been probed.

(2) $empty(b)$ is the lowest among all the vertices still in B (remember that the vertices in R are already removed from B).

To see why, note that Condition 1 implies that we have obtained the exact $deg(b)$, and Condition 2 ensures that $deg(b) = m - empty(b) \geq m - empty(b') \geq deg(b')$ for any $b' \in B, b' \neq b$, namely, b has the largest degree among all vertices in B.

SOE terminates when (i) R has at least k vertices, and (ii) the remaining vertices in B definitely have lower degrees than those in R (namely, for each vertex $b \in B$, we have found at least $m - t + 1$ of its empty edges, where t is the smallest degree of the vertices in R). Figure 7 formally summarizes the algorithm.

LEMMA 1. *SOE returns the k-MCV result correctly.*

PROOF. As mentioned before, a black vertex enters R only if its exact degree is (i) already known, and (ii) guaranteed to be the maximum among the remaining vertices in B. This ensures that vertices are added to R in non-ascending order of their degrees, and that the minimum degree in R is at least the maximum degree in B. Therefore, t becomes the the degree of the k-th most connected vertex in B at the moment $|R|$ first reaches k. After that, R is guaranteed to be a subset of the k-MCV result since no vertex with degree less than t can be appended to R. Finally, R is also a superset of the k-MCV result, because the terminating condition will be triggered only after all vertices with degrees at least t have been removed from B (equivalently, put into R). □

Example. We illustrate SOE using the input graph in Figure 8 where B and W have 2 and 5 vertices, respectively. Assume that $k = 1$ and that the algorithm class considered is the random-probe class $\mathcal{A}_{\mathrm{RAN}}$ (the case of the deterministic-probe class $\mathcal{A}_{\mathrm{DET}}$ is similar). At the beginning, all the edges are hidden; so for each black vertex, SOE initializes an upper bound of $|W| = 5$ on its degree.

Then, SOE executes its rounds, each of which keeps probing a black vertex's hidden edges until encountering an empty edge or the vertex has no more hidden edge. In round 1, for b_1, suppose that SOE probes first its edge with w_2, which turns out to be solid. Hence, the algorithm probes another edge of b_1, for example, its edge with w_5. As the

algorithm *switch-on-empty*

/* for each $b \in B$, $solid(b)$ and $empty(b)$ are dynamically maintained throughout the algorithm */

1. $R = \emptyset$ /* the result set */
2. maintain t = the smallest degree of the vertices in R in the rest of the algorithm ($t = -\infty$ if $|R| < k$)
3. maintain e_{min} = the smallest $empty(b)$ of all vertices b still in B
4. **repeat**
5. *perform-a-round* /* see below */
6. B_{done} = {the vertices in B with no more hidden edge}
7. B_{min} = {the vertices in B_{done} with degree $m - e_{min}$}
8. **if** $B_{min} \neq \emptyset$ and $m - e_{min} \geq t$
9. add B_{min} to R, and remove B_{min} from B
   /* this may change the values of t and e_{min} */
10. **until** all vertices still in B have a degree upper bound smaller than t, namely, $m - e_{min} \leq t - 1$
11. return R

algorithm *perform-a-round*

1. **for** each $b \in B$
2. **repeat**
3. *probe-next*(b)
4. **until** an empty edge is found **or** b has no more hidden edge

Fig. 7. Algorithm *switch-on-empty*

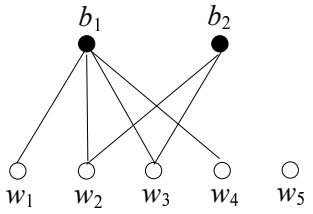


Fig. 8. An example to illustrate SOE

edge is empty, SOE is done with b_1 in this round. For b_2, suppose that SOE first probes its edge with w_3, (since it is solid) then its edge with w_4, and (since an empty edge is found) stops. The first round finishes at this point. No result can be confirmed, because each black vertex still has hidden edges. Nevertheless, the algorithm knows that the degree of each black vertex can be at most 4 because one empty edge has been found for b_1 and b_2, respectively.

In the second round, as all the hidden edges of b_1 are solid, SOE probes all of them before processing the next black vertex. For b_2, suppose that SOE probes (among its hidden edges) its edge with w_1, which is empty. Thus, the algorithm finishes the second round. At this time, SOE sees that $deg(b_1)$ equals 4, and $deg(b_2)$ is at most 3 (as 2 empty edges of b_2 have been identified). Therefore, it terminates by reporting b_1 as the result. □

Remark. Algorithm SOE simultaneously belongs to both the random-probe algorithm class $\mathcal{A}_{\mathrm{RAN}}$ and the deterministic-probe algorithm class $\mathcal{A}_{\mathrm{DET}}$, depending on which version of *probe-next*(b) (Figure 3 or 4) is plugged in. Although the same is true for algorithm SS, it is better suited for $\mathcal{A}_{\mathrm{RAN}}$. The reason is that, in the context of $\mathcal{A}_{\mathrm{DET}}$, the sampling phase can no longer guarantee probing a set of random edges for each black vertex, because the sequence of white vertices in Figure 4 may not be a random sequence.

5. THEORETICAL ANALYSIS OF THE EXACT ALGORITHMS

In this section, we analyze the performance of algorithms SS and SOE. Section 5.1 first establishes their theoretical guarantees in $\mathcal{A}_{\mathrm{RAN}}$, and then, Section 5.2 extends the discussion to $\mathcal{A}_{\mathrm{DET}}$.

5.1 The randomized algorithm class

Let us start with a property of all the algorithms $A \in \mathcal{A}_{\mathrm{RAN}}$. Consider any black vertex $b \in B$. Assume, without loss of generality, that b has ml empty edges in the input graph G, where l is a value between 0 and 1. In other words, b is connected to $m(1-l)$ white vertices in G. Let $Q(u)$ be the expected number of edge-probing queries that A must perform for b, in order to find u empty edges of b. We have:

PROPOSITION 2. $Q(u) = u(m+1)/(ml+1)$.

PROOF. Consider a set of x balls, among which y are black. Keep randomly removing balls from the set without replacement until $z \leq y$ black balls have been removed. The total number of balls that are removed follows the *negative hypergeometric distribution* with expectation $z(x+1)/(y+1)$ [Matuszewski 1962].

Let X be the random variable that equals the number of queries that A must perform on b before seeing u empty edges of b. Then, x, y, z correspond to m, ml, u, respectively. Therefore, the expectation of X, namely $Q(u)$, equals $u(m+1)/(ml+1)$. □

Equipped with the proposition, next we discuss algorithms SS and SOE separately.

Sample-sort. Recall that SS has a parameter s, which specifies the number of edges to probe for each black vertex in the sampling phase. In general, s can be a function of n and m, that is, SS may decide s after obtaining the sizes of B and W.

As shown in the experiments, with a suitable s, SS can be fairly efficient, but such an s appears to heavily depend on the dataset. Because of this, we are interested in knowing whether there is a "universal" choice of s that makes SS instance optimal. A positive answer would allow us to get rid of this parameter. Unfortunately, we ended up proving:

THEOREM 1. *If s is already determined prior to running the first query, SS cannot be instance optimal.*

PROOF. We will find two families of bipartite graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, such that (i) for any sufficiently large n and m satisfying $n > m$, there is a graph $G_1(n,m)$ in $\mathcal{G}_1$ and a graph $G_2(n,m)$ in $\mathcal{G}_2$, both of which have n (m) black (white) vertices, and (ii) they demand conflicting ways to set s so that algorithm SS can be instance optimal. Since (without probing any edge) SS cannot tell whether the input is from $\mathcal{G}_1$ or $\mathcal{G}_2$, it is not able to set s correctly, and thus, fails to be instance optimal. For the above purpose, we focus on $k = 1$. Given a pair of n and m, next we explain how to construct $G_1(n,m)$ and $G_2(n,m)$ respectively.

$G_1(n, m)$ is exactly the graph illustrated in Figure 5, where a unique black vertex has degree m, and the other black vertices all have degree 0. In Section 4, we have shown that algorithm SOE solves the problem with $n+m-1 = O(n)$ queries. As for SS, its sampling phase already probes $O(sn)$ edges; so s must be $O(1)$ if SS needs to be instance optimal. In the sequel, we assume $s \leq \lambda$, where λ is a constant.

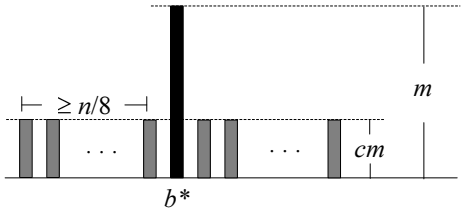


Fig. 9. Illustration of $G_2(n, m)$

$G_2(n, m)$ is such that one black vertex $b^\star$ has degree m, and the other black vertices all have degree cm, where constant $c \in (0, 1)$ will be determined later. Figure 9 illustrates $G_2(n, m)$ by using the height of a column to represent a black vertex's degree. Consider the sampling phase of SS on $G_2(n, m)$. Let S be the set of black vertices $b \in B$ such that *all* the s edges of b probed by SS are solid (notice that $b^\star$ is definitely in S). The choice of c will make sure that $|S| \geq n/4$ with probability at least $1/2$ (later we will argue that such c always exists). Assuming $|S| \geq n/4$, let us look at the refinement phase of SS, where the black vertices b are processed in descending order of $solid(b)$, i.e., how many solid edges of b were found in the sampling phase. Since all vertices in S have the same $solid(b)$, their ordering is random. Hence, with probability at least $1/4$, $n/8$ of the vertices in S rank before $b^\star$. For each such vertex b, SS needs to probe all of its m edges; hence, at least $nm/8$ edges are probed in total. Therefore, the expected cost on $G_2(n, m)$ is at least $(1/4) \cdot (nm/8) = \Omega(nm)$.

The 1-MCV problem on $G_2(n, m)$ can be solved by algorithm SOE with $O(n)$ queries in expectation. Specifically, when SOE terminates, it has found exactly one empty edge of each $b \in B, b \neq b^\star$, plus all the m edges of $b^\star$. By Proposition 2, in expectation, SOE probes $\frac{m+1}{m(1-c)+1} = O(1)$ edges of b. Hence, the expected cost of SOE is $O(n-1+m) = O(n)$, meaning that SS is worse by a factor of $\Omega(m)$.

It remains to show that the c we need always exists. Let X be a random variable that equals the size of S after SS finishes its sampling phase. X follows a Binomial distribution $B(n-1, p)$, where p is the probability that all the s edges probed for a $b \in B, b \neq b^\star$ are solid. More precisely, p is the success probability of the following *sampling-without-replacement* operation: imagine a bag with m balls in which cm are red, and the others blue; we sample s balls from the bag without replacement, and call it a *success* if all of them are red. When m is large enough, p can be approximated with arbitrarily small error by the success probability c^s of the corresponding *sampling-with-replacement* operation. So, conservatively, assume $p \geq c^s - \epsilon \geq c^\lambda - \epsilon$, where $\epsilon > 0$ is an arbitrarily small constant. By Hoeffding's inequality[1], $X \geq (n-1)/2 > n/4$ with probability at least $1 - \exp(-2(n-1)(p-0.5)^2)$, which is at least 0.5 if $p \geq (\frac{\ln\sqrt{2}}{n-1})^{0.5} + 0.5$. To ensure

[1] In general, if X obeys $B(n, p)$, then $\Pr[X \leq x] \leq \exp(-2(np-x)^2/n)$ for all $x \leq np$.

this, it suffices to guarantee $c^\lambda \geq (\frac{\ln\sqrt{2}}{n-1})^{0.5} + 0.5 + \epsilon$. Hence, for large n, we can set c to $0.6^{1/\lambda}$.

We have shown, for a specific λ, there is always a c that makes SS worse than SOE by a factor of $\Omega(m)$ on $G_2(n, m)$ (implying SS cannot be instance optimal). To break the argument, λ cannot exist which, by the definition of λ, means that s cannot be a constant. This, however, conflicts with the requirement of s on $G_1(n, m)$. □

The theorem indicates that, while sampling is a natural idea to attack the k-MCV problem, it is non-trivial to decide the proper sample size. In particular, straightforward strategies such as "sample a certain percentage of the edges of each $b \in B$" does not work. In other words, the correct sample size needs to be chosen *adaptively*, based on the degree distributions of the black vertices. This is consistent with the design of algorithm SOE, since it proceeds by continuously monitoring the edges found on all the black vertices.

