Finding Maximum Degrees in Hidden Bipartite Graphs[*]

Yufei Tao   Cheng Sheng
Chinese University of Hong Kong
{taoyf, csheng}@cse.cuhk.edu.hk

Jianzhong Li
Harbin Institute of Technology
lijzh@hit.edu.cn

ABSTRACT

An *(edge) hidden graph* is a graph whose edges are not explicitly given. Detecting the presence of an edge requires expensive *edge-probing* queries. We consider the k *most connected vertex* problem on hidden bipartite graphs. Specifically, 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 a semi-join, and is encountered in many applications in practice (e.g., *top-k spatial join* with arbitrarily complex join predicates).

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 easy inputs, the problem can be efficiently settled with only $km+n$ edges, 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. We give such an algorithm, and prove that it 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 currently unknown) by only a constant factor, which can be shown to be at most 2. Extensive experiments demonstrate that, in practice, the number of queries required by our technique is far less than nm, and agrees with our theoretical findings very well.

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

General Terms: Theory

Keywords: Maximum Degree, Bipartite Graph, Competitive Analysis, Instance Optimality

[*]Yufei Tao and Cheng Sheng were supported by grants GRF4161/07, GRF 4173/08, GRF4169/09 from HKRGC, and a direct grant (2050395) from CUHK. Jianzhong Li was supported by National Grant Fundamental Research 973 Program of China (2006CB303000), and Key Program of National Natural Science Foundation of China (60533110).

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 performing one, sometimes several, expensive operations, each of which is called an *edge-probing query*. In recent years, *learning hidden graphs* [18] has attracted considerable attention in the theory community [4, 7, 11, 18]. The main objective of the relevant research is to find out whether the graph has a certain *property*, by issuing the least number of edge-probing queries. 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 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* (kMCV) 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 $\{b_1, b_2, ..., b_4\}$, and W is $\{w_1, w_2, ..., w_5\}$; then, the 1MCV problem should return b_2. Unlike many graph problems, the novel feature of kMCV is that the edges of G are *unknown* initially, such that a costly edge-probing query is required to detect the presence of an edge. The challenge is to solve the problem using the minimum number of queries. The problem is encountered in many database applications, some of which are discussed below.

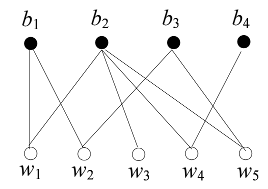


Figure 1: The 1MCV result is b_2

1.1 Motivation

Application 1 (*Semi-join aggregation with complex predicates*). Consider B and W as relational tables, and the edge-probing query as a join condition between B and W. The result of the kMCV problem is the k tuples in B that can be joined with the most tuples in W, as described by the following pesudo-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, kMCV 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 another table W. Such an extension is useful in many scenarios. 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* [30], 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 (which cannot be supported by [30], since it focuses on Euclidean distances). 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 the kMCV problem, which may improve efficiency dramatically by reducing the number of join-predicate evaluation. 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 [30] are not appropriate for DBMS incorporation).

In fact, the concept of semi-join exists not only in relational databases, but is implicit in numerous applications of other environments. As detailed below, our kMCV 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 kMCV 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, currently, the pharmaceutical industry has been establishing a novel methodology of discovering new drugs, called *fragment-based drug discovery* [22]. This is motivated by the frustration that *"finding a new drug is like playing golf, where the target is the pin"* [22]. The new methodology relieves the frustration by initiating a drug-searching process from a *fragment*, which is a basic chemical compound common 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 kMCV 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*, which can be very 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* [23] and even ℓ_p norms in *ultra-high dimensional spaces* [19]). In those cases, the cost of edge-probing queries will most likely 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, without having to store locally the gigantic datasets. 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 (without requiring their coordinates).

These services can be leveraged to solve many kMCV 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 kMCV result is the k actors/actresses that participated in the largest number of movies. Similarly, *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. As mentioned earlier, B can be a set of hotels, and W a set of attractions. 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 kMCV 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 kMCV algorithm.

1.2 Our main results

The objective of this work is to design a generic algorithm for the kMCV 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 certain inputs, the problem can be settled efficiently 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 kMCV problem. We propose an adaptive algorithm (with the properties described earlier), and prove that it is *instance optimal* among a broad class of solutions (to be defined in the next section). Instance optimality [17] requires that, on *any* data input, our algorithm should be as fast as the optimal solution (which is currently unknown), up to only a constant factor. We are able to show that the constant is at most 2, regardless of the value of k. 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 give a strong argument that our algorithm can be slower than the optimal solution by only a tiny factor of $1 + O(1/n)$.

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 explains the preliminary concepts required by our discussion. Section 4 explains the details of the proposed algorithms, and Section 5 presents a theoretical study of their performance. Section 6 experimentally evaluates the efficiency of our techniques. Finally, Section 7 concludes the paper with a summary of our findings.

2. PROBLEM AND RELATED WORK

Next, we first expand the discussion in Section 1 to formally define the *k most-connected vertex* (kMCV) problem. Then, we review the existing research on the relevant problems.

