Formal Barriers to Simple Algorithms for the Matroid Secretary Problem

Maryam Bahrani[1], Hedyeh Beyhaghi[2(✉)], Sahil Singla[3], and S. Matthew Weinberg[4]

[1] Columbia University, New York, USA
m.bahrani@columbia.edu
[2] Toyota Technological Institute at Chicago, Chicago, USA
hedyeh@ttic.edu
[3] Georgia Institute of Technology, Atlanta, USA
ssingla@gatech.edu
[4] Princeton University, Princeton, USA
smweinberg@princeton.edu

Abstract. Babaioff et al. [4] introduced the matroid secretary problem in 2007, a natural extension of the classic single-choice secretary problem to matroids, and conjectured that a constant-competitive online algorithm exists. The conjecture still remains open despite substantial partial progress, including constant-competitive algorithms for numerous special cases of matroids, and an $O(\log \log \text{rank})$-competitive algorithm in the general case.

Many of these algorithms follow principled frameworks. The limits of these frameworks are previously unstudied, and prior work establishes only that a handful of particular algorithms cannot resolve the matroid secretary conjecture. We initiate the study of impossibility results for frameworks to resolve this conjecture. We establish impossibility results for a natural class of greedy algorithms and for randomized partition algorithms, both of which contain known algorithms that resolve special cases.

Keywords: Secretary problem · Matroids · Optimal stopping theory · Graph theory · Greedy algorithms

1 Introduction

The problem of finding a max-weight basis of a matroid $\mathcal{M} = (V, \mathcal{I})$[1] is central in the field of combinatorial optimization (see books [18,21,23]). More specifically, each element $e \in V$ has a weight $w(e) \geq 0$, and the goal is to find the set $S \in \mathcal{I}$ maximizing $w(S) := \sum_{e \in S} w(e)$. Seminal works of Rado, Gale, and Edmonds establish that the following simple greedy algorithm finds a max-weight basis of a

[1] Given a finite set V and a family of subsets of V called I, we say $\mathcal{M} = (V, \mathcal{I})$ is a matroid if it satisfies (i) $\emptyset \in \mathcal{I}$, (ii) *Hereditary Property (downwards closed):* $\forall T \subseteq S \subseteq V$, set $S \in \mathcal{I}$ implies $T \in \mathcal{I}$, and (iii) *Exchange Property:* For any $S, T \in \mathcal{I}$ where $|S| > |T|$, there exists some $x \in S$ such that $T \cup \{x\} \in \mathcal{I}$.

M. Feldman et al. (Eds.): WINE 2021, LNCS 13112, pp. 280–298, 2022.
https://doi.org/10.1007/978-3-030-94676-0_16

matroid $(V, \mathcal{I})$: Initialize $A = \emptyset$, then process the elements of V in decreasing order of $w(e)$, adding to A any element such that $A \cup \{e\} \in \mathcal{I}$ [8,11,22]. In fact, if for some $(V, \mathcal{I})$ this algorithm is optimal for all $w(\cdot)$, then $(V, \mathcal{I})$ must be a matroid.

While simple, this algorithm still requires knowledge of all weights up front. Motivated by applications to mechanism design and other online problems [3, 13], recent work considered the problem in an online setting: elements are still processed one at a time and are immediately and irrevocably accepted or rejected upon processing, but an element's weight remains unknown until the element is processed. In particular, the algorithm does not have control over the order of elements and therefore cannot run the simple greedy algorithm.

For a fully adversarial order, it's folklore that the best algorithm can do no better than simply selecting a random element. Babaioff et al. [4][2] therefore introduced the *Matroid Secretary Problem* (MSP), where elements arrive in a *uniformly random order* (while the weight function is still adversarial). This formulation extends the classic single-item secretary problem [7].

Consider an algorithm $\mathcal{A}$ for the matroid secretary problem on matroid $\mathcal{M}$. Let OPT be the max-weight basis of $\mathcal{M}$ under $w(\cdot)$, and let ALG be the set of elements chosen by $\mathcal{A}$ (under $w(\cdot)$). The following notion of *utility-competitiveness* for a matroid secretary algorithm was studied in Babaioff et al. [4].