Switch-on-empty. In the sequel, we denote by R the set of black vertices in the result. Let $t^\star$ be the lowest degree of the vertices in R, or formally:

$$t^\star = \min_{b \in R} deg(b) \tag{9}$$

Denote by $R_{tail} \subseteq R$ the set of vertices in R having degree $t^\star$. Let $k^\star = |R|$. Apparently, $k^\star \geq k$; furthermore, if $k^\star > k$, then R_{tail} must contain at least $k^\star - k + 1$ vertices.

We first point out two more properties of all algorithms $A \in \mathcal{A}_{\mathrm{RAN}}$. The first one concerns the status of A when it finishes. For each $b \in B$, let $solid_A(b)$ and $empty_A(b)$ be the numbers of solid and empty edges that A has found on b at its termination, respectively. Denote by t_A the minimum $solid_A(b)$ of all vertices $b \in R$, namely, $t_A = \min_{b \in R} solid_A(b)$. We have:

LEMMA 2. *At termination, for each non-result black vertex $b \in B \setminus R$, it holds that $empty_A(b) \geq m - t_A + 1$.*

PROOF. Obvious because otherwise A cannot have concluded that b has a smaller degree than the vertices in R. □

The second property concerns the scenario where $k^\star > k$:

LEMMA 3. *If $k^\star > k$, at termination, A has probed all the m edges of at least $k^\star - k + 1$ black vertices in R_{tail}.*

PROOF. Let $S \subseteq R_{tail}$ be the set of vertices in R_{tail} such that, for any black vertex in S, algorithm A did *not* probe all of its edges. Let $g = |R_{tail}| - (k^\star - k)$. Note that g is always positive because $|R_{tail}|$ is at least $k^\star - k + 1$, as mentioned earlier.

A crucial observation is that $|S|$ must be at most g. Otherwise, assume $|S| \geq g$; then consider any g vertices, say b_1, ..., b_g, in S, and use S' to denote the set of those vertices. Since each b_i has at least 1 hidden edge, it is possible that all those g hidden edges (one for each b_i) turn out to be solid, and at the same time, the black vertices in $S \setminus S'$ have no more hidden solid edge. In this case, $R_{tail} \setminus S'$ must be eliminated from the result, which contradicts the fact that A was able to terminate safely.

Therefore, A must have probed all the m edges of at least $|R_{tail}| - |S| \geq |R_{tail}| - (g - 1) = k^\star - k + 1$ vertices. □

The next lemma states a property of algorithm SOE:

LEMMA 4. *SOE probes all the m edges of each vertex in R. For each vertex $b \in B \setminus R$, it finds exactly $m - t^\star + 1$ of its empty edges. Furthermore, the last edge of b probed by SOE is empty.*

PROOF. The lemma follows directly from the algorithm description in Figure 7. □

Let us label the $n - k^\star$ black vertices *not* in the result R as

$$b_{k^\star+1}, b_{k^\star+2}, ..., b_n,$$

respectively (ordering unimportant). For each $i \in [k^\star + 1, n]$, let

$$l_i = 1 - deg(b_i)/m.$$

Equivalently, ml_i is the number of empty edges of b_i. Furthermore, define $Q_i(u)$ as the expected number of edges of b_i that must be probed by an algorithm in $\mathcal{A}_{\mathrm{RAN}}$, in order to find u empty edges of b_i. $Q_i(u)$ is calculated as in Proposition 2. By Lemma 4, the expected cost of SOE can be written as

$$cost(SOE, G) = mk^\star + \sum_{i=k^\star+1}^{n} Q_i(m - t^\star + 1). \tag{10}$$

Denote by A_{opt} the fastest algorithm in $\mathcal{A}_{\mathrm{RAN}}$ for solving the k-MCV problem on the input graph G. Namely,

$$A_{opt} = \arg\min_{A \in \mathcal{A}_{\mathrm{RAN}}} \{cost(A, G)\}.$$

Next, we proceed to show that SOE is optimal up to a small factor $1 + \frac{k}{n-k}$, by discussing cases $k^\star = k$ and $k^\star > k$ separately. For $k^\star = k$, we have:

LEMMA 5. *If $k^\star = k$, $cost(SOE, G)/cost(A_{opt}, G) \leq 1 + \frac{k}{n-k}$.*

PROOF. Define a random variable:

$$t_{opt} = \min_{b \in R} solid_{opt}(b). \tag{11}$$

where, for each $b \in R$, $solid_{opt}(b)$ is the number of solid edges of $b \in R$ found by A_{opt} at termination. In the sequel, we fix an integer x, and focus on the event

$$\Xi_x : t_{opt} = x,$$

i.e., the event that A_{opt} terminates with $t_{opt} = x$. As $solid_{opt}(b) \leq deg(b)$ for each $b \in R$, it holds that:

$$x = \min_{b \in R} solid_{opt}(b) \leq \min_{b \in R} deg(b) = t^\star$$

Define function $C(x)$ to be the expected cost of A_{opt} conditioned on Ξ_x. The rest of the proof will show that $r = cost(SOE, G)/C(x) \leq 1 + k/(n - k)$ for any x. This, together with $cost(A_{opt}, G) = \sum_x C(x) \cdot \Pr[\Xi_x]$, will establish the lemma.

By Lemma 2, A_{opt} needs to find at least $m - x + 1$ empty edges of each black vertex b_i ($k^\star + 1 \leq i \leq n$), meaning that it is expected to perform $Q_i(m - x + 1)$ probes on b_i. For every other black vertex, A_{opt} has to discover at least x solid edges. Therefore,

$$C(x) \geq xk + \sum_{i=k^\star+1}^{n} Q_i(m - x + 1).$$

Combining the above with (10), we know

$$\begin{aligned} r &\leq \frac{mk^\star + \sum_{i=k^\star+1}^{n} Q_i(m - t^\star + 1)}{xk + \sum_{i=k^\star+1}^{n} Q_i(m - x + 1)} \Rightarrow \text{(applying } x \leq t^\star\text{)} \\ r &\leq \frac{mk^\star + \sum_{i=k^\star+1}^{n} Q_i(m - x + 1)}{xk + \sum_{i=k^\star+1}^{n} Q_i(m - x + 1)} \Rightarrow \\ r - 1 &\leq \frac{(m-x)k^\star}{xk^\star + \sum_{i=k^\star+1}^{n} Q_i(m - x + 1)}. \end{aligned}$$

By Proposition 2, $Q_i(m - x + 1) = (m - x + 1)\frac{m+1}{ml_i+1}$. Hence:

$$r - 1 < \frac{(m-x)k^\star}{xk^\star + a(m-x)}$$

where

$$a = \sum_{i=k^\star+1}^{n} \frac{m+1}{ml_i + 1}. \tag{12}$$

If $x = m$, then $r = 1$, trivially satisfying $r \leq 1 + k/(n-k)$. For $x < m$, equipped with $a \geq n - k^\star = n - k$, we have

$$r - 1 \leq \frac{(m-x)k}{(n-k)(m-x)} = \frac{k}{n-k}.$$

This completes the proof. □

The following lemma covers the other case $k^\star > k$:

LEMMA 6. *If $k^\star > k$, $cost(SOE, G)/cost(A_{opt}, G) \leq 1 + \frac{k}{n-k}$.*

PROOF. According to Lemma 3, at termination, A_{opt} must have probed all the edges of at least $k^\star - k + 1 > 1$ vertex in R_{tail}. Hence, t_{opt}, as defined in (11), equals $t^\star$. Consequently, A_{opt} probes

—$Q_i(m - t^\star + 1)$ edges in expectation for each non-result vertex b_i ($k^\star + 1 \leq i \leq n$), by Lemma 2 and the definition of Q_i;

—all the m edges of $k^\star - k + 1$ result vertices, by Lemma 3;

—at least $t^\star$ solid edges for each of the remaining $k - 1$ result vertices, in order to confirm that their degrees are at least $t^\star$.

Therefore,

$$cost(A_{opt}, G) \geq t^\star(k-1) + m(k^\star - k + 1) + \sum_{i=k^\star+1}^{n} Q_i(m - t^\star + 1)$$

Set $r = cost(SOE, G)/cost(A_{opt}, G)$. Combining the above formula with (10) gives:

$$\begin{aligned} r &\leq \frac{mk^\star + \sum_{i=k^\star+1}^{n} Q_i(m - t^\star + 1)}{t^\star(k-1) + m(k^\star - k + 1) + \sum_{i=k^\star+1}^{n} Q_i(m - t^\star + 1)} \\ \text{(as } t^\star \leq m\text{)} &\leq \frac{mk^\star + \sum_{i=k^\star+1}^{n} Q_i(m - t^\star + 1)}{mk^\star + \sum_{i=k^\star+1}^{n} Q_i(m - t^\star + 1) - k(m - t^\star)} \end{aligned}$$

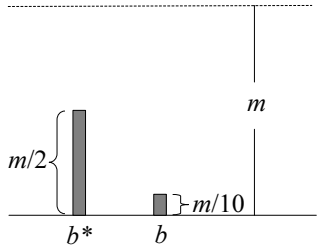


Fig. 10. Why SOE is not strictly optimal

Hence, applying Proposition 2, we have:

$$r - 1 \leq \frac{k(m - t^\star)}{mk^\star + a(m - t^\star + 1) - k(m - t^\star)}$$

where a is given in (12). Again, if $t^\star = m$, then $r = 1 < 1 + k/(n - k)$. Otherwise, knowing $a \geq n - k^\star$, we derive:

$$\begin{aligned} r - 1 &\leq \frac{k(m - t^\star)}{mk^\star + (n - k^\star)(m - t^\star) - k(m - t^\star)} \\ &\leq \frac{k(m - t^\star)}{n(m - t^\star) - k(m - t^\star)} = \frac{k}{n - k}. \end{aligned}$$

This establishes the lemma. □

Combining the above two lemmas, we have proved the following theorem:

THEOREM 2. *The expected cost of SOE is at most* $r \cdot cost(A_{opt}, G)$, *where* $r = 1 + \frac{k}{n-k}$.

There are two interesting corollaries:

—For any $k \leq n/2$, the value of r is *always lower than 2*, that is, SOE is instance optimal.

—When $k = O(1)$, $r = 1 + O(1/n)$, namely, SOE is *nearly as fast as the optimal algorithm* in finding the *top few* (e.g., 10) black vertices having the maximum degrees.

We close the subsection with a note on why SOE is not *strictly* better than all other algorithms in $\mathcal{A}_{\rm RAN}$. Imagine a simple G whose B has only 2 vertices $b^\star$ and b with degrees $m/2$ and $m/10$, respectively. Figure 10 illustrates this by using the height of a column to represent the degree of a node. Consider the 1-MCV problem on such G. By Lemma 4, SOE probes all m edges of $b^\star$, and enough edges of b until seeing $1 + m/2$ empty edges. Hence, by Proposition 2, the expected cost of SOE is $m + (1 + \frac{1}{2}m)(m + 1)/(\frac{9}{10}m + 1) \approx 1.56m$. An alternative solution is to probe all edges of b, and enough edges of $b^\star$ until seeing $1 + m/10$ solid edges. This strategy's expected cost is $m + (1 + \frac{1}{10}m)(m + 1)/(1 + \frac{1}{2}m) \approx 1.2m$.

5.2 The deterministic algorithm class

Next, we extend the analysis of the previous subsection to the algorithm class $\mathcal{A}_{\rm DET}$. We focus on only SOE because the instance optimality of SS in $\mathcal{A}_{\rm DET}$ can be disproved using an argument similar to, but much simpler than, the proof of Theorem 1. For $\mathcal{A}_{\rm DET}$, Proposition 2 obviously is not applicable; Lemmas 2-4, however, are still correct. Define

A_{opt} as the fastest algorithm in $\mathcal{A}_{\rm DET}$ for solving the k-MCV problem on the input G. Namely:

$$A_{opt} = \arg\min_{A \in \mathcal{A}_{\rm DET}} \{cost(A, G)\}.$$

We first give a theorem that is the counterpart of Theorem 2.

THEOREM 3. *The cost of SOE is at most* $(1 + \frac{k}{n-k}) \cdot cost(A_{opt}, G)$.