Problem definition. 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 kMCV problem is to find the k black vertices with the largest degrees, by minimizing the number of queries (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 $b \in B$; then, the kMCV 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 $b' \in B - R$

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

Denote by n and m the numbers of vertices in B and W, respectively. Apparently, the number of edges in G can range from 0 to nm. The value of k can be any integer from 1 to n. Notice that, interestingly, kMCV is in fact the same problem as finding the $n - k$ vertices in B with the *smallest* degrees (which are exactly the vertices in $B - R$). In practice, users are usually interested in the *top few* (e.g., 10) black vertices with the maximum or minimum degrees. Therefore, the values of k that are of higher practical importance are close to either 1 or n. In the former case, k can be regarded as a constant, namely $k = O(1)$, whereas in the latter case, $n - k$ can be regarded as a constant, meaning $k = n - O(1)$. Ideally, a solution to the kMCV problem should be especially efficient in these two extreme cases.

Related work. Although *graph databases* have been extensively studied (see [6] for a recent survey), we are not aware of any previous work dealing with the kMCV problem or hidden graphs. Traditionally, the edges of a graph are given explicitly (e.g., in an *adjacency matrix*), so that accessing an edge incurs negligible cost. In that scenario, it is not expensive to find the k vertices with the largest degrees. The novel feature of our kMCV problem is that detecting an edge is costly, such that the number of edge-probing queries is the key factor deciding the overall execution time.

Learning hidden graphs, also known as *graph testing*, was first studied by Goldreich et al. [18]. 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 such a 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 [3, 11, 18] 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, [4, 5] for a summary.

In the original setup of [18], an edge-probing query is assumed to detect an edge between only two vertices. In recent years, several authors [2, 7, 10] have considered *super queries*, each of which detects whether a set of vertices induce any edge 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 their corresponding chemicals react with each other. Then, a super query can be understood as an experiment of mixing many 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 kMCV problem differs from the *graph testing* formulation of [18]. Specifically, we are not attempting to verify any general property (that is possessed by a class of graphs) as in [18]. 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 the best of our knowledge, the kMCV problem has not been addressed in the literature of graph testing.

Finding the vertex with the maximum degree is a basic operation in attacking many classical 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 problem of *minimum set cover* (MSC), which has a huge number of applications in practice. In the context of a bipartite graph between two sets B and W of vertices, 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 classical greedy algorithm [14], which requires solving multiple 1MCV problems. Our techniques can be immediately employed.

The concept of instance optimality was introduced by Fagin et al. [17]. An earlier, similar, concept is *competitive analysis* [12], whose differences from instance optimality are nicely explained in [17]. Instance optimal algorithms have been designed for many other problems, such as manipulating binary search trees [15], approximating the distance from a point to a curve [8], computing the union/intersection of sorted lists [16], finding the convex hull of polygons [9], to mention just a few. The most recent work to our knowledge is [1], which proposes instance optimal algorithms for several computational geometry problems.

Finally, the kMCV problem can be regarded as a variant of the *top-k problem*, which has been extensively studied in distributed systems [17], relational databases [21], uncertain data [28], and so on. However, the solutions in those works are specific to their own contexts, and cannot be adapted for kMCV. Another related problem in relational databases is *top-k join* [20, 24, 26], 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 kMCV, 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).

3. PRELIMINARIES

This section lays down the key concepts that pave the path to the technical discussion in the later sections. Specifically, we will first explain the classes of algorithms considered by our analysis. Then, we will elaborate the concept of instance optimality, following the framework established by Fagin et al. [17].

Algorithm classes. 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 a broad class of kMCV algorithms.

```
algorithm MCV
1. repeat
2.     b = pick-black
3.     probe-next(b)
4. until it is safe to return the result
```

Figure 2: An algorithmic framework

The framework describes two core operations that are performed repetitively by an algorithm:

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 some properties about G, such as the correlations between the edges already probed and the one to be probed next. 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 quite reasonable when the algorithm cannot predict the nature (i.e., present or not) of any hidden edge.

```
algorithm probe-next(b)
/* for the random-probe algorithm class $\mathcal{A}_{\text{RAN}}$ */
1. if b has no more hidden edge
2.     return NULL
3. $w$ = a random vertex of $W$ whose edge with $b$ remains hidden
4. return $q(b, w)$
```

Figure 3: Randomized *probe-next*(b)

- *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 in the sequence. This is a natural choice in scenarios where the data items corresponding to the white vertices are fetched in a sequential order, for example, by the *get-next* function of a search engine [17].

```
algorithm probe-next(b)
/* for deterministic-probe the algorithm class $\mathcal{A}_{\text{DET}}$ */
/* assume that the vertices in $W$ have been labeled as $w_1, w_2, ..., w_m$,
respectively */
1. $i$ = the number of edges of $b$ that have been probed
2. if $i = m$ then return NULL
3. return $q(b, w_{i+1})$
```

Figure 4: Deterministic *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: $\mathcal{A}_{\text{RAN}}$ and $\mathcal{A}_{\text{DET}}$. Specifically, $\mathcal{A}_{\text{RAN}}$, referred to as the *random-probe algorithm class*, includes algorithms that apply the randomized version; $\mathcal{A}_{\text{DET}}$, the *deterministic-probe algorithm class*, contains algorithms that apply the deterministic version. In each class, the algorithms differ in their implementations of *pick-black*. As the number of edges that need to be probed can be nm, there are $(nm)!$ different probing orders, each corresponding to an implementation of *pick-black*. Hence, the number of possible algorithms is at least $(nm)!$ in both $\mathcal{A}_{\text{RAN}}$ and $\mathcal{A}_{\text{DET}}$.