Definition 1 (Utility-Competitive). *An algorithm $\mathcal{A}$ is* α-utility-competitive *if $\mathbb{E}[w(\mathsf{ALG})]/w(\mathsf{OPT}) \geq \alpha$, where the expectation is over the randomness of the arrivals and any internal randomness of algorithm $\mathcal{A}$.*

In the same paper that introduced the matroid secretary problem, Babaioff et al. [4] conjecture that there is a constant-utility-competitive algorithm. The stronger form of the conjecture is that this constant is $1/e$.

Conjecture 1 (Matroid Secretary). There is an $\Omega(1)$-utility-competitive algorithm for the matroid secretary problem.

Despite extensive follow-up work, this conjecture still remains open. Many constant-utility-competitive algorithms have been proposed for specific classes of matroid (see related work in Sect. 1.3). For general matroids, however, the best known algorithms are $1/O(\log\log r)$-competitive [9,20] (here, r denotes the *rank* of the matroid, which is the size of the largest set in $\mathcal{I}$).

As the only known lower bound, even for general matroids, is the same $1/e$ from the classic single-item setting, and because Dynkin's algorithm guarantees a stronger property that the heaviest element is selected with probability $1/e$, the following stronger notion of *probability-competitive* algorithms has been also studied [14,25].

Definition 2 (Probability-Competitive). *An algorithm $\mathcal{A}$ is* α-probability-competitive *if for all $i \in \mathsf{OPT}$ it satisfies that $\mathbb{P}[i \in \mathsf{ALG}] \geq \alpha$.*

Note that probability-competitiveness is a stronger notion than utility-competitiveness, since the former implies the latter with the same competitive

[2] Conference version [5] appeared in 2007.

ratio. Soto et al. [25] showed that many (but not all) existing utility-competitive algorithms can be extended to obtain probability-competitive algorithms. This results in the following more ambitious conjecture. Again, the stronger version conjectures that this constant is $1/e$.

Conjecture 2. There is an $\Omega(1)$-probability-competitive algorithm for the matroid secretary problem.

Progress on both conjectures has been slow. Indeed, even the strong version of Conjecture 2 remains plausible, while the best utility-competitive algorithms have stalled at $1/O(\log \log r)$ [9,20]. One thesis motivating our work is that the community currently lacks structure for narrowing a search among numerous promising approaches. Existing algorithms for special cases indeed follow principled frameworks, but these frameworks are quite flexible and it remains unknown which (if any) of them might produce a resolution to either conjecture.

One particularly enticing possibility is that a simple "greedy-like" algorithm might even work. Note that such algorithms indeed work in the Free-Order model [16], or for the related Matroid Prophet Inequality [17], or for special cases of the Matroid Secretary Problem [2,7]. There are numerous variants of "greedy" algorithms, though. While many particular variants are known to fail on the same "hat graph" [4], there is previously no approach to quickly tell whether a novel greedy variant is already known to fail.

In this work, we rigorously consider two general classes of algorithms, and prove super-constant lower bounds on what they can achieve for the matroid secretary problem. This both helps explain why these types of algorithms have faced difficulty extending beyond the special cases for which they were originally designed, and helps guide future work towards precisely the variants that merit further exploration.

1.1 Greedy Algorithms

Since finding the max-weight basis of matroids without requiring irrevocable commitments can be done exactly by the simple greedy algorithm, the class of greedy algorithms is a very natural candidate for solving the Matroid Secretary Problem. We consider a large family of "greedy-like" algorithms. We define three natural properties that a greedy algorithm might have, and establish that any algorithm satisfying these properties cannot be constant-utility-competitive (Theorem 2). We postpone formal statements of the properties until Sect. 3, but overview them here: (i) the algorithm should reject the first T fraction of elements, (ii) the algorithm at all times stores an independent set I containing all accepted elements and no elements rejected after T, (iii) an element is accepted if and only if it improves the max-weight basis of I after contracting the accepted elements.[3] Note that this a general framework rather than a fully-specified algo-

[3] To rephrase (iii), an element e is accepted iff after contracting the accepted elements (not including e), the max-weight basis of the restricted matroid to $I \cup \{e\}$ is heavier than the max-weight basis of the restricted matroid to I (the latter being exactly the weight of I since I is independent).

rithm, since it allows for the algorithm to choose I (it need not be the max-weight basis after contracting the accepted elements, just some independent set).