PROOF. The proof is similar to that of Theorem 2 (called the *old proof* in the sequel). Refer to the sequence $(w_1, w_2, ..., w_m)$ in Figure 3 as the *probing sequence*. Let $k^\star$, $t^\star$, b_i $(k^\star + 1 \le i \le n)$ retain their meanings in the old proof.

Let τ_i $(k^\star+1 \le i \le n)$ be the number of edges of b_i that SOE has probed at termination. τ_i equals the position of the $(m - t^\star + 1)$-th white vertex (in the probing sequence) that has an empty edge with b_i. By Lemma 4, $cost(SOE, G) = mk^\star + \sum_{i=k^\star+1}^{n} \tau_i$. Define t_{opt}, $solid_{opt}(b)$, $C(x)$ in the same way as in the old proof. Let $\tau_i^\star$ be the number of edges that A_{opt} probes for b_i, conditioned on $t_{opt} = x$. Since $x = t_{opt} \le t^\star$, by Lemma 2, A_{opt} must have seen at least $m - x + 1 \ge m - t^\star + 1$ empty edges of b_i. In other words, A_{opt} probes all the edges of b_i that SOE needs to probe; hence:

$$\tau_i \le \tau_i^\star. \tag{13}$$

Set $r = cost(SOE, G)/C(x)$ and $a = \sum_{i=k^\star+1}^{n} \tau_i^\star$. As explained earlier, A_{opt} probes at least $m - x + 1$ edges for each of $b_{k^\star+1}, ..., b_n$, indicating

$$a \ge (n - k^\star)(m - x + 1). \tag{14}$$

Next, we establish the counterpart of Lemma 5. When $k^\star = k$, it holds that $C(x) \ge xk^\star + \sum_{i=k^\star+1}^{n} \tau_i^\star$. Hence

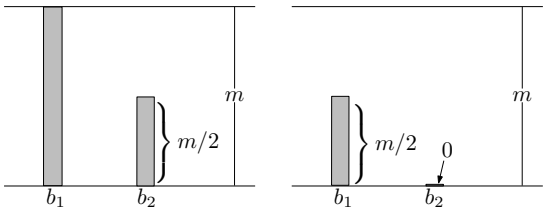
$$r \le \frac{mk^\star + \sum_{i=k^\star+1}^{n} \tau_i}{xk^\star + \sum_{i=k^\star+1}^{n} \tau_i^\star} \le \frac{mk^\star + a}{xk^\star + a}.$$

where the last inequality used (13). If $x = m$, then $r = 1$, and the lemma is trivially true. For $x < m$:

$$\begin{aligned} r - 1 \le \frac{(m-x)k^\star}{xk^\star + a} &\le \frac{(m-x)k^\star}{a} \\ \text{(by (14))} &\le \frac{(m-x)k^\star}{(n-k^\star)(m-x)} = \frac{k^\star}{n-k^\star} = \frac{k}{n-k}. \end{aligned}$$

We now prove the counterpart of Lemma 6. When $k^\star > k$, an argument similar to that of Lemma 6 shows $C(x) \ge t^\star(k-1) + m(k^\star - k + 1) + \sum_{i=k^\star+1}^{n} \tau_i^\star$. Thus,

$$\begin{aligned} r &\le \frac{mk^\star + \sum_{i=k^\star+1}^{n} \tau_i}{t^\star(k-1) + m(k^\star - k + 1) + \sum_{i=k^\star+1}^{n} \tau_i^\star} \\ \text{(by (13) and } t^* \le m) &\le \frac{mk^\star + a}{mk^\star + a - k(m - t^\star)} \Rightarrow \\ r - 1 &\le \frac{k(m - t^\star)}{mk^\star + a - k(m - t^\star)} \end{aligned} \tag{15}$$



(a) G_3 (b) G_4

Fig. 11. No algorithm is strictly optimal in $\mathcal{A}_{\rm DET}$

If $t^\star = m$, then $r = 1$, in which case the lemma is trivially true. For $t^\star < m$, by (14) and (15), we have:

$$\begin{aligned} r - 1 &\leq \frac{k(m-t^\star)}{mk^\star + (n-k^\star)(m-t^\star+1) - k(m-t^\star)} \\ &\leq \frac{k(m-t^\star)}{n(m-t^\star) - k(m-t^\star)} = \frac{k}{n-k}, \end{aligned}$$

which completes the proof. □

The same conclusions in $\mathcal{A}_{\rm RAN}$ can be drawn about SOE in $\mathcal{A}_{\rm DET}$. Specifically, for $k \leq n/2$, SOE is also instance optimal in $\mathcal{A}_{\rm DET}$. Furthermore, when $k = O(1)$, SOE can be more expensive than the optimal algorithm in $\mathcal{A}_{\rm DET}$ only by a factor of $1 + O(1/n)$.

Absence of a strictly optimal algorithm. We conclude the section by proving that *no algorithm in $\mathcal{A}_{\rm DET}$ can be strictly optimal*. We will use two input graphs G_3 and G_4 as illustrated in Figure 11. For G_3, order the white vertices so that the first two *probe-next*(b_2) return *empty* and *solid*, respectively. Similarly, for G_4, impose an ordering for the first two *probe-next*(b_1) to return *solid* and *empty*, respectively. Observe that, for each of G_3 and G_4, there is an algorithm that can settle the 1-MCV problem with $m + 1$ queries. Specifically, to achieve this for G_3, the algorithm can probe a single edge of b_2 and all the edges of b_1, whereas the algorithm for G_4 can probe a single edge of b_1 and all the edges of b_2. We will prove, however, that no algorithm can guarantee finishing with at most $m + 1$ queries on *both* graphs, which essentially means that no algorithm is optimal in all cases.

Following the notations before, given an algorithm $A \in \mathcal{A}_{\rm DET}$ and a graph G, we denote by $solid_A(b)$ the number of solid edges that A probes on a black vertex b of G, and by $empty_A(b)$ the corresponding number on the empty edges of b. We have:

LEMMA 7. *For any algorithm $A \in \mathcal{A}_{\rm DET}$, $\max\{cost(A, G_3), cost(A, G_4)\} > m + 1$.*

PROOF. We will use an adversary argument, the rationale of which is not to permit A to distinguish between G_3 and G_4 until we are sure that it must probe at least $m + 2$ edges. First, note that, to conclude on $deg(b_1) > deg(b_2)$, A needs to show $solid_A(b_1) > m - empty_A(b_2)$, or equivalently:

$$solid_A(b_1) + empty_A(b_2) \geq m + 1. \tag{16}$$

Hence, if the input is G_3 (G_4) and two edges of b_2 (b_1) have been probed, the cost of A must be at least $m + 2$. To see this for G_3, recall that one of the first two edges probed on b_2 is solid. This edge is not counted by the left hand side of (16), thus making the total cost at least $m + 2$. The reason for G_4 is similar.

Next, we describe the strategy that the adversary follows to force A to probe at least $m+2$ edges. Let b_i be the first vertex selected by *pick-black*. Since the first *probe-next* of b_1 (b_2) must be *solid* (*empty*) no matter the input graph is G_3 or G_4, A cannot distinguish between the two graphs after the first query. Denote by b_j the vertex chosen by the second *pick-black*. We enumerate all the possible cases:

—$b_i = b_j = b_1$: We fix the input to be G_4. By the earlier discussion, A costs at least $m+2$ as it has probed two edges of b_1.

—$b_i = b_j = b_2$: We fix the input as G_3. A costs at least $m+2$ as it has probed two edges of b_2.

—$b_i \neq b_j$: In this case, one solid edge of b_1 and one empty edge of b_2 are found. The algorithm is still unable to decide whether the input graph is G_3 or G_4. We then fix the input according to the vertex b_z chosen by the third *pick-black*. If $b_z = b_1$, let the input be G_4; if $b_z = b_2$, let the input be G_3. In either case, (again, by our earlier discussion) A requires at least $m+2$ queries.

This establishes the lemma. □

6. APPROXIMATE ALGORITHMS AND THEIR ANALYSIS

We proceed to study the ϵ-approximate version of the k-MCV problem. Section 6.1 first presents an algorithm for solving the problem when $k = 1$, and establishes its performance guarantees. Then, Section 6.2 extends our solution and analysis to general $k > 1$. Given a constant $\delta \in (0, 1)$, our algorithms succeed with probability at least $1 - \delta$ and guarantee good efficiency in expectation.

6.1 1-MCV

Algorithm. The basic component of our method is a procedure called *naive-sampling* (NS) which, as given in Figure 12, is similar to the SS algorithm in Section 4. NS is given a parameter p, which is used to determine the number s of edges probed for each black vertex (Line 1). Each of these edges is randomly sampled from all the possible edges of b. We perform the sampling in a *with-replacement* manner, namely, each edge is chosen independently of the previous edges probed. Occasionally, we may waste some work by probing the same edge more than once, but allowing such redundancy facilitates the analysis considerably, as will be clear later. NS returns the black vertex having the largest number of solid edges sampled.

Our algorithm for $k = 1$, called AMCV (see Figure 12), invokes NS repetitively with doubly decreasing p. Specifically, the first invocation uses $p = 1$, whereas every subsequent invocation halves the previous p. Assume that, in the current invocation, NS returns a black vertex b. AMCV terminates with b as the result if $solid(b)$ is large enough (see Line 3); otherwise, another invocation is performed.

Analysis. Next, we analyze the behavior of AMCV, and by doing so, reveal the rationales behind its design. For each black vertex b, define:

$$p(b) = deg(b)/m. \tag{17}$$

Denote by $b^\star$ the black vertex with the maximum degree. Set $t^\star = deg(b^\star)$ and $p^\star = p(b^\star)$. The next lemma shows that, if the input parameter p of NS is set to $p^\star$, then NS returns a correct answer with high probability:

algorithm *naive-sampling*(p)

1. $s = \frac{12}{p}\frac{1}{\epsilon^2}\ln\frac{3n}{\delta}$
2. **for** each black vertex b
3. sample with replacement s edges of b
4. $solid(b)$ = the number of solid edges of b sampled
   (counting the same edge once more each time it is sampled)
5. **return** $(b_{ret}, solid(b_{ret}))$, where b_{ret} is the black vertex b with the largest $solid(b)$
   (breaking ties arbitrarily)

algorithm *AMCV*

1. **for** $p = 1, 1/2, 1/4, 1/8, ...$
2. $(b, solid(b))$ = *naive-sampling*(p)
3. **if** $solid(b) \geq 2ps$ **then return** b
   /* s is given at Line 1 of *naive-sampling* */

Fig. 12. Algorithm for solving the ϵ-approximate 1-MCV problem

LEMMA 8. *When executed on $p = p^\star$, NS returns a correct answer for the ϵ-approximate 1-MCV problem with probability at least $1 - \delta/3$.*

PROOF. Consider the following two conditions:

(1) $solid(b^\star) > (1 - \epsilon/2)sp^\star$.

(2) for every b such that $deg(b) < (1 - \epsilon)t^\star$ (that is, b cannot be used as an answer), $solid(b) < (1 - \epsilon/2)sp^\star$.

If both conditions are satisfied, NS returns a correct result (for the approximate 1-MCV problem) because for any vertex b that is an illegal result, it must hold that $solid(b) < (1-\epsilon/2)sp^\star < solid(b^\star)$, meaning that b cannot be selected by Line 5 of algorithm *naive-sampling* (Figure 12). Next, we show that the two conditions hold simultaneously with high probability.

For any black vertex b, $solid(b)$ follows a binomial distribution, measuring the number of successes in s trials, each of which succeeds with probability $p(b)$. Hence, $solid(b)$ has expectation $sp(b)$. By setting $\alpha = \epsilon/2$ and $p = p^\star$ in (7), we know that the probability for Condition 1 to fail is bounded above by $exp(-sp^\star\epsilon^2/12)$ which equals $\delta/(3n)$ given our choice of s.

In the rest of the proof, consider b as a vertex described in Condition 2. Set $\alpha = \frac{p^\star}{p(b)}(1 - \epsilon/2) - 1$ to ensure $(1 - \epsilon/2)p^\star = (1 + \alpha)p(b)$. Thus, $\Pr[solid(b) \geq (1 - \epsilon/2)sp^\star] = \Pr[solid(b) \geq (1 + \alpha)sp(b)]$. We then distinguish two possibilities:

—If $\alpha \geq 1$, apply (5) with $p = p(b)$, which gives:

$$\Pr[solid(b) \geq (1 + \alpha)sp(b)] \leq exp(-(1 - \epsilon/2)sp^\star/6) < \delta/(3n),$$

where the last inequality used the fact that $1 - \epsilon/2 > 1/2$.