Instance optimality. In the worst case, nm edge-probing queries are needed to solve the kMCV problem. To prove this, consider an input G with no edge at all, namely, no black vertex is connected to any white vertex. As a result, any algorithm 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 kMCV 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^* in B has degree m (i.e., b^* 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 1MCV problem can be solved by issuing less than $m+n$ queries. Specifically, we can probe all the edges of b^*, and only *one* edge for every other black vertex $b \in B, b \neq b^*$. The total number of queries is $m+n-1$, but this is enough to find out that (i) b^* has degree m, and (ii) any other black vertex b has degree *at most* $m-1$. Therefore, b^* must be the only vertex in the result.

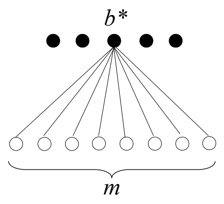


Figure 5: An easy input to 1MCV

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 very small factor.

Next, we formalize the above discussion using the concept of *instance optimality* introduced by Fagin et al. [17]. 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 $A^* \in A$ is *instance optimal* over $\mathcal{A}$ and $\mathcal{D}$ if there is a constant r satisfying

$$cost(A^*, D) \le r \cdot cost(A, D) \tag{1}$$

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

In our context, $\mathcal{A}$ is either $\mathcal{A}_{\text{RAN}}$ or $\mathcal{A}_{\text{DET}}$, and $\mathcal{D}$ includes all the bipartite graphs. Note that while all the algorithms in $\mathcal{A}_{\text{RAN}}$ must be randomized, those in $\mathcal{A}_{\text{DET}}$ 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 $\mathcal{A}_{\text{RAN}}$ or $\mathcal{A}_{\text{DET}}$) on the input graph $G \in \mathcal{D}$ (a 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^* in each algorithm class that makes Equation 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 not only the algorithm is instance optimal (notice that $1 + 1/n$ is at most 2), but it is actually nearly optimal in the absolute sense when n is large (in which case r is very close to 1).

4. ALGORITHMS

In this section, we give two algorithms for solving the kMCV 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.

Before elaborating the two algorithms, we will first introduce some key notations and explain a basic bounding strategy. Furthermore, we will first make the assumption that $k \le n/2$, where n is the number of black vertices. Later, we will remove the assumption, by extending our solutions to $k > n/2$.

Notations and basic strategy. Denote by $deg(b)$ the degree of a black vertex $b \in B$ in the input graph G. Let $R \subseteq B$ be the set of black vertices that an algorithm decides to return. As mentioned in Section 2, the algorithm must have evidence showing:

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

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 $deg(b')$.

Let us introduce two notions that will help the presentation. 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

$$solid(b) \le deg(b) \le m - empty(b). \tag{2}$$

For each $b \in B$, our algorithm 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 k-th largest 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 expected 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 $empty(b) \ge m - t + 1$.

```
algorithm sample-sort
/* 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 tested or
              $empty(b) \ge m - t + 1$
9. return the $k$ black vertices with the largest degrees (handle ties if
   necessary)
```

Figure 6: Algorithm *sample-sort*

The overall algorithm is presented in Figure 6. Its main drawback is the need of a parameter s, on 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.

More precisely, 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 $b' \in B, b' \neq b$, which means that 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 k-th largest degree of the vertices in the result R. Figure 7 formally summarizes the algorithm.

```
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 $k$-th largest degree of the vertices in $R$ in the rest
    of the algorithm
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 $B_{min} \neq \emptyset$
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
```

Figure 7: Algorithm *switch-on-empty*

Example. Next, we illustrate SOE using the input graph in Figure 8 where B and W have 2 and 5 vertices, respectively. Assume $k = 1$ (i.e., we aim to solve the 1MCV problem), and the algorithm class considered is the random-probe class $\mathcal{A}_{\text{RAN}}$ (the case of the deterministic-probe class $\mathcal{A}_{\text{DET}}$ 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 algoritm probes another edge of b_1, for example, its edge with w_5. As the 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 (remaining) 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 happens to be 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. □

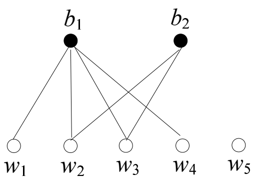


Figure 8: An example to illustrate SOE

Dealing with $k > n/2$. So far we have assumed that k is at most $n/2$. Now, we are ready to explain how to handle the case $k > n/2$. As mentioned in Section 3, this is equivalent to finding the $(n-k) < n/2$ vertices in B having the *lowest* degrees.