In Sect. 3 we overview several existing algorithms that fit this framework, and Theorem 2 unifies a proof that none of these algorithms (or many hypothetical ones) can be constant-utility-competitive. Our lower bound construction is a variant of the well-known "hat graph", which has been known since [4] to be problematic for greedy-like algorithms. So our main contribution is not this construction itself, but rather a formalization of precisely the class of greedy algorithms for which this graph is problematic.

Main Result 1 (Informal, see Theorem 2). *No Greedy algorithm (as per Algorithm 1) is constant-utility-competitive.*

We emphasize that while the hat graph itself is not a novel construction, our proof is quite distinct (and more involved) from prior work as it must rule out a broad class of algorithms rather than just a single one.

1.2 Randomized Partition Algorithms

Another class of particularly simple algorithms are *randomized partition algorithms*:

1. Before looking at any weights, (perhaps randomly) partition all the elements[4] into parts S_i.
2. Within each part, run Dynkin's algorithm.
3. Output the union of the selected elements.

Note that these algorithms are allowed to use *any randomized* partition. The elegant $1/(2e)$-approximation of Korula and Pal for graphic matroids[5] is a randomized partition algorithm [19]. Their algorithm is utility-competitive, but not probability-competitive. Soto et al. [25] recently designed a different constant probability-competitive algorithm for graphic matroids. While their algorithm is still quite elegant, it is perhaps not quite as simple as randomized partition algorithms. It is also worth noting that algorithms such as [9,20] follow a more general framework, where the algorithm in step one looks at the weights before partitioning and step two is not necessarily Dynkin's single-choice algorithm (but perhaps some simple greedy algorithm). This raises the question whether the novel development beyond [19] is necessary to achieve probability-competitive algorithms? Our second main result answers this question: no randomized partition algorithm can be constant-probability-competitive (or even $\omega(n^{-1/8})$-probability-competitive).

[4] We consider the known matroid setting where the matroid is known but the weights are revealed one-by-one.

[5] Given a graph with edges E, a graphic matroid $(E, \mathcal{I})$ is defined with $\mathcal{I}$ consisting of all subsets of edges that do not contain a cycle.

Main Result 2 (Informal, see Theorem 4). *No Randomized Partition algorithm is constant-probability-competitive.*

Our construction witnessing Theorem 4 is also a graphic matroid, although it is unrelated to the hat graph (and to the best of our knowledge, novel). Note that our proof cannot be extended to utility-competitive algorithms since we know [19] is a constant-utility-competitive randomized partition algorithm for graphic matroids.

1.3 Related Work and Brief Summary

There is a *substantial* body of work on random-order problems for matroids (the Matroid Secretary Problem [4]) and for several other discrete optimization problems; we will not attempt to overview it (e.g., see [6,12]). Here, we will briefly repeat the most related works.

Our work takes first steps towards characterizing classes of algorithms which might resolve the Matroid Secretary Problem. We focus on the simplest classes of algorithms which previously succeeded in special cases or for related problems, Greedy [16,17] or Randomized Partition [19], and study the limits of these classes. First, we consider extremely simple greedy algorithms. A specific instantiation of this class of algorithms was shown to fail on a now-canonical "hat graph" in [4], but related algorithms known to succeed in the Free-Order Model [1,16], and in the related Matroid Prophet Inequality [17]. In addition, Dynkin's algorithm and the Optimistic algorithm for k-uniform matroids of [2] fit this model. Our Theorem 2 shows that no Greedy algorithm is constant-utility-competitive for all matroids. Second, we consider probability-competitive algorithms, formally considered in [25], and related to the ordinal model considered in [14]. Soto et al. [25], in particular, develop several probability-competitive algorithms for core settings such as graphic, transversal, and laminar matroids. Our work asks whether the extremely simple algorithms previously developed in [19] can match these stronger probability-competitive guarantees, and we show in Theorem 4 that the answer is no.