—If $\alpha < 1$, apply (4) with $p = p(b)$, which gives:

$$\Pr[solid(b) \geq (1 + \alpha)sp(b)] \leq exp(-sp(b)\alpha^2/3) = exp(-sp^\star\beta/3) \qquad (18)$$

where $\beta = (1-\epsilon/2)\alpha^2/(1+\alpha)$. Note that $deg(b) < (1-\epsilon)t^\star$ implies that

$$\alpha > (1-\epsilon/2)/(1-\epsilon) - 1 = \epsilon/(2-2\epsilon).$$

As β monotonically increases with α when $\alpha > 0$, it holds that

$$\beta > \frac{(1-\epsilon/2)(\epsilon/(2-2\epsilon))^2}{1+\epsilon/(2-2\epsilon)} = \frac{\epsilon^2}{4-4\epsilon} > \epsilon^2/4.$$

Plugging this into (18) shows that $\Pr[solid(b) \geq (1+\alpha)sp(b)] \leq \exp(-sp^\star\epsilon^2/12) = \delta/(3n)$.

As there can be at most $n-1$ such vertices b, by the union bound (a.k.a., Boole's inequality), the probability that at least one such b satisfies $solid(b) \geq (1-\epsilon/2)sp^\star$ is at most $\frac{n-1}{n}\frac{\delta}{3}$, which is also the probability for Condition 2 to fail.

Again, by the union bound, the probability that either Condition 1 or 2 fails is bounded above by $\delta/3$. Hence, they hold at the same time with probability at least $1-\delta/3$. □

The previous lemma suggests that, if $p^\star$ was known in advance, we could easily settle the ϵ-approximation 1-MCV problem by NS. Of course, in reality $p^\star$ is not necessarily available. AMCV deals with this by using p to approach $p^\star$ gradually. Even without knowing $p^\star$, we still hope that AMCV can terminate with a correct answer when p has eventually fallen into the range $[p^\star/8, p^\star]$. The reasons are two-fold. First, using a $p \leq p^\star$ essentially tells NS to sample more edges than necessary, and hence, guarantees at least the same success probability as in Lemma 8. Second, ensuring that p is not much smaller than $p^\star$ (we choose $p \geq p^\star/8$) prevents NS from sampling excessively, so that we can still control the overall cost to be at most $O(1)$ times greater than the cost of running NS with $p^\star$. The next lemma shows that our hope as described earlier will come true with high probability.

LEMMA 9. *With probability at least $1-\delta$, both of the following happen:*

(*1*) *AMCV terminates with a correct answer for the ϵ-approximate 1-MCV problem;*

(*2*) *at termination, $p \in [p^\star/8, p^\star]$.*

PROOF. The proof considers $n \geq 2$ because the lemma is trivially correct for $n = 1$. If either of the two conditions stated in the lemma is violated, exactly one of the following events must have occurred:

—*Premature*: AMCV terminates when $p > p^\star$.

—*Overdue*: AMCV does not terminate after invoking NS with a $p < p^\star/4$.

—*Wrong-result*: AMCV terminates when $p \in [p^\star/8, p^\star]$, but returns an incorrect result.

We will show that with high probability, none of these events occurs. The same argument in the proof of Lemma 8 can be used to show that *Wrong-result* happens with probability at most $\delta/3$. Notice that for $p < p^\star$, the value of s is even greater than that in the proof of Lemma 8, i.e., we are using more samples than needed to guarantee Conditions 1 and 2. Next, we show that the same is true for both *Premature* and *Overdue*, which will complete the proof with the union bound.

Bounding the probability of Premature. Let us focus on a single invocation of NS. Denote by ν the ratio between the current p and $p^\star$, namely, $\nu = p/p^\star$. Let b_{ret} be the vertex

returned by NS, and $solid(b_{ret})$ the number of solid edges of b_{ret} found in this invocation. We will prove

$$\Pr[solid(b_{ret}) \geq 2sp] \leq (\delta/3)/(2\nu) \tag{19}$$

which is equivalent to saying that AMCV terminates after this invocation with probability at most $(\delta/3)/(2\nu)$. We will give a stronger fact that, for each black vertex b, it holds that

$$\Pr[solid(b) \geq 2sp] \leq (\delta/3)/(2n\nu) \tag{20}$$

which validates (19) with the union bound.

To prove (20), set $\alpha = 2\frac{p}{p(b)} - 1$, which makes $(1+\alpha)p(b) = 2p$. Applying (5) and (6) with this α yields, respectively:

$$\Pr[solid(b) \geq 2sp] \leq \exp(-2sp/6) \tag{21}$$

$$\Pr[solid(b) \geq 2sp] \leq \left(\frac{e}{2p/p(b)}\right)^{2sp} \leq \left(\frac{e}{2\nu}\right)^{2sp} \tag{22}$$

where the last inequality used the fact that $p/p(b) \geq \nu$. Our choice of s leads to

$$\exp(-2sp/6) = ((\delta/3)/n)^{4/\epsilon^2} \leq (\delta/3)/4n$$

where the last inequality is true for any $n \geq 2$. This, together with (21), proves (20) for $\nu \leq 2$. On the other hand,

$$(e/2\nu)^{2sp} = (2/\nu)^{2sp}(e/4)^{2sp} \leq (2/\nu)^{2sp}\exp(-2sp/6) \leq (2/\nu)((\delta/3)/4n)$$

where the first inequality used the fact that $e/4 < e^{-1/6}$. The above, when combined with (22), proves (20) for $\nu > 2$.

Now that we have (19), the probability of *Premature* can be bounded above by the sum of the $(\delta/3)/(2\nu)$ of all $p > p^\star$ deployed by AMCV to invoke NS. Let p_{min} be the smallest of those p; and set $\nu_{min} = p_{min}/p^\star$. Thus, the probability of *Premature* is at most

$$\frac{\delta/3}{2\nu_{min}} + \frac{\delta/3}{2\nu_{min}\cdot 2} + \frac{\delta/3}{2\nu_{min}\cdot 4} + ... \leq \frac{\delta/3}{\nu_{min}} \leq \delta/3.$$

Bounding the probability of Overdue. It suffices to prove that, when invoked with a $p < p^\star/4$, NS fails to terminate with probability at most $\delta/3$. In fact, if NS does not terminate, it must be that $solid(b^\star) \leq 2sp \leq sp^\star/2 \leq (1-\epsilon/2)sp^\star$. The probability of *Overdue* does not exceed

$$\Pr[solid(b^\star) \leq 2sp] \leq \Pr[solid(b^\star) \leq (1-\epsilon/2)p^\star] \leq (\delta/3)/n$$

where the last inequality has been established in the proof of Lemma 8. □

Now it remains to bound the running time of AMCV.

LEMMA 10. *AMCV probes $O(\frac{1}{\epsilon^2}\frac{nm}{t^\star}\log\frac{n}{\delta})$ edges in expectation.*

PROOF. It is easy to see that the cost of NS with input parameter p is $O(sn) = O(\frac{n}{p}\frac{1}{\epsilon^2}\log\frac{n}{\delta})$. Note that this cost is proportional to $1/p$. Starting from the second invocation of NS, p is half of the p of the previous invocation. Hence, the cost of an invocation doubles each

time. Until p drops below $p^\star/8$, the total cost spent on NS is bounded by that of the last invocation, which in turn is bounded above by $O(\frac{n}{p^\star}\frac{1}{\epsilon^2}\log\frac{n}{\delta}) = O(\frac{1}{\epsilon^2}\frac{nm}{t^\star}\log\frac{n}{\delta})$.

We use the term *late phase* to refer to the execution of AMCV with $p < p^\star/8$. Let λ be the cost of the entire late phase. Next, we bound the expectation of λ. Let λ_i ($i \geq 1$) be the cost of the i-th invocation of NS in the late phase. It follows that $\lambda_1 = O(\frac{n}{p^\star}\frac{1}{\epsilon^2}\log\frac{n}{\delta})$, and $\lambda_i = 2\lambda_{i-1}$ for $i \geq 2$. As shown in the proof of Lemma 9, the probability that AMCV needs to go into the late phase is at most $\delta/3$ because an *Overdue* event must have occurred. Furthermore, if NS needs to be executed i times in the late phase, it means that the previous $i-1$ invocations in the late phase have all generated an *Overdue* event, respectively. In other words, the i-th invocation of NS in the late phase occurs with probability at most $(\delta/3)^i$. It follows that:

$$E[\lambda] \;\leq\; \lambda_1\frac{\delta}{3} + \lambda_2\left(\frac{\delta}{3}\right)^2 + \lambda_3\left(\frac{\delta}{3}\right)^3 + ... = \lambda_1\frac{\delta}{3}\left(1 + \frac{2\delta}{3} + \left(\frac{2\delta}{3}\right)^2 + ...\right)$$

which is bounded by $O(\lambda_1\delta/3)$. □

So we conclude:

THEOREM 4. *For any $\delta \in (0,1)$, there is an algorithm that solves the ϵ-approximate 1-MCV problem with probability at least $1-\delta$, and probes $O(\frac{1}{\epsilon^2}\frac{nm}{t^\star}\log\frac{n}{\delta})$ edges in expectation, where $t^\star$ is the maximum degree of the black vertices.*

6.2 k-MCV

Algorithm. The algorithm in Figure 12 can be easily modified to support $k > 1$:

—In NS (*naive-sampling*), Line 1 sets $s = \frac{24}{p}\frac{1}{\epsilon^2}\ln\frac{3n}{\delta}$, namely, twice as large as the original value.

—For each black vertex b, as before, let $solid(b)$ be the number of solid edges of b found. NS returns the k vertices b having the greatest $solid(b)$ (breaking ties arbitrarily), together with their $solid(b)$ values.

—In AMCV, let $b_1, ..., b_k$ be the vertices obtained from NS at Line 2, sorted in such a way that $solid(b_i) \geq solid(b_j)$ for $1 \leq i < j \leq k$. At Line 3, AMCV returns these vertices if $solid(b_k) \geq 2ps$.

In the sequel, all occurrences of NS and AMCV refer to the above adapted algorithms, which capture the ones in Figure 12 as special cases.

Analysis. Denote by $b_1^\star$, ..., $b_k^\star$ the k black vertices with the maximum degrees (ties broken arbitrarily) such that $deg(b_i^\star) \geq deg(b_j^\star)$ for $1 \leq i < j \leq k$. Define $t_i^\star = deg(b_i^\star)$ and $p_i^\star = p(b_i^\star)$, where function $p(.)$ is as given in (17). The next two lemmas are the counterparts of Lemmas 8 and 9. As the new proofs are based on the ideas already clarified in Section 6.1, we will focus on explaining only the differences.

LEMMA 11. *When executed on $p = p_k^\star$, NS returns a correct answer for the ϵ-approximate k-MCV problem with probability at least $1-\delta/3$.*

PROOF. Given an $i \leq k$ and a black vertex b, we say that b *fails* on i in either of the following cases:

—$deg(b) \geq t_i^\star$ whereas $solid(b) \leq (1-\epsilon/2)sp_i^\star$;

—$deg(b) < (1-\epsilon)t_i^\star$ whereas $solid(b) \geq (1-\epsilon/2)sp_i^\star$.

Observe that NS returns a correct answer if no vertex fails on any $i \leq k$.

With an argument similar to the proof of Lemma 8, we can show that each vertex b fails on an $i \leq k$ with probability at most $((\delta/3)/n)^2$. Here, the square comes from the fact that we are using an s twice larger than that in Figure 12. Hence, with probability at least $1 - nk((\delta/3)/n)^2 > 1 - \delta/3$, no vertex fails on any $i \leq k$. □

LEMMA 12. *With probability at least* $1-\delta$*, both of the following happen:*

(*1*) *AMCV terminates with a correct answer for the* ϵ*-approximate* k*-MCV problem;*

(*2*) *at termination,* $p \in [p^\star/8, p^\star]$.