Let us consider the *complement* $\bar{G}$ of the input graph G. Specifically, $\bar{G}$ has the same vertex sets B and W as G. However, for each pair of black vertex $b \in B$ and white vertex $w \in W$, there is an edge in $\bar{G}$, if and only if b is *not* connected to w in G. In other words, a vertex b with degree $deg(b)$ in G has a degree $m - deg(b)$ in $\bar{G}$. It thus follows that the $n-k$ black vertices with the minimum degrees in G are *exactly the same* as the $n-k$ black vertices with the maximum degrees in $\bar{G}$.

Since the original kMCV problem on G with $k > n/2$ has been reduced to a k'MCV problem on $\bar{G}$ with $k' = n - k < n/2$, we can solve the k'MCV problem directly using the proposed algorithms SS and SOE. In fact, no modification is needed in those algorithms. The only change required is to simply *reverse* the outcome of each edge-probing query. Namely, if the query returns yes, the algorithms should take it as no, and vice versa.

Remark. Algorithm SOE simultaneously belongs to both the random-probe algorithm class $\mathcal{A}_{\text{RAN}}$ and the deterministic-probe algorithm class $\mathcal{A}_{\text{DET}}$, 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 $\mathcal{A}_{\text{RAN}}$. The reason is that, in the context of $\mathcal{A}_{\text{DET}}$, 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

In this section, we analyze the performance of the proposed algorithms SS and SOE. Since both of them belong to $\mathcal{A}_{\text{RAN}}$ and $\mathcal{A}_{\text{DET}}$, we will discuss the two algorithm classes separately in Sections 5.1 and 5.2, respectively.

5.1 The randomized algorithm class

Let us start with a property of all the algorithms $A \in \mathcal{A}_{\text{RAN}}$. 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 the following about $Q(u)$:

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

PROOF. Let X be the random variable whose expectation is captured by $Q(u)$. Namely, X is the number of queries that A must perform on b before seeing u empty edges of b. The distribution of X is a standard *negative hypergeometric distribution*, whose expectation is as given in the proposition. □

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 chosen without any query, algorithm SS cannot be instance optimal.*

PROOF. See the appendix. □

The theorem indicates that, while sampling is a natural idea to attack the kMCV 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. The theorem strongly implies that the correct sample size needs to be chosen *adaptively*, based on the degree distributions of the black vertices. This implication 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 result (black) vertices. Let t^* be the lowest degree of the vertices in R, or formally:

$$t^* = \min_{t \in R} deg(t). \tag{3}$$

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

We first point out two more properties of all algorithms $A \in \mathcal{A}_{\text{RAN}}$. 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 1. *At termination, for each non-result black vertex $b \in B - R$, it holds that $empty(b) \geq m - t_A + 1$.*

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 $k^* > k$:

LEMMA 2. *If $k^* > k$, then A has probed all the m edges of at least $k^* - k + 1$ black vertices in R.*

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 $g = |R_{tail}| - (k^* - k)$. Note that g must be positive.

The crucial observation is that $|S|$ must be smaller than g. Otherwise, assume $|S| \geq g$; then consider any g vertices, say b_1, b_2, ..., 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 $R_{tail} - S'$ have no more hidden solid edge. In this case, $R_{tail} - S'$ 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 $|R_{tail}| - |S| \geq |R_{tail}| - (g - 1) = k^* - k + 1$ vertices. □

The above two properties also hold for all algorithms that cannot predict whether a hidden edge is solid or empty. The next lemma states a property of algorithm SOE:

LEMMA 3. *SOE probes all the m edges of each vertex in R. For each vertex $b \in B - R$, its finds exactly $m - t^* + 1$ 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. □

We are ready to prove the main result of this paper:

THEOREM 2. *Let A_{opt} be the fastest algorithm in $\mathcal{A}_{\text{RAN}}$ for solving the kMCV problem on the input G, namely:*

$$A_{opt} = \{A \mid cost(A, G) \leq cost(A', G); A, A' \in \mathcal{A}_{\text{RAN}}\}.$$

For any $k \leq n/2$, the expected cost of SOE is at most $r \cdot cost(A_{opt}, G)$, where $r \leq 1 + \frac{k}{n-k}$.

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

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

respectively (the ordering is not important). For each $i \in [k^* + 1, n]$, 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 $A \in \mathcal{A}_{\text{RAN}}$, in order to find u empty edges of b_i.

For convenience, we denote algorithm SOE as A^*. By Lemma 3, the expected cost of A^* can be written as:

$$cost(A^*, G) = mk^* + \sum_{i=k^*+1}^{n} Q_i(m - t^* + 1). \tag{4}$$

Now consider the optimal algorithm A_{opt}. Define a random variable:

$$t_{opt} = \min_{b \in R} solid_{opt}(b). \tag{5}$$

where $solid_{opt}(b)$ is the number of solid edges of a vertex $b \in R$ that are found by A_{opt} at termination. Next, we focus on the event:

$$\omega : t_{opt} = x.$$

Note that as $solid_{opt}(b) \leq deg(b)$, it must hold that $x \leq t^*$. Under event ω, Lemma 1 indicates that A_{opt} probes in expectation at least $Q_i(m - x + 1)$ edges of b_i. Define function $C(x)$ to be the expected cost of A_{opt} *conditioned on* $t_{opt} = x$.