2 Preliminaries

The Matroid Secretary Problem (MSP) is defined as:

1. There is a matroid $\mathcal{M} = (V, \mathcal{I})$, and weight function $w(\cdot) : V \to \mathbb{R}_{\geq 0}$. Matroid $\mathcal{M}$ is fully-known to the algorithm in advance.[6] Function $w(\cdot)$ is initially completely unknown to the algorithm.

[6] We are not concerned with computational efficiency of our algorithms in this work (our lower bounds are unconditional), so we will not stress about the precise format in which access to the matroid is given. To be concrete, one access model is that the algorithm has oracle access to $\mathcal{I}$ (query a set S and learn whether or not $S \in \mathcal{I}$). To the best of our knowledge, most algorithms previously considered for MSP are polytime given oracle access to $\mathcal{I}$.

2. Initially, the set of accepted elements, A, is empty. Elements of V arrive in a uniformly random order. When an element $i \in V$ arrives, the algorithm learns its weight $w(i)$, and must make an immediate and irrevocable decision whether or not to accept it (adding it to A). The algorithm must maintain $A \in \mathcal{I}$ at all times.
3. If set A is selected, the algorithm achieves payoff $\sum_{i \in A} w(i)$.

We will abuse notation and use $w(S) := \sum_{i \in S} w(i)$. Because $w(\cdot)$ is fixed, the offline optimum is the max-weight basis: $\text{MWB}(\mathcal{M}) := \arg\max_{S \in \mathcal{I}} \{w(S)\}$.[7] We will also use standard matroid notation such as *restriction*: the matroid $\mathcal{M}|_S$ is the matroid $\mathcal{M}$ restricted to S, and has ground set S and independent sets $\mathcal{I}|_S := \{T \cap S \mid T \in \mathcal{I}\}$. We also discuss matroid *contractions*: the matroid $\mathcal{M} \setminus S$ is the matroid $\mathcal{M}$ contracted by S, and has ground set $V \setminus S$ and independent sets $\mathcal{I} \setminus S := \{T \mid T \cup S \in \mathcal{I}\}$. When $\mathcal{M}$ is clear from context, we will also (slightly) abuse notation and write $\text{MWB}(T) := \text{MWB}(\mathcal{M}|_T)$.

We will later reference Dynkin's $1/e$-probability-competitive algorithm for selecting a single item, i.e., a 1-uniform matroid: (1) Reject the first $T = \text{Binom}(n, 1/e)$ elements and call this the *sampling stage*. (2) Afterwards, accept an element i iff it is the heaviest element seen so far.

Theorem 1 [7]**.** *Dynkin's algorithm is* $1/e$*-probability-competitive for* 1*-uniform matroids, this is optimal.*

3 Greedy Algorithms

Because matroids are exactly the constraints for which the simple greedy algorithm is optimal, greedy-like algorithms are a natural family to consider as candidates for resolving the Matroid Secretary Problem. Indeed greedy-like algorithms solve the related Matroid Prophet Inequality [17], Matroid Secretary in the free-order model [1,16], and special cases of Matroid Secretary [2,7]. In this section, we give an impossibility result for certain greedy algorithms. This helps unify counterexamples for related algorithms, and also helps narrow future research towards algorithms which have hope of resolving the Matroid Secretary Problem.

3.1 A Class of Greedy Algorithms

We now define a natural framework of greedy algorithms for the Matroid Secretary Problem (Algorithm 1). Without loss of generality, we consider the continuous arrival setting, where each element $e \in V$ arrives at a time $t(e)$ independently and uniformly drawn from $[0, 1]$. We refer by V_t to the set of elements that arrive (strictly) before t, and by A_t to the set of elements accepted by the algorithm (strictly) before time t.

[7] In this work, we assume for simplicity that the max-weight basis is unique. In case of ties, we tie-break by choosing the lexicographically-earlier basis.