PROOF. Below we redefine the three events in the proof of Lemma 9 (referred to as the *old proof* in the sequel) and bound their occurrence probabilities:

—*Premature*: AMCV terminates when $p > p_k^\star$. When $p > p_k^\star$, the argument in the old proof shows that, with probability at least $1 - \delta/3$, no vertex $b \in B \setminus \{b_1^\star, ..., b_{k-1}^\star\}$ satisfies $solid(b) \geq 2sp$. Hence, *Premature* occurs with probability at most $\delta/3$.

—*Overdue*: AMCV does not terminate after invoking NS with a $p < p_k^\star/4$. The argument in the old proof can be used to prove that, when $p < p_k^\star/4$,

$$\Pr[solid(b_i^\star) \leq 2sp] \leq (\delta/3)/n$$

for all $i \leq k$. As a result, the probability that $solid(b_i^\star) > 2sp$ for all $i \leq k$ is at least $1 - \delta/3$ by the union bound. In other words, *Overdue* occurs with probability at most $\delta/3$.

—*Wrong-result:* AMCV terminates when $p \in [p^\star/8, p^\star]$, but returns an incorrect result. The occurrence probability of this event is bounded above by $\delta/3$ due to Lemma 11.

If either of the two conditions stated in the lemma is violated, one of these events must have occurred. Hence, with the union bound, the above discussion completes the proof. □

The proof of Lemma 10 applies to the adapted AMCV directly, after changing $t^\star$ to $t_k^\star$. Therefore, we arrive at:

THEOREM 5. *For any* $\delta \in (0,1)$*, there is an algorithm that solves the* ϵ*-approximate* k*-MCV problem with probability at least* $1-\delta$*, and probes* $O(\frac{1}{\epsilon^2}\frac{nm}{t_k^\star}\log\frac{n}{\delta})$ *edges in expectation, where* $t_k^\star$ *is the degree of the* k*-th most connected black vertex.*

Remark. For fixed ϵ and δ, the cost of our AMCV algorithm beats the $\Omega(nm)$ lower bound of solving the exact k-MCV problem as long as $t_k^\star = \omega(\log n)$. As another interesting case, when $t_k^\star = \Omega(m)$, AMCV probes only $O(n \log n)$ edges.

7. EXPERIMENTS

In the sequel, we experimentally evaluate the performance of the proposed algorithms. Section 7.1 describes the data employed in our experimentation, and Section 7.2 clarifies the alternative methods to be examined. Sections 7.3-7.5 present the results on the exact k-MCV problem. Specifically, Section 7.3 explores under which environments can the problem be settled much faster than the naive solution that simply probes all edges. Sections 7.4 and 7.5 evaluate the proposed techniques in the random-probe and deterministic-probe algorithm classes, respectively. Finally, Section 7.6 is devoted to the approximate k-MCV problem.

7.1 Datasets

Our experiments are based on synthetic and real data which are explained in the sequel:

Power-law graphs. This is a family of synthetic graphs where the degrees of black vertices follow a *power law* distribution. Each graph is generated as follows. It has 5000 black and white vertices, respectively (i.e., $|B| = |W| = 5000$). For each black vertex $b \in B$, its degree $deg(b)$ equals d ($0 \leq d \leq 5000$) with probability

$$c(d+1)^{-\gamma} \tag{23}$$

where γ is a parameter of the power law, and c is a normalizing constant chosen to make $\sum_{d=0}^{5000} c(d+1)^{-\gamma}$ equivalent to 1 (i.e., $c = 1/\sum_{d=0}^{5000}(d+1)^{-\gamma}$). Once $deg(b)$ is decided, the $deg(b)$ white vertices connected to b are selected randomly.

As will be clear in the next section, we often need to control the *average degree* $\overline{deg}$ of the black vertices in a power-law graph. Hence, we need to set the parameter γ to generate a graph with the desired $\overline{deg}$. This is achieved by utilizing the fact that the expectation of the power law in (23) is:

$$\sum_{d=0}^{5000} \left(cd(d+1)^{-\gamma}\right)$$

Therefore, we can solve γ by equating the above formula to $\overline{deg}$.

NBA. This is a real graph selected to assess the benefits of the proposed algorithms when they are incorporated into the execution engine of a relational DBMS. The original data (from *www.nba.com*) consists of 16739 NBA players in history. For each player, the dataset contains his performance statistics in 13 aspects, such as the numbers of points scored, rebounds, assists, etc. We define a *dominating relationship* between players based on the concept of k-*dominance* [Chan et al. 2006]. Specifically, a player p_1 *7-dominates* another player p_2 if p_1 has better statistics than p_2 in at least 7 aspects (i.e., a majority of the total 13 aspects). We want to find the k players that 7-dominate the largest number of players, as given by the following pseudo-SQL statement[2]:

`SELECT` p_1 `FROM` PLAYER p_1, PLAYER p_2
`WHERE` p_1 7-dominates p_2
`GROUP BY` p_1
`HAVING` $count(*) \geq$ the size of the k-th largest group

where PLAYER is a table with 13 attributes, and one row for each player. The entire table occupies less than 1 mega bytes, and can be comfortably kept in main memory. Therefore, the total overhead is determined by the number of times the join predicate is evaluated. As explained in Section 1.1, evaluating the above statement is a k-MCV problem on a bipartite graph $G = (B, W, E)$, where each of the vertex sets B and W includes all the players, and the edge set E has an edge between two players $b \in B$ and $w \in W$ if b 7-dominates w. The optimization goal is to minimize the number of edges probed.

[2]This statement is essentially a *top-k dominating query*, which has been studied in [Papadias et al. 2005; Yiu and Mamoulis 2009]. However, the solutions in [Papadias et al. 2005; Yiu and Mamoulis 2009] are designed for a different dominance definition, where an item p_1 dominates another p_2 if and only if p_1 is better than p_2 in *all* aspects. Those solutions heavily rely on *transitivity*, namely, the fact that p_1 dominates p_2 and p_2 dominates p_1 implies that p_1 dominates p_3. As shown in [Chan et al. 2006], transitivity does *not* hold on k-dominance.

***Actor*.** This is a real graph chosen to evaluate our algorithms in a *querying-by-web-service* environment (introduced in Section 1.1). The underlying data, which is publicly available at IMDB (*www.imdb.com*), is a social network between a set of actors, where two actors have an edge if they collaborated in a movie before. We extracted the 10000 most "active" actors that have the largest number of collaborators, and focused on studying their *2-hop relationships*. Specifically, an actor a_1 has a 2-hop relationship with another actor a_2 if either a_1 is a collaborator of a_2, or they have a common collaborator (i.e., a_1 is at most two hops away from a_2 in the social network). Note that 2-hop relationships are an important type of characteristics of a social network, as pointed out in [Singla and Richardson 2008].

We aimed at finding the k actors that have the largest number of 2-hop relationships. This is a k-MCV problem on a graph $G = (B, W, E)$, where each of B and W contains all the actors, and E has an edge between two actors $b \in B$ and $w \in W$ if b has a 2-hop relationship with w. Detecting a 2-hop relationship between b and w can be accomplished by submitting the names of b and w to the website *Cinema Freenet* (see Section 1.1) and obtaining its reply. The overall cost is dominated by the network latency, which in turn is decided by the total number of relationships checked (i.e., the number of edges in E probed).

7.2 Methods

Exact *k*-MCV. Since no previous solution is known for the k-MCV problem, we concentrate on comparing the proposed algorithms *sample-and-sort* (SS) and *switch-on-empty* (SOE), both of which were presented in Section 4. The value of k will be varied from 1 to 100. Since the black vertex set B in all our data graphs have at least $n = 5000$ vertices, the condition $k \leq n/2$ always holds.

The cost of an algorithm is measured in the number of edge-probing queries issued (if the algorithm is randomized, the cost reported is the average of 5 runs). Sometimes we will also give a theoretical *lower bound* (LB) of the cost of any algorithm on the same data input. The lower bound is derived using the fact that the cost of SOE can be greater than that of the optimal algorithm by a factor of at most $1 + k/(n - k)$ (see Theorems 2 and 3 and apply $k \leq n/2$). Therefore, if SOE needs to probe x edges, we will report a lower bound of $\frac{x}{1+k/(n-k)}$.

In Sections 7.3 and 7.4, we study the random-probe algorithm class $\mathcal{A}_{\mathrm{RAN}}$, where an algorithm deploys the *probe-next* implementation in Figure 3. Section 7.5 investigates the deterministic-probe algorithm class $\mathcal{A}_{\mathrm{DET}}$, where an algorithm applies the *probe-next* in Figure 4.

Approximate *k*-MCV. We will focus on the AMCV algorithm proposed in Section 6, which is the sole known solution to the approximate k-MCV problem. As before, the cost of AMCV is gauged as the average number of probed edges in 5 runs, unless otherwise stated. Recall that, the building block of AMCV is the *naive-sampling* (NS) algorithm, where the number s of samples per black vertex is determined as $s = \frac{24}{p}\frac{1}{\epsilon^2}\ln\frac{3n}{\delta}$ (see Figure 12 and the adaptations in Section 6.2). For convenience, we write s to be equivalent to ξ_{theory}/p, where

$$\xi_{\mathrm{theory}} = \frac{24}{\epsilon^2}\ln\frac{3n}{\delta} \tag{24}$$

which remains fixed in all the invocations of NS in AMCV. This parameter is crucial to the

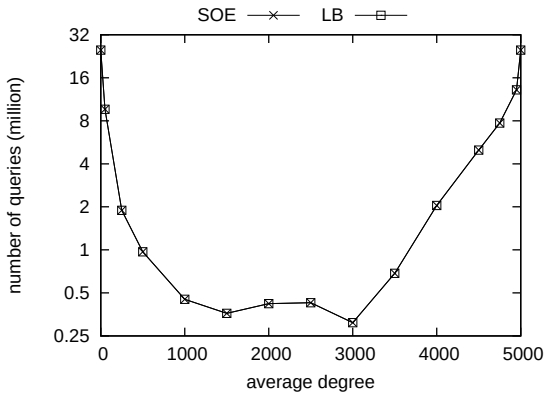


Fig. 13. Impact of the average degree of black vertices

efficiency of AMCV.

As with most randomized algorithms, the theoretical analysis of AMCV is rather pessimistic, which in our context means that the value of s as computed with ξ_{theory} is typically unnecessarily larger than what is needed in practice by a wide margin. The main cause is the extensive use of the union bound, which is known to be, albeit helpful for theoretical analysis, almost always excessively loose in reality. The implication is that, in practice, there is hope for utilizing a much smaller s to achieve the desired precision requirements.

To give a simple heuristic of setting s for practical use, we aim at replacing ξ_{theory} with a good, much lower, ξ so that we can calculate s to be ξ/p. With a tuning process to be presented in Section 7.6, we observed that

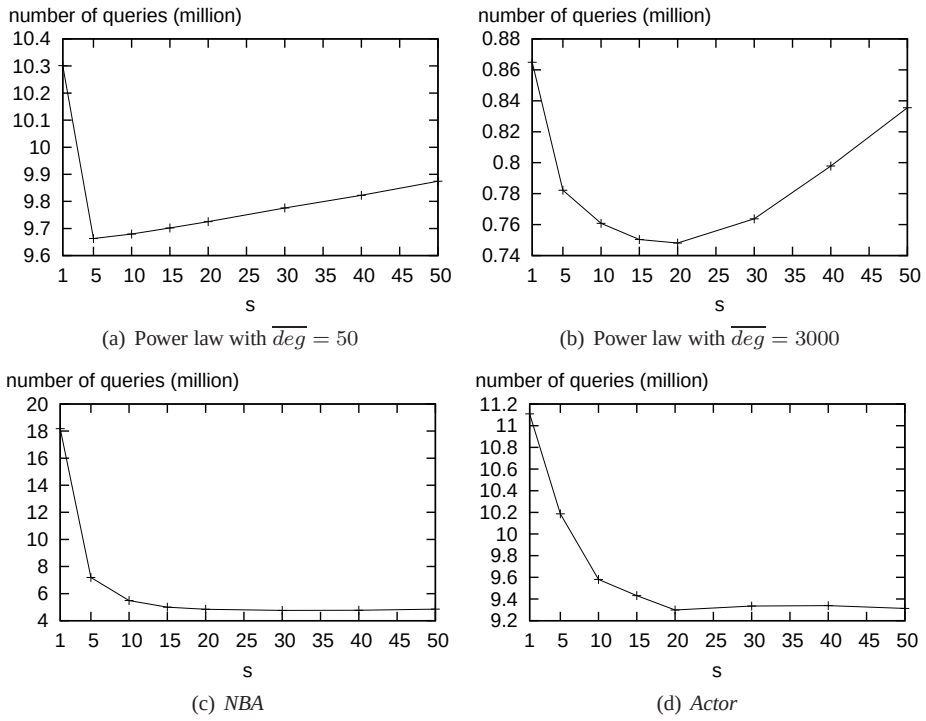
$$\xi_{heuristic} = \xi_{\text{theory}}/2000 \tag{25}$$

turns out to be a nice choice. The resulting version of AMCV, which is the same as the theoretical version but applies the above equation to decide s, is referred to as AMCV-H (standing for heuristic AMCV).