The rest of the proof will show that $r = cost(A^*, G)/C(x) \leq 1 + k/(n - k)$. This, together with $cost(A_{opt}, G) = \sum_x C(x) \cdot \Pr[x]$, will establish the theorem. We proceed in two cases, depending on the comparison of k^* and k:

Case 1: $k^* = k$. It holds that

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

Combining the above with Equation 4, we know

$$r \leq \frac{mk^* + \sum_{i=k^*+1}^{m} Q_i(m - t^* + 1)}{xk + \sum_{i=k^*+1}^{n} Q_i(m - x + 1)}.$$

Since $x \le t^*$, we have

$$r \le \frac{mk^* + \sum_{i=k^*+1}^{m} Q_i(m-x+1)}{xk + \sum_{i=k^*+1}^{n} Q_i(m-x+1)}.$$

leading to

$$r - 1 \le \frac{(m-x)k^*}{xk^* + \sum_{i=k^*+1}^{n} Q_i(m-x+1)}.$$

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

$$r - 1 \le \frac{(m-x)k^*}{xk^* + a(m-x)}.$$

where

$$a = \sum_{i=k^*+1}^{n} \frac{m+1}{ml_i+1}. \qquad (6)$$

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

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

Case 2: $k^* > k$. Lemma 2 indicates the existence of a set S of $k^* - k + 1$ vertices in R, such that A_{opt} must have probed all the m edges of each vertex in S. Hence:

$$C(x) \ \ge \ x(k-1) + m(k^*-k+1) + \sum_{i=k^*+1}^{n} Q_i(m-x+1)$$

Combining the above with Equation 4 gives:

$$\begin{aligned} r &\le \frac{mk^* + \sum_{i=k^*+1}^{m} Q_i(m-t^*+1)}{x(k-1) + m(k^*-k+1) + \sum_{i=k^*+1}^{n} Q_i(m-x+1)} \\ &\le \frac{mk^* + \sum_{i=k^*+1}^{n} Q_i(m-x+1)}{mk^* + \sum_{i=k^*+1}^{n} Q_i(m-x+1) - k(m-x)} \end{aligned}$$

where the last inequality used $x \le t^*$ and $x \le m$. Hence, decreasing both sides of the above inequality by 1 and applying Proposition 1, we have:

$$r - 1 \ \le \ \frac{k(m-x)}{mk^* + a(m-x+1) - k(m-x)}$$

where a is given in Equation 6. Again if $m = x$, then $r = 1 < 1 + k/(n-k)$. Otherwise, knowing $a \ge n - k^*$, we derive:

$$\begin{aligned} r - 1 &\le \frac{k(m-x)}{mk^* + (n-k^*)(m-x) - k(m-x)} \\ &\le \frac{k(m-x)}{n(m-x) - k(m-x)} = \frac{k}{n-k}. \end{aligned}$$

This completes the proof. □

The above theorem concerns $k \le n/2$. As mentioned in Section 4, the case of $k > n/2$ can be reduced to a k'MCV problem with $k' = n - k < n/2$ on the complement $\bar{G}$ of the input G. The proof of Theorem 3 also holds on $\bar{G}$. Thus, we arrive at the following general result:

COROLLARY 1. *The expected cost of SOE is at most $r \cdot cost(A_{opt}, G)$, where $r \le 1 + \frac{\min\{k,n-k\}}{\max\{k,n-k\}}$, and A_{opt} as defined in Theorem 2.*

The corollary has two significant implications:

- For any k, the value of r is *definitely lower than 2*. Hence, SOE is instance optimal. Furthermore, since SOE performs at least $km + n - 1$ queries on any input G, it follows that $\Omega(km + n)$ is a cost lower bound of *any* algorithm in the class $\mathcal{A}_{\text{RAN}}$.

- When $k = O(1)$ or $k = n - O(1)$, $r = 1 + O(1/n)$, namely, SOE is *nearly as fast as the optimal algorithm* in these two extremes. Recall that, in practice, $k = O(1)$ and $k = n - O(1)$ correspond to the important scenarios where users want to find the *top few* (e.g., 10) black vertices having the maximum and minimum degrees, respectively.

5.2 The deterministic algorithm class

Next, we extend the analysis of the previous subsection to the algorithm class $\mathcal{A}_{\text{DET}}$. We focus on only SOE because the instance optimality of SS in $\mathcal{A}_{\text{DET}}$ can be disproved using an argument similar to but much simpler than the proof of Theorem 1. For $\mathcal{A}_{\text{DET}}$, Proposition 1 obviously is not applicable; Lemmas 1-3, however, are still correct. We first give a theorem that is the counterpart of Theorem 2.

THEOREM 3. *Let A_{opt} be the fastest algorithm in $\mathcal{A}_{\text{DET}}$ for solving the kMCV problem on the input G, namely:*

$$A_{opt} = \{A \mid cost(A,G) \le cost(A',G); A, A' \in \mathcal{A}_{\text{DET}}\}.$$

For any $k \le n/2$, the cost of SOE is at most $r \cdot cost(A_{opt}, G)$, where $r \le 1 + \frac{k}{n-k}$.