Algorithm 1. Greedy Algorithm for the matroid secretary problem

We define a *greedy algorithm* as one that satisfies the following properties:

(i) Reject (but store) elements that arrive before T (sampling stage). Denote $S := V_T$ to emphasize this.
(ii) At all times t, maintain an independent set I_t such that:
  – I_t contains all accepted elements and no elements which were rejected after T, i.e. $A_t \subseteq I_t \subseteq A_t \cup S$.
  – At all times t, I_t spans V_t.
(iii) Accept e if and only if $e \in \text{MWB}((\mathcal{M} \setminus A_{t(e)})|_{I_{t(e)} \cup \{e\}})$ (and $t(e) > T$). That is, accept e if and only if it is in the max-weight basis of $I_{t(e)} \cup \{e\}$ *after contracting by* $A_{t(e)}$.

Before getting into our results, it is helpful to understand why Algorithm 1 is a class of algorithms (rather than a fully-specified algorithm). The reason is that the algorithm has flexibility in which subset of S to include in I_t (but it must include A_t, and must span V_t). The restriction is that the algorithm does not know which element might arrive at time t, nor its weight, when setting I_t. Furthermore, the algorithm can choose the length of the sampling stage T.

It is also helpful to see how this framework captures (or doesn't capture) existing greedy-like algorithms:

– Dynkin's algorithm (with $T = 1/e$) fits this framework. But so do suboptimal algorithms (e.g., accept the first element after T which exceeds the 5^{th}-highest sample. Or even accept an element which arrives at time $t > T$ iff it exceeds the $(\lfloor 5t/T \rfloor)^{\text{th}}$-highest sample).
– The Optimistic Algorithm for k-uniform matroids of [2] fits this framework. The algorithm maintains a list U, initially the k heaviest elements of S. If e exceeds the lightest element in U, it is accepted, and the lightest element of U is removed. In our language, this has $I_t := A_t \cup U$ at all times.
– There is a natural extension of the Optimistic Algorithm to all matroids, which was previously considered in [4].
– A related Pessimistic Algorithm (similar to the rehearsal algorithm for the related k-uniform prophet inequality of [1]) for k-uniform matroids fits this framework. The algorithm also maintains a list U, initially the k heaviest elements of S. If e exceeds the lightest element in U, it is accepted, but the *heaviest element of* U *lighter than* e is removed. In our language, this again has $I_t := A_t \cup U$ at all times (but U is updated differently to the previous bullet).
– The Virtual Algorithm for k-uniform matroids of [2] does *not* fit this framework. The algorithm accepts an element e if and only if e is one of the heaviest k elements so far *and* the k^{th}-heaviest element of $V_{t(e)}$ is in S (i.e., e is accepted if and only if it "kicks out a sample" from the top k so far). This is because the algorithm needs to remember rejected elements in order to properly keep track of the k^{th}-heaviest element so far, and whether it was a sample.

Observe finally that all of the algorithms above (which fit the framework) further have the following. First, if an element is rejected (after T), it is forgotten forever, and the algorithm proceeds as if the element had never existed in the first place.[8] Similarly, once an element e is accepted, the algorithm updates $\mathcal{M}$ by contracting by e, and then proceeds identically as if the true matroid had been $\mathcal{M} \setminus \{e\}$ the whole time.[9] These attributes are shared by the matroid prophet inequality of [17], and initially drove our formulation.

With an understanding of Greedy algorithms in hand, we now state our main result.

Theorem 2. *Any algorithm satisfying the 3 properties of Algorithm 1 cannot be constant-utility-competitive.*

3.2 Hard Instance: The Hat

In this section, we will study a *hat graph* which drives our impossibility result. The hat has a special element which is significantly heavier than the sum of all others, and thus any algorithm with a good utility-competitive ratio must accept it. Furthermore, this special element appears in many small circuits, so the algorithm must not accept the remaining elements of any of these circuits prior to the arrival of the heavy element (otherwise, the heavy element cannot be accepted when it arrives). The hat was used in [4] as a counterexample against a particular greedy algorithm; and variants of the graph have been informally known to be problematic for "greedy-like" algorithms. However, prior to our work there was no formal classification of "greedy-like".