7.3 How pessimistic is the worst case?

If B and W have n and m edges respectively, solving a k-MCV problem requires probing nm edges in the worst case. The objective of this subsection is to find out when it is possible to achieve cost (much) lower than nm. For this purpose, we generated a series of power-law graphs whose $\overline{deg}$ (i.e., the average degree of black vertices) ranges from the minimum 0 to the maximum 5000. Then, we measured the performance of SOE (the version in $\mathcal{A}_{\text{RAN}}$) in settling the 10-MCV problem on each of these graphs.

Figure 13 plots the cost of SOE and the lower bounds as a function of $\overline{deg}$ (notice that the vertical axis is in log scale). Recall that both n and m are 5000 in every power-law graph, so the value of nm equals 25 million. When $\overline{deg}$ is close to the extreme value 0 or 5000, SOE needs to probe all the edges, and thus, incurs the worst-case cost. However, its efficiency improves dramatically soon after $\overline{deg}$ moves away from the extreme values. For example, when $\overline{deg}$ equals 250 (i.e., on average, a black vertex is connected to 5% of the white vertices), SOE probes around 2 million edges, which is smaller than the worst case by a factor over an order of magnitude. The minimum overhead of SOE is observed when



(a) Power law with $\overline{deg} = 50$ (b) Power law with $\overline{deg} = 3000$

(c) *NBA* (d) *Actor*

Fig. 14. Tuning the parameter s of algorithm SS

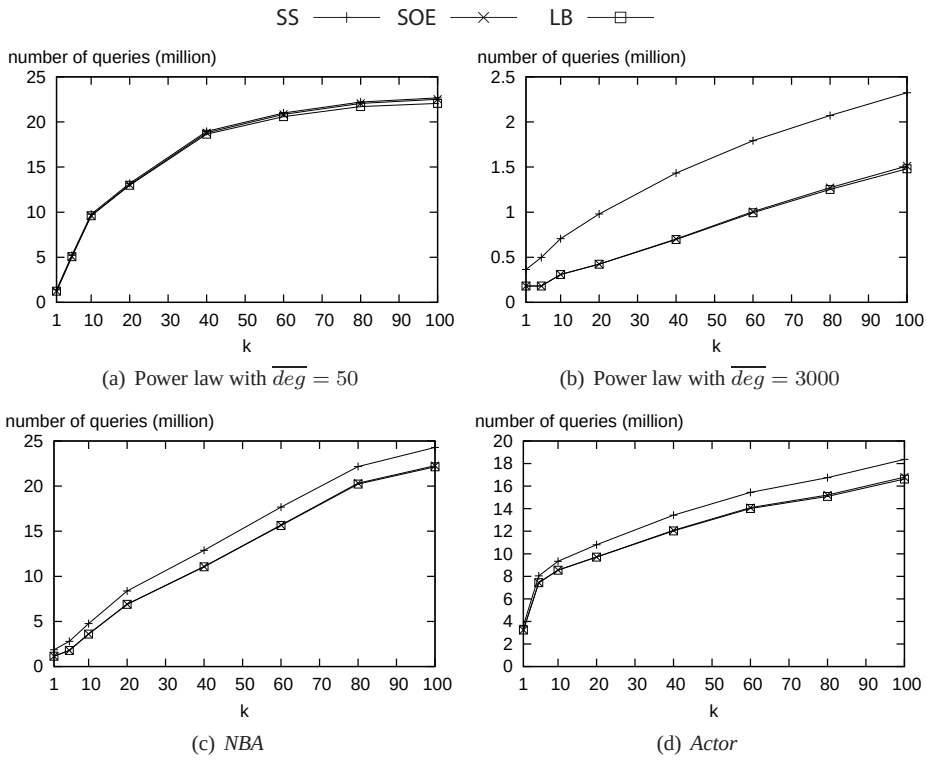
$\overline{deg}$ is close to the middle value 2500; in this case, SOE needs to probe only less than half million edges.

It is clear that the worst-case cost can occur *only in a highly sparse or dense graph*. For other graphs, the cost can be substantially reduced. The efficiency of SOE is built exactly on this observation. In fact, as shown in Figure 13, the cost of SOE is very close to the lower bound.

7.4 Performance of random-probe algorithms

Tuning *sample-and-sort*. Recall that algorithm SS needs a parameter s, which is the number of edges that are probed for each black vertex in the sampling phase. The next set of experiments aims to decide a good value of s. Towards this, given a data graph $G = (B, W, E)$, we measure the cost of SS when s is set to 1, 2, ..., 50, respectively. Figure 14 shows the results when the input G is the power law graphs with $\overline{deg} = 50$ and 3000 respectively, and the real graphs *NBA* and *Actor*. Clearly, the best value of s (minimizing the overhead of SS) is different for each dataset. Nevertheless, a common pattern is that SS is expensive when s is too small. Overall, a good choice of s is around 20, which achieves reasonable efficiency in all cases. Therefore, we fix s to 20 in the following experiments.

Scalability with k. We proceed to compare SOE and SS in k-MCV computation by increasing k from 1 to 100. Figure 15 illustrates the results, as well as the lower bounds, on the same graphs in Figure 14. For benchmarking, remember that the worst-case cost is 25



(a) Power law with $\overline{deg} = 50$ (b) Power law with $\overline{deg} = 3000$

(c) *NBA* (d) *Actor*

Fig. 15. Performance vs. k (random-probe class)

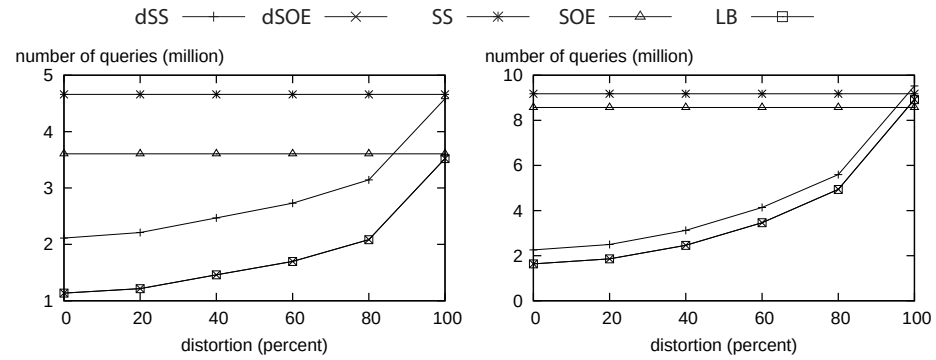
million for power-law graphs, $16739^2 > 280$ million for *NBA*, and $10000^2 = 100$ million for *Actor*.

The overhead of SS and SOE is always significantly lower than the worst case (often by orders of magnitude), especially for $k \leq 10$. The only exception is in Figure 15a, when k approaches 100. This is expected because a graph with $\overline{deg} = 50$ is very sparse (on average, a black vertex is connected to only 1% of the white vertices), so most of the edges must be probed to deal with a relatively large k. In all the experiments, SOE consistently outperforms SS, and its cost is only slightly higher than the lower bounds.

7.5 Performance of deterministic-probe algorithms

The previous experiments focused on the random-probe algorithm class $\mathcal{A}_{\mathrm{RAN}}$. This subsection evaluates SS and SOE when they are deployed as algorithms in the deterministic-probe class $\mathcal{A}_{\mathrm{DET}}$. Recall that every algorithm in $\mathcal{A}_{\mathrm{DET}}$ probes the hidden edges of each black vertex in the same *probing sequence* (instead of a random order as in $\mathcal{A}_{\mathrm{RAN}}$) that is prescribed by the underlying application (see Figure 4).

The following experiments have two objectives. The first one is to inspect the efficiency of SS and SOE in the deterministic scenario. The second, perhaps more interesting, objective is to understand how their efficiency is affected by the ordering of the white vertices in the probing sequence. For this purpose, we considered a set of sequences that are controlled by a parameter called *distortion* d, which ranges from 0 to 1. Specifically, a sequence with



(a) *NBA* (b) *Actor*

Fig. 16. Effects of distortion (deterministic-probe class)

distortion 0 ranks the white vertices in ascending order of their degrees (or equivalently, in descending order of how many empty edges they have). On the other extreme, a sequence with distortion 1 is simply a random permutation of the white vertices. In general, in a sequence with distortion d, the positions of dm white vertices are randomly permutated (the other white vertices remain in ascending order of their degrees), where m is the number of white vertices.

To distinguish with the SS (SOE) in the random-probe class $\mathcal{A}_{\mathrm{RAN}}$, we refer to the version of SS (SOE) in the deterministic-probe class $\mathcal{A}_{\mathrm{DET}}$ as dSS (dSOE). The parameter s of dSS is also set to 20, after a tuning process similar to Figure 14. Concerning 10-MCV computation on *NBA*, Figure 16a plots the performance of dSS and dSOE as a function of distortion, together with the theoretical lower bounds (which are calculated by dividing the cost of dSOE by $1 + \frac{10}{n-10}$, where n is the number of black vertices). For referencing, we also include the cost of SS and SOE so that comparison can be made between random- and deterministic-probe solutions. In the same fashion, Figure 16b presents the 10-MCV results on *Actor*.

Clearly, dSS and dSOE benefit significantly from a sorted ordering. In particular, when distortion is 0 (i.e., completely sorted), the cost of dSOE is nearly 10 times lower than its cost when distortion is 1 (i.e., completely random). In general, the overhead of both dSS and dSOE grows with distortion, and eventually (i.e., at distortion 1) reaches the cost of SS and SOE. This phenomenon is not surprising at all. When the white vertices with more empty edges are probed first, many empty edges can be discovered sooner for each black vertex. As a result, the upper bounds of the degrees of the black vertices drop faster, which enables earlier termination. The relative performance of dSS and dSOE is similar to the random-probe class reported in Figure 15. Also, dSOE is once again nearly optimal, leaving little room for further improvements. It is worth pointing out that, the above results do not imply the superiority of dSOE over SOE in all cases, which can be easily disproved by designing an adverse probing sequence that forces dSOE to discover, for each non-result vertex, many solid edges before empty edges. Consider the example in Figure 10, on which the expected cost of SOE is approximately $1.56m$, as explained in Section 5.1. If the probing sequence is such that all the solid edges of b are probed before its empty edges, then dSOE needs to probe $m/10 + 1 + m/2 = 0.6m + 1$ edges of b, in order to find $1 + m/2$ empty edges. As dSOE also checks all the m edges of $b^\star$, the total overhead of

dSOE is $1.6m + 1 > 1.56m$, i.e., more expensive than the expected cost of SOE.

7.6 Performance of AMCV

Having elaborated on the characteristics of our exact solutions, we now proceed to study the behavior of the proposed algorithm AMCV for the approximate k-MCV problem. As explained earlier, AMCV applies (24) to calculate the parameter s, which as will be shown shortly is much larger than necessary. Hence, the first set of experiments below is designed to (i) measure how small s can be without violating the precision constraints, and (ii) examine the effectiveness of the heuristic in Section 7.2 that resorts to (25). As the second step, we compare the efficiency of AMCV to our fastest exact algorithm, namely, SOE.