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 A^*, k^*, t^*, b_i ($k^*+1 \le i \le n$) retain their meanings in the old proof.

Let Q_i^* ($k^*+1 \le i \le n$) be the number of edges of b_i that A^* has probed at termination. Q_i^* equals the position of the $(m-t^*+1)$-th white vertex (in the probing sequence) that has an empty edge with b_i. By Lemma 3, $cost(A^*, G) = mk^* + \sum_{i=k^*+1}^{n} Q_i^*$.

Define t_{opt}, $solid_{opt}(b)$, $C(x)$ in the same way as in the old proof. Let Q_i be the least number of edges that (all possible execution of) A_{opt} needs to probe for b_i, conditioned on $t_{opt} = x$. The crucial observation is that, since $x = t_{opt} \le t^*$, by Lemma 1, A_{opt} must have seen at least $m - x + 1 \ge m - t^* + 1$ empty edges of b_i. In other words, A_{opt} must have probed all the edges of b_i probed by A^*; hence:

$$Q_i^* \le Q_i. \qquad (7)$$

Let $r = cost(A^*, G)/C(x)$ and $a = \sum_{i=k^*+1}^{n} Q_i$. Note that $a \ge (n-k^*)(m-x+1)$. Assuming $m \ne x$ (same as in Theorem 2, if $m = x$, then $r = 1 < 1 + k/(n-k)$), we proceed in two cases:
Case 1: $k^* = k$. It holds that $C(x) \ge xk^* + \sum_{i=k^*+1}^{n} Q_i$. Hence

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

where the last inequality used Inequality 7. Hence:

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

Case 2: $k^* > k$. By Lemma 2, $C(x) \geq x(k-1) + m(k^* - k + 1) + \sum_{i=k^*+1}^{n} Q_i$. Thus

$$\begin{aligned} r &\leq \frac{mk^* + \sum_{i=k^*+1}^{n} Q_i^*}{x(k-1) + m(k^*-k+1) + \sum_{i=k^*+1}^{n} Q_i} \\ &\leq \frac{mk^* + a}{mk^* + a - k(m-x)} \end{aligned}$$

Hence:

$$\begin{aligned} r - 1 &\leq \frac{k(m-x)}{mk^* + a - k(m-x)} \\ &\leq \frac{k(m-x)}{mk^* + (n-k^*)(m-x+1) - k(m-x)} \\ &< \frac{k(m-x)}{n(m-x) - k(m-x)} = \frac{k}{n-k} \end{aligned} \tag{8}$$

which completes the proof. □

With the same argument leading to Corollary 1, we get:

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

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

6. EXPERIMENTS

In the sequel, we experimentally evaluate the performance of the proposed algorithms. Section 6.1 describes the data employed in our experimentation, and Section 6.2 clarifies how alternative methods will be compared. Then, Section 6.3 studies the environments where the kMCV problem can be settled much faster than probing all edges. Sections 6.4 and 6.5 evaluate our techniques in the random-probe and deterministic-probe algorithm classes, respectively.

6.1 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

$$c(d+1)^{-\gamma} \tag{9}$$

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 discussed 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 Equation 9 is:

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

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* [13]. 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[1]:

```
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 kMCV 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.

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 [27].

We aimed at finding the k actors that have the largest number of 2-hop relationships. This is a kMCV 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).

6.2 Methods

Since no previous solution is known for the kMCV problem, we concentrate on comparing the proposed algorithms *sample-and-sort* (SS) and *switch-on-empty* (SOE). The value of k will be varied from 1 to 100. Since the black vertex set B in all our data graphs

[1]This statement is essentially a *top-k dominating query*, which has been studied in [25, 29]. However, the solutions in [25, 29] 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 [13], transitivity does *not* hold on k-dominance.

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 6.3 and 6.4, we study the random-probe algorithm class $\mathcal{A}_{\text{RAN}}$, where an algorithm deploys the *probe-next* implementation in Figure 3. Section 6.5 investigates the deterministic-probe algorithm class $\mathcal{A}_{\text{DET}}$, where an algorithm applies the *probe-next* in Figure 4.

6.3 How pessimistic is the worst case?

If B and W have n and m edges respectively, solving a kMCV problem requires probing nm edges in the worst case. The objective of this subsection is to find out when it is possible to achieve a 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 in settling the 10MCV problem on each of these graphs.

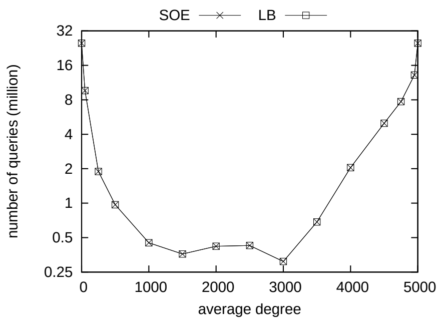


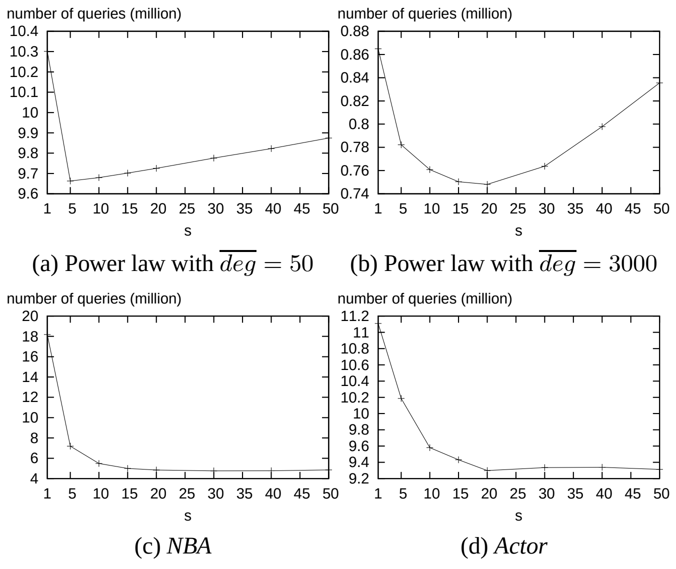
Figure 9: Impact of the average degree of black vertices

Figure 9 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 $\overline{deg}$ 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 9, the cost of SOE is very close to the lower bound.

6.4 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



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

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

Figure 10: Tuning the parameter s of algorithm SS

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 10 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.

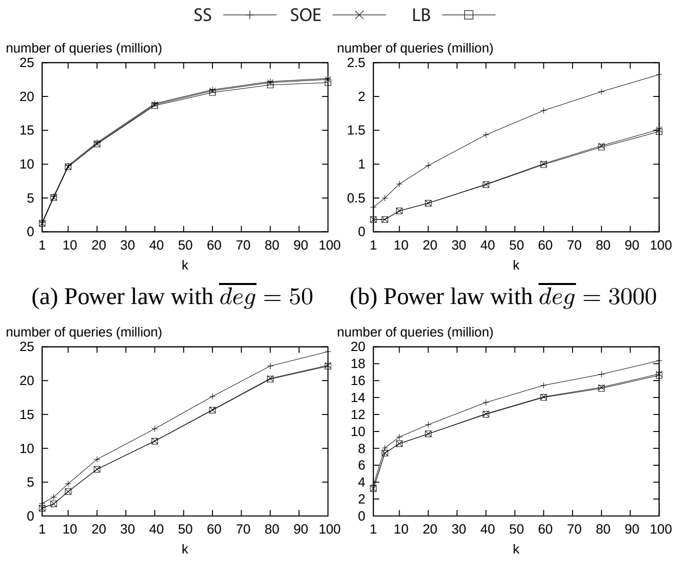
Scalability with k. We proceed to compare SOE and SS in kMCV computation by increasing k from 1 to 100. Figure 11 illustrates the results, as well as the lower bounds, on the same graphs in Figure 10. For benchmarking, remember that the worst-case cost is 25 million for power-law graphs, $16739^2 > 280$ 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 11a, 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.

6.5 Performance of deterministic-probe algorithms

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



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

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

Figure 11: Performance vs. k (random-probe class)

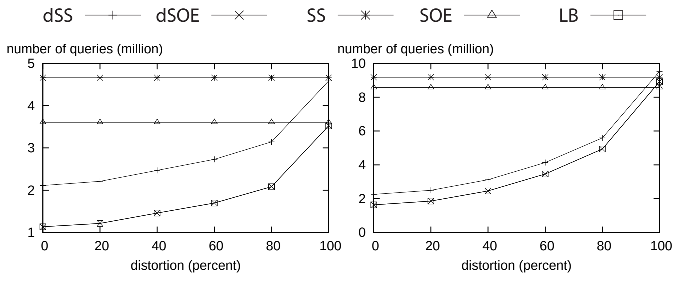
tion 0 ranks the white vertices in ascending order of their degrees (or equivalently, in descending order of how many empty edges they have). On the other extreme, a sequence with distortion 1 is simply a random permutation of the white vertices. In general, in a sequence with distortion d, the positions 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 $\mathcal{A}_{\text{RAN}}$, we refer to the version of SS (SOE) in the deterministic-probe class $\mathcal{A}_{\text{DET}}$ as dSS (dSOE). The parameter s of dSS is also set to 20, after a tuning process similar to Figure 10. Concerning 10MCV computation on *NBA*, Figure 12a 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 12b presents the 10MCV 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. Finally, the relative performance of dSS and dSOE is similar to the random-probe class reported in Figure 11. Also, dSOE is once again nearly optimal, leaving little room for further improvements.

7. CONCLUSIONS

This paper studied the *k most connected vertex* (kMCV) problem on hidden bipartite graphs, which has a large number of database applications in practice. The novelty of the problem is that (unlike many other graph problems) the edges are not explicitly given; instead, a unit cost must be paid to detect the presence of each edge. We presented an algorithm that is guaranteed to be instance optimal



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

Figure 12: Effects of distortion (deterministic-probe class)

in a broad class of algorithms. In other words, for any data input, our algorithm can be worse than the optimal algorithm (which remains unknown) by at most a constant factor, which can be shown to be at most 2. Furthermore, for small k (such as 10), we proved a much stronger result indicating that our solution is nearly as fast as the optimal algorithm.