The hat on $n+2$ vertices is a collection of n triangles, all sharing the same edge. Formally, an undirected graph (V, E) is a hat if $V = \{a, b, v_1, \ldots, v_n\}$ for some $n > 0$, and $E = \{\{a, b\}\} \cup \{e_i = \{a, v_i\} : i \in [n]\} \cup \{e'_i = \{b, v_i\} : i \in [n]\}$. Several weight assignments to the edges of the hat can serve as counterexamples to the algorithms considered in this section, but we consider a particular weight assignment for ease of exposition (as we only need one counterexample). We define this weight function $w : E \to \mathbb{R}_{\geq 0}$ to maintain the following ordering of the edge weights: $w(e_1) > \ldots > w(e_n) > w(e'_1) > \ldots > w(e'_n)$. Furthermore, $w(\{a, b\})$ is much larger than the sum of the weights of all other edges. We will refer to $\{a, b\}$ as the *infinity edge*, and we refer to its arrival time as $t_\infty := t(\{a, b\})$ to emphasize this. Additionally, we consider the drawing of the hat in the plane as shown in Fig. 1, where e_i is to the left of e_j for $i < j$, and e_i is above e'_i for all i. Accordingly, we will sometimes refer to the relative position of edges to imply a relation between their relative weights.

[8] But, the framework is rich enough to also allow for algorithms which update I_t as they reject an element. This makes impossibility results stronger.

[9] The framework is rich enough to allow for algorithms which update I_t based on A_t, rather than just $\mathcal{M} \setminus \{A_t\}$, which again just makes impossibility results stronger.

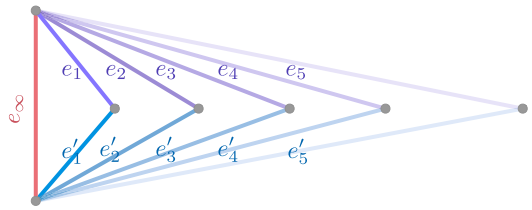

Fig. 1. A hat on seven vertices. All purple edges $(e_1, \ldots, e_5)$ are heavier than all blue edges $(e'_1, \ldots, e'_5)$, and e_∞ is significantly heavier than all other edges. Within each color, darker edges are heavier. (Color figure online)

We call the pair of edges (e_i, e'_i) the i-th *claw*. Recall that any algorithm satisfying the 3 properties listed in Sect. 3.1 has memory limited to an independent set I_t. At any time t, given the history of arrivals and the algorithm's past decisions, we can classify the claws into one of 9 kinds in $\{-, A, S\}^2$. The first character in the pair describes the state of the top edge e_i, and the second character describes the state of the bottom edge e'_i. S refers to an edge that is in I_t and arrived in the sampling stage. A refers to an edge that has been accepted by the algorithm (and is therefore in I_t). $-$ refers to any edge that is not in I_t. For example, if the i-th claw is of type $(S-)$ at some time t, it means that $t(e_i) < T$, $e_i \in I_t$, and $e'_i \notin I_t$. Figure 2 illustrates these claws.

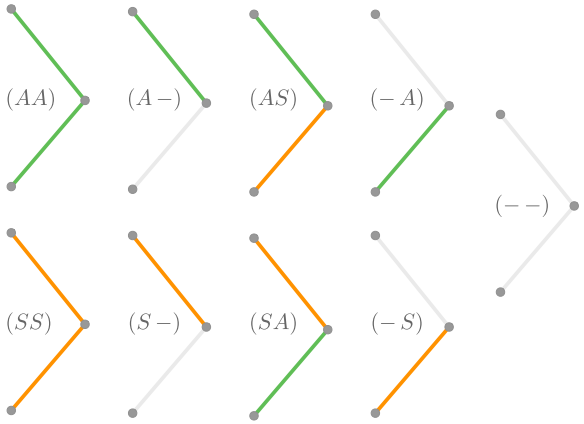

Fig. 2. All possible kinds of claws at any time t. S refers to sample edges in I_t (drawn in orange), A refers to an accepted edge in I_t (drawn in green), and $-$ refers to any other edge (drawn in gray). (Color figure online)

We next state a few lemmas about different classes of claws and their implications about the performance of the algorithm. Since the infinity edge weighs significantly more than other edges combined, we say the algorithm "loses" (i.e.,

fails to have a constant utility-competitive ratio) if it fails to accept the infinity edge. Conversely, the algorithm "wins" if it accepts the infinity edge. Our first observation characterizes the exact scenarios in which the algorithm loses. All missing proofs can be found in the full version.