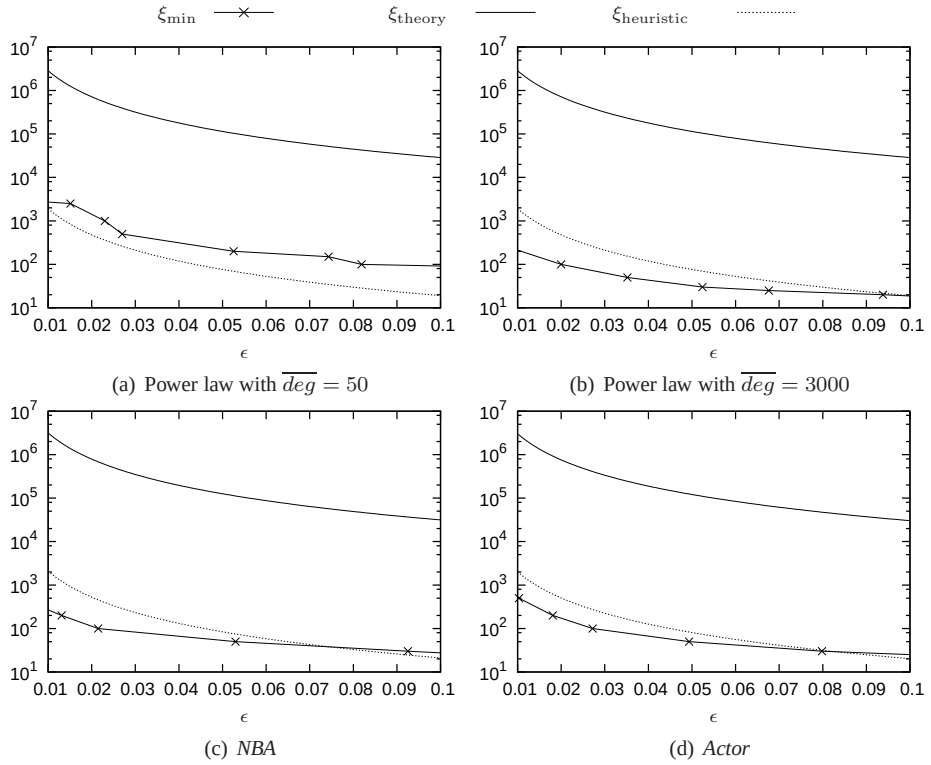
Behavior of AMCV in practice. Given a value of ξ, let us use $[\xi]$-AMCV to refer to the algorithm that differs from AMCV only in that the value of s is set to be ξ/p. In other words, AMCV is essentially $[\xi_{\text{theory}}]$-AMCV, whereas AMCV-H is $[\xi_{\text{heuristic}}]$-AMCV, with ξ_{theory} and $\xi_{\text{heuristic}}$ given in (24) and (25), respectively. Lowering ξ reduces the execution cost of the algorithm, but on the other hand, increases the risk of being unable to meet the precision guarantees as mandated by ϵ and δ. Given a fixed pair of ϵ and δ and a particular dataset, we define ξ_{min} to be the minimum ξ such that $[\xi]$-AMCV is able to achieve the desired precision requirements on that dataset. Equivalently, the minimum value of s for attaining those requirements equals ξ_{min}/p.

To measure ξ_{min}, we started with a large ξ and gradually decreased it. For each ξ, we ran algorithm $[\xi]$-AMCV (on the underlying dataset) 100 times, and recorded the number x of times the algorithm successfully returned a result that is legal under the definition of ϵ-approximate k-MCV. We say that the ξ is *acceptable* if $x/100 \geq 1 - \delta$. Then, ξ_{min} took the value of the smallest acceptable ξ. The value of k was fixed to 10 in all the following experiments, unless otherwise stated.

To examine how $\xi_{\min}$ scales with ϵ, we fixed $\delta = 0.1$ and measured $\xi_{\min}$ as ϵ varied from 0.01 to 0.1. The results on the power-law graph with $\overline{deg} = 50$ are presented in Figure 17a, where the corresponding ξ_{theory} and $\xi_{\text{heuristic}}$ are also given for comparison. The results of the same experiment the power-law graph with $\overline{deg} = 3000$, *NBA* and *Actor* are illustrated in Figure 17b, 17c and 17d, respectively. It is clear that $\xi_{\min}$ is always lower than ξ_{theory} by orders of magnitudes, confirming our earlier conjecture that ξ_{theory} obtained from theoretical analysis is over pessimistic in practice. Furthermore, observe that $\xi_{heuristic}$ presents itself as a nice fitting line of ξ_{min}.

In a similar experiment, we inspected the behavior of $\xi_{\min}$ with respect to δ, by fixing ϵ to 0.05 while increasing δ from 0.05 to 0.5. Figure 18 demonstrates the results on the same datasets as in Figure 17. Once again, $\xi_{\min}$, closely approximated by $\xi_{\text{heuristic}}$, is significantly smaller than its theoretical counterpart ξ_{theory}.

Efficiency of AMCV-H. Treating algorithm SOE as a benchmark, the subsequent experiments evaluate the performance of AMCV-H, i.e., the practical version of AMCV parameterized by $\xi_{heuristic}$ as discussed in Section 7.2. We start by assessing how the cost of the algorithms is affected by k. Figure 19 plots the cost as a function of k on different datasets, when ϵ and δ are set to 0.05 and 0.1, respectively. Recall that, due to its heuristic nature, AMCV-H may not achieve the theoretical guarantees prescribed by ϵ and δ. Whenever this happens, in the diagrams of Figure 19, we present a percentage ϵ_{actual} to indicate that the output by AMCV-H is an ϵ_{actual}-approximate k-MCV result according to the failure probability designated by δ. For example, in Figure 19a, the 6.2% means



(a) Power law with $\overline{deg} = 50$

(b) Power law with $\overline{deg} = 3000$

(c) *NBA*

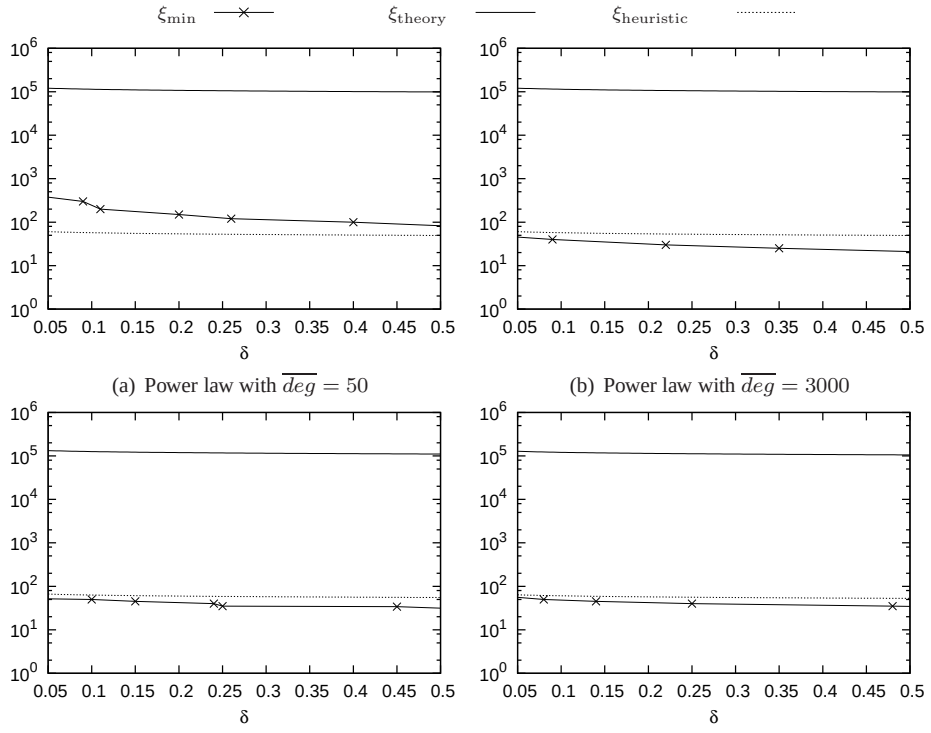
(d) *Actor*

Fig. 17. $\xi_{\min}$, ξ_{theory}, and $\xi_{\text{heuristic}}$ vs. ϵ ($\delta = 0.1, k = 10$)

that, for $k = 7$, AMCV-H achieves the precision level of 0.062-approximate k-MCV with failure probability $\delta = 0.1$.

A general observation from Figure 19 is that AMCV-H significantly outperforms SOE when the cost of SOE is large, i.e., the data input is "hard". When the input is "easy", both algorithms are very fast with AMCV-H sometimes being more expensive. This is consistent with the common understanding that probabilistic algorithms find their values mainly in dealing with datasets that are costly to process with deterministic algorithms. Another key observation is that the cost of AMCV-H is insensitive to k, while that of SOE increases rapidly with this parameter. In fact, we can see that for $k = 10$, AMCV-H is always better than SOE except on the easiest input (i.e., the graph in Figure 19b).

The next experiments inspect the influence of ϵ and δ, by fixing k to 10. Setting $\delta = 0.1$, Figure 20 presents the results when ϵ varies from 0.01 to 0.1, while setting $\epsilon = 0.05$, Figure 21 presents the results when δ varies from 0.05 to 0.5. We annotate all diagrams with percentages that carry the same meanings as in Figure 19. The main observation in Figure 20 is that AMCV-H is expensive when ϵ is very low, i.e., an exceedingly small error is targeted, such that in this case we would be better off by simply running the exact algorithm SOE. However, AMCV-H quickly improves as ϵ increases, and outperforms SOE in all datasets starting from $\epsilon = 0.06$. On the other hand, as shown in Figure 21, the cost of AMCV-H is insensitive to δ, which is expected because δ appears in a logarithm in the running time (see Theorem 5).



(a) Power law with $\overline{deg} = 50$

(b) Power law with $\overline{deg} = 3000$

(c) *NBA*

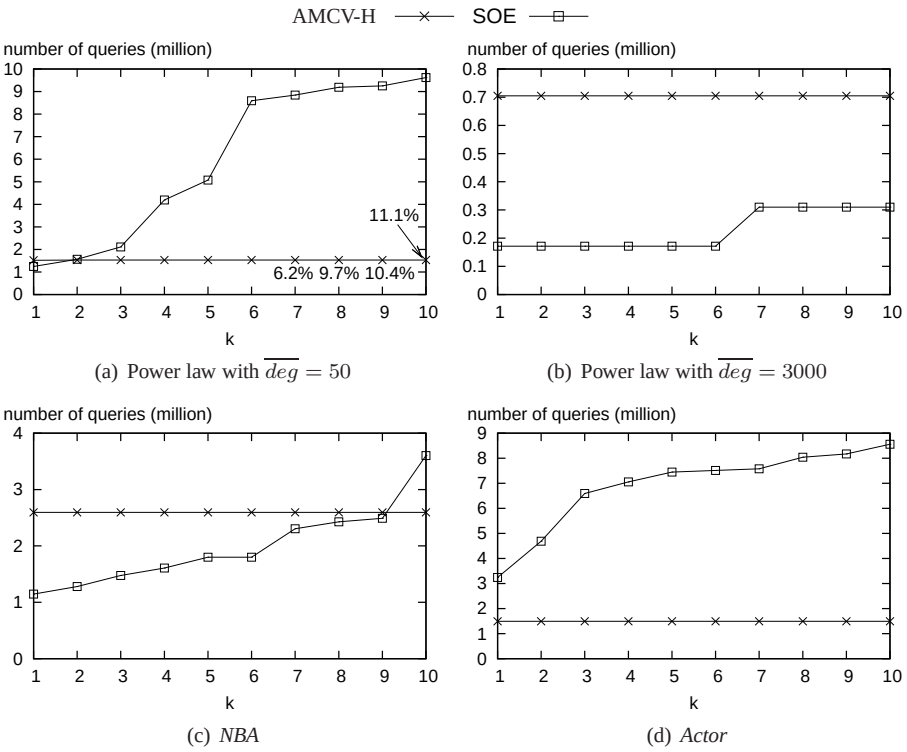
(d) *Actor*

Fig. 18. $\xi_{\min}$, ξ_{theory}, and $\xi_{\text{heuristic}}$ vs. δ ($\epsilon = 0.05, k = 10$)

In general, as long as ϵ is not excessively small, AMCV-H is efficient regardless of the data input. This is a nice advantage over SOE, which can be as expensive as the naive solution when the input is hard, as is evident from Figure 13. Hence, AMCV-H is preferred in scenarios where ϵ-approximate results suffice, and yet, the hardness of the dataset cannot be reliably estimated. In fact, AMCV-H can even be used as a pilot run that serves as a "hardness test". Specifically, if the output of AMCV-H indicates that the degree of the k-th most connected black vertex is close to m (i.e., the number of white vertices), we can infer that the dataset is easy, and invoke SOE to find the exact answers. On the other hand, if the degree of the k-th most connected black vertex is far from m, we know that the dataset is hard, in which case running SOE should be avoided because it may incur prohibitively expensive overhead.