We believe that *query processing in hidden graphs* is a promising research direction in the database area. Such graphs constitute a powerful way to model many problems in a large number of existing and emerging applications. The kMCV problem studied in this paper serves as the first step into this exciting topic. In fact, it is worth re-visiting many conventional graph problems due to the novel features of hidden graphs. The existing algorithms may not necessarily treat edge-probing as a cost-dominating operation, in which case potential improvements are possible.

8. REFERENCES

[1] P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. In *FOCS*, 2009.

[2] N. Alon, R. Beigel, S. Kasif, S. Rudich, and B. Sudakov. Learning a hidden matching. *SIAM J. Comput.*, 33(2):487–501, 2004.

[3] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM J. Comput.*, 15(2):211–227, 2002.

[4] N. Alon and A. Shapira. A characterization of the (natural) graph properties testable with one-sided error. *SIAM J. Comput.*, 37(6):1703–1727, 2008.

[5] N. Alon and A. Shapira. Every monotone graph property is testable. *SIAM J. Comput.*, 38(2):505–522, 2008.

[6] R. Angles and C. Gutiérrez. Survey of graph database models. *ACM Comp. Surv.*, 40(1), 2008.

[7] D. Angluin and J. Chen. Learning a hidden graph using O(logn) queries per edge. *JCSS*, 74(4):546–556, 2008.

[8] I. Baran and E. D. Demaine. 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, 2005.

[9] J. Barbay and E. Y. Chen. Convex hull of the union of convex objects in the plane: an adaptive analysis. In *CCCG*, 2008.

[10] T. C. Biedl, B. Brejová, E. D. Demaine, A. M. Hamel, A. López-Ortiz, and T. Vinar. Finding hidden independent sets in interval graphs. *Theo. Comp. Sci.*, 310(1-3):287–307, 2004.

[11] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *FOCS*, pages 93–102, 2002.

[12] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[13] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and

Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, pages 503–514, 2006.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[15] E. D. Demaine, D. Harmon, J. Iacono, D. Kane, and M. Pătraşcu. The geometry of binary search trees. In *SODA*, pages 496–505, 2009.

[16] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *SODA*, pages 743–752, 2000.

[17] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[18] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653–750, 1998.

[19] M. E. Houle and J. Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *ICDE*, pages 619–630, 2005.

[20] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB*, pages 754–765, 2003.

[21] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comp. Surv.*, 40(4), 2008.

[22] D. B. Kell. Screen idols: faster, smaller, cheaper and smarter. *Trends in Biotechnol*, 18:186–187, 2000.

[23] E. J. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[24] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, pages 281–290, 2001.

[25] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 30(1):41–82, 2005.

[26] K. Schnaitter and N. Polyzotis. Evaluating rank joins with optimal cost. In *PODS*, pages 43–52, 2008.

[27] P. Singla and M. Richardson. Yes, there is a correlation: - from social networks to personal behavior on the web. In *WWW*, pages 655–664, 2008.

[28] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Probabilistic top-k and ranking-aggregate queries. *TODS*, 33(3), 2008.

[29] M. L. Yiu and N. Mamoulis. Multi-dimensional top-dominating queries. *VLDB J.*, 18(3):695–718, 2009.

[30] M. Zhu, D. Papadias, J. Zhang, and D. L. Lee. Top-k spatial joins. *TKDE*, 17(4):567–579, 2005.

Appendix (proof of Theorem 1)

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 only need to 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 $s \le \lambda$, where λ is a constant.

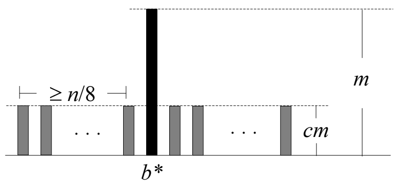


Figure 13: Illustration of $G_2(n,m)$

$G_2(n,m)$ is such that one black vertex b^* has degree m, and the other black vertices all have degree cm, where c, which will be determined later, is a constant close to 1 from below. Figure 13 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 (clearly, b^* is definitely in S). The choice of c will make sure that $|S| \ge n/4$ with probability at least $1/2$ (later we will argue that such c always exists). Assuming $|S| \ge n/4$, 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^*. 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 1MCV 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 $b \in B, b \ne b^*$, plus all the m edges of b^*. By Proposition 1, 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 $b \in B, b \ne b^*$ 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 $p \ge c^s - \epsilon \ge c^\lambda - \epsilon$, where $\epsilon > 0$ is an arbitrarily small constant. By Hoeffding's inequality[2], $X \ge (n-1)/2 > n/4$ with probability at least $1 - \exp(-2(n-1)(p-0.5)^2)$, which is at least 0.5 if $p \ge (\frac{\ln\sqrt{2}}{n-1})^{0.5} + 0.5$. To ensure this, it suffices to guarantee $c^\lambda \ge (\frac{\ln\sqrt{2}}{n-1})^{0.5} + 0.5 + \epsilon$. 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 must be $\omega(1)$. This, however, conflicts with the requirement of s on $G_1(n,m)$.

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