8. CONCLUSIONS

This paper studied the k *most connected vertex* (k-MCV) problem on an $n \times m$ hidden bipartite graph such that $k \leq n/2$. We presented an algorithm that is instance optimal in a class of randomized algorithms, and a class of deterministic algorithms. On any data input, our solution can be more expensive than the optimal algorithm of each class by a factor of at most 2. We also proved that no algorithm in the deterministic class can be optimal in all cases. Currently, it remains open whether an optimal algorithm exists in the randomized class.



(a) Power law with $\overline{deg} = 50$ (b) Power law with $\overline{deg} = 3000$

(c) *NBA* (d) *Actor*

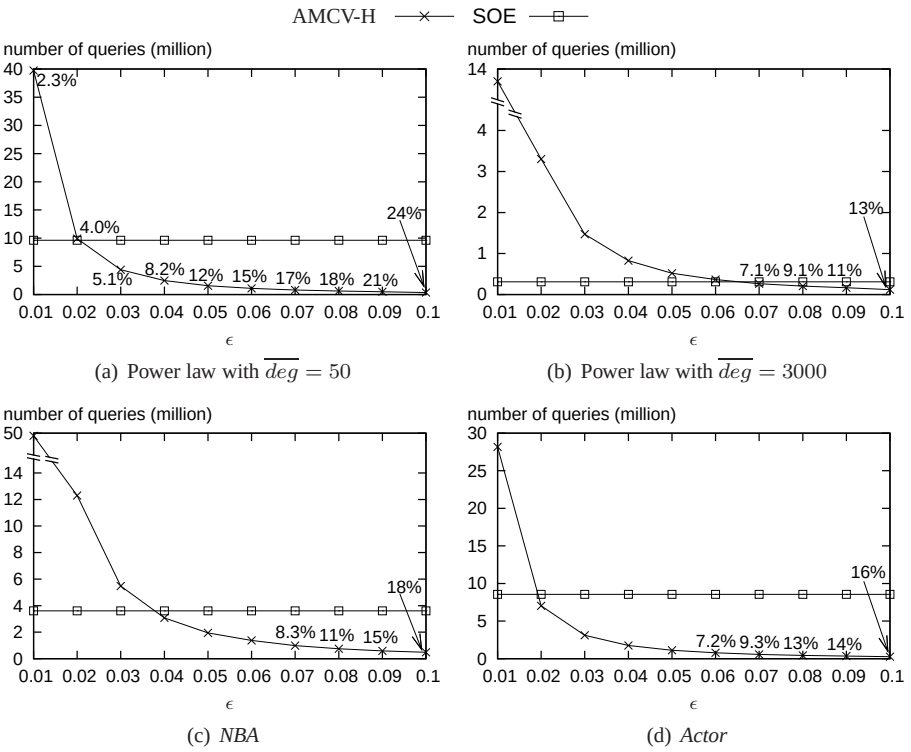
Fig. 19. Performance vs. k ($\epsilon = 0.05, \delta = 0.1$)

As a second step, we gave an algorithm for solving an ϵ-approximate version of the k-MCV problem with probabilistic quality bounds. While this paper has concentrated on bipartite graphs, our algorithm can be extended to work on general graphs, still ensuring all the theoretical guarantees.

We believe that *query processing in hidden graphs* is a promising research direction. For future work, one may consider generalizing the k-MCV problem to multi-partite graphs, namely, the top-k version of t-ary semi-join with $t > 2$. It may also be interesting to re-visit conventional graph problems on hidden graphs. The existing algorithms may not regard edge-probing as a costly operation, and thus, can be prohibitively expensive if applied in a straightforward manner.

Acknowledgements

This work was (i) supported by WCU (World Class University) program under the National Research Foundation of Korea, and funded by the Ministry of Education, Science and Technology of Korea (Project No: R31-30007), (ii) supported by grants 4169/09, 4166/10, 4165/11 from HKRGC, and (iii) supported by National Grant Fundamental Research 973 Program of China (project No: 2012CB316200). We would like to thank the anonymous reviewers for their insightful comments.



(a) Power law with $\overline{deg} = 50$

(b) Power law with $\overline{deg} = 3000$

(c) *NBA*

(d) *Actor*

Fig. 20. Performance vs. ϵ ($k = 10, \delta = 0.1$)

REFERENCES

AFSHANI, P., BARBAY, J., AND CHAN, T. M. 2009. Instance-optimal geometric algorithms. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 129–138.

ALON, N., BEIGEL, R., KASIF, S., RUDICH, S., AND SUDAKOV, B. 2004. Learning a hidden matching. *SIAM Journal of Computing 33,* 2, 487–501.

ALON, N. AND KRIVELEVICH, M. 2002. Testing k-colorability. *SIAM Journal of Computing 15,* 2, 211–227.

ALON, N. AND SHAPIRA, A. 2008a. A characterization of the (natural) graph properties testable with one-sided error. *SIAM Journal of Computing 37,* 6, 1703–1727.

ALON, N. AND SHAPIRA, A. 2008b. Every monotone graph property is testable. *SIAM Journal of Computing 38,* 2, 505–522.

ANGLES, R. AND GUTIÉRREZ, C. 2008. Survey of graph database models. *ACM Computing Surveys 40,* 1.

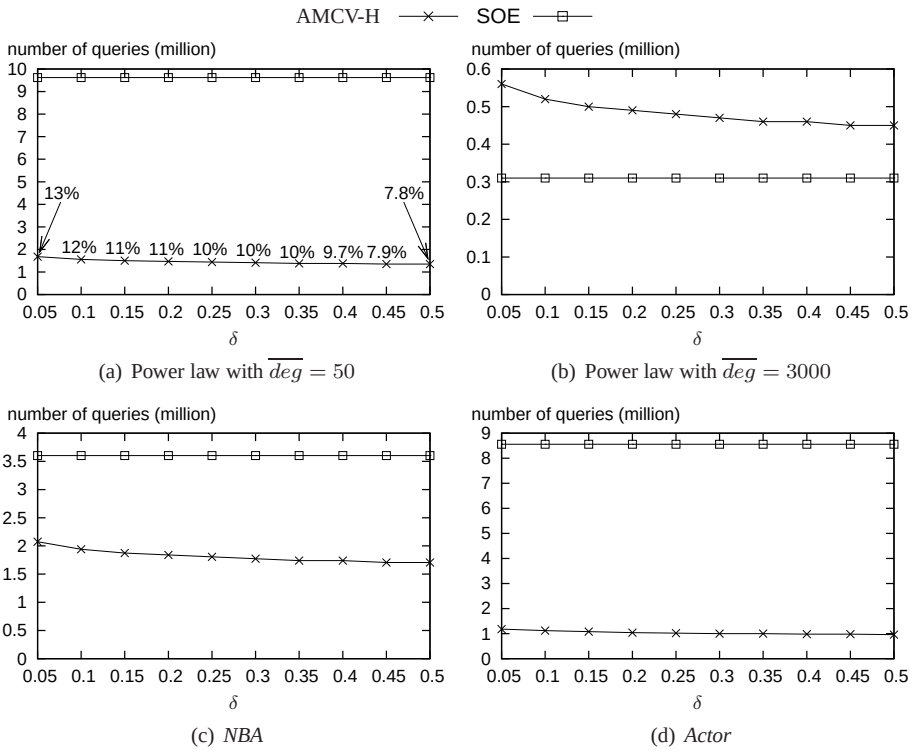
ANGLUIN, D. AND CHEN, J. 2008. Learning a hidden graph using O(logn) queries per edge. *Journal of Computer and System Sciences (JCSS) 74,* 4, 546–556.

BARAN, I. AND DEMAINE, E. D. 2005. Optimal adaptive algorithms for finding the nearest and farthest point on a parametric black-box curve. *Int. J. Comput. Geometry Appl. 15,* 4, 327–350.

BARBAY, J. AND CHEN, E. Y. 2008. Convex hull of the union of convex objects in the plane: an adaptive analysis. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*.

BIEDL, T. C., BREJOVÁ, B., DEMAINE, E. D., HAMEL, A. M., LÓPEZ-ORTIZ, A., AND VINAR, T. 2004. Finding hidden independent sets in interval graphs. *Theor. Comput. Sci. 310,* 1-3, 287–307.

BOGDANOV, A., OBATA, K., AND TREVISAN, L. 2002. A lower bound for testing 3-colorability in bounded-degree graphs. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 93–102.



(a) Power law with $\overline{deg} = 50$

(b) Power law with $\overline{deg} = 3000$

(c) *NBA*

(d) *Actor*

Fig. 21. Performance vs. δ ($k = 10, \epsilon = 0.05$)

BORODIN, A. AND EL-YANIV, R. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press.

CHAN, C. Y., JAGADISH, H. V., TAN, K.-L., TUNG, A. K. H., AND ZHANG, Z. 2006. Finding k-dominant skylines in high dimensional space. In *Proceedings of ACM Management of Data (SIGMOD)*. 503–514.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms, Second Edition.* The MIT Press.

DEMAINE, E. D., HARMON, D., IACONO, J., KANE, D., AND PĂTRAŞCU, M. 2009. The geometry of binary search trees. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 496–505.

DEMAINE, E. D., LÓPEZ-ORTIZ, A., AND MUNRO, J. I. 2000. Adaptive set intersections, unions, and differences. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 743–752.

FAGIN, R., LOTEM, A., AND NAOR, M. 2001. Optimal aggregation algorithms for middleware. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

GOLDREICH, O., GOLDWASSER, S., AND RON, D. 1998. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM) 45*, 4, 653–750.

HAGERUP, T. AND RUB, C. 1990. A guided tour of chernoff bounds. *Information Processing Letters (IPL) 33*, 6, 305–308.

HOULE, M. E. AND SAKUMA, J. 2005. Fast approximate similarity search in extremely high-dimensional data sets. In *Proceedings of International Conference on Data Engineering (ICDE)*. 619–630.

ILYAS, I. F., AREF, W. G., AND ELMAGARMID, A. K. 2003. Supporting top-k join queries in relational databases. In *Proceedings of Very Large Data Bases (VLDB)*. 754–765.

ILYAS, I. F., BESKALES, G., AND SOLIMAN, M. A. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys 40,* 4.

IMIELINSKI, T., VISWANATHAN, S., AND BADRINATH, B. R. 1997. Data on air: Organization and access. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 9,* 3, 353–372.

KAPOOR, S. 2000. Dynamic maintenance of maxima of 2-d point sets. *SIAM Journal of Computing 29,* 6, 1858–1877.

KEOGH, E. J. 2002. Exact indexing of dynamic time warping. In *Proceedings of Very Large Data Bases (VLDB)*. 406–417.

MATUSZEWSKI, T. I. 1962. Some properties of pascal distribution for finite population. *Journal of the American Statistical Association 57,* 297, 172–174.

NATSEV, A., CHANG, Y.-C., SMITH, J. R., LI, C.-S., AND VITTER, J. S. 2001. Supporting incremental join queries on ranked inputs. In *Proceedings of Very Large Data Bases (VLDB)*. 281–290.

PAPADIAS, D., TAO, Y., FU, G., AND SEEGER, B. 2005. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS) 30,* 1, 41–82.

SCHNAITTER, K. AND POLYZOTIS, N. 2008. Evaluating rank joins with optimal cost. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*. 43–52.

SINGLA, P. AND RICHARDSON, M. 2008. Yes, there is a correlation: - from social networks to personal behavior on the web. In *Proceedings of International World Wide Web Conferences (WWW)*. 655–664.

SOLIMAN, M. A., ILYAS, I. F., AND CHANG, K. C.-C. 2008. Probabilistic top-k and ranking-aggregate queries. *ACM Transactions on Database Systems (TODS) 33,* 3.

TAO, Y., SHENG, C., AND LI, J. 2010. Finding maximum degrees in hidden bipartite graphs. In *Proceedings of ACM Management of Data (SIGMOD)*. 891–902.

YIU, M. L. AND MAMOULIS, N. 2009. Multi-dimensional top- dominating queries. *The VLDB Journal 18,* 3, 695–718.

ZHU, M., PAPADIAS, D., ZHANG, J., AND LEE, D. L. 2005. Top-k spatial joins. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 17,* 4, 567–579.