Observation 3 (Loss condition). *The algorithm loses iff there is an* (AA) *claw before* t_∞.[10]

The next lemma specifies the unique blocking structure that would prevent the loss-inducing (AA) claws from forming. Our analysis focuses on the case of a $(-A)$ claw becoming a (AA) claw, as these events are significantly more likely than a $(A-)$ claw turning into an (AA) claw, and suffice for our analysis.

Lemma 1 (Blockers and Protection). *Suppose there is no* (AA) *claw yet. Consider a* $(-A)$ *claw whose upper edge is about to arrive. The upper edge is accepted iff there is no* (SA) *claw to its left. For this reason, we will refer to* (SA) *as the* blocker. *We say that the algorithm is* protected *at time* t *if there is a blocker in* I_t.

Importantly, note that there can be *at most one blocker in* I_t, as two blockers form a cycle. So we can unambiguously refer to *the* blocker at any time t. A blocker's effectiveness is a function of its location: Blockers far to the left "protect" more claws and are therefore more effective.

With this language in mind, we can reframe the algorithm's objective, while working within the Greedy framework. The algorithm loses whenever the upper edge of a $(-A)$ claw arrives without a blocker to its left. So the algorithm would like to maintain a blocker in I_t as far to the left as possible.[11] So the remainder of this section studies decisions the algorithm can make (again, within the Greedy framework) to include blockers far to the left. Lemma 2, however, establishes that we cannot create a new blocker without destroying our old one first (thereby going "unprotected" for some period).

Lemma 2. *If the lower edge of an* $(S-)$ *arrives at time* t *and* I_t *has a blocker, this edge will not be accepted.*

Lemma 2 means that the algorithm faces a tradeoff. If I_t has a blocker, it is safe from accepting the upper edge of a $(-A)$ claw *to its right* at time t. But, the algorithm *cannot move its blocker to the left, even if the lower edge of an* $(S-)$ *arrives during this interval*. Alternatively, the algorithm may not have a blocker during I_t. In that case, the algorithm can possibly accept a good blocker, if one happens to arrive at time t. But, the algorithm is at risk of accepting the upper edge of a $(-A)$ claw that arrives at time t *no matter its location*, because I_t has no blockers at all.

[10] Babaioff et al. [4] used the same graph as a counterexample to a special case of our greedy algorithm, also relying on this observation. Our lemmas are otherwise new, and necessary since we rule out a much larger class of greedy-like algorithms.

[11] Note that an arbitrary algorithm can simply decide to violate the properties defining Greedy. Our goal is to analyze Greedy algorithms, which must fit this framework.

3.3 Main Result: Ruling out all Greedy Algorithms

Armed with a better understanding of some properties of the hat structure, we are ready to prove Theorem 2, which states that greedy algorithms fail to be α-utility-competitive for any constant α.

We give a detailed proof sketch below, and defer calculations to the full version. We first repeat the main intuition: The algorithm's goal is to not accept any (AA) claw before t_∞ (Observation 3). To do so, the algorithm *must* make sure I_t includes a blocker to the left of every $(-A)$ whose upper edge arrives at time $t < t_\infty$ (Lemma 1). We can order potential blockers $(S-)$ by the arrival times of their lower edges, each of which is uniformly distributed in $[T, 1]$. Therefore, it is unlikely that a blocker far to the left arrives very early.

The algorithm can try to start with a mediocre blocker and improve it over time by accepting blockers further to the left as they arrive. The caveat is that due to Lemma 2, *blocker improvements are only possible in unprotected periods*, during which *any* arriving upper edge of $(-A)$ claws is accepted. Therefore, the algorithm faces a trade-off: Forming a more effective blocker costs more unprotected time. Importantly, the algorithm does not know whether the next arriving edge will be part of a potential blocker, or part of an $(-A)$.

In order to show that the algorithm fails, we show that with high probability there will be a (AA) claw before the arrival of the infinity edge. Specifically, we show that with high probability, an $(-A)$ claw becomes (AA) in an interval of length $\ell = n^{-0.1}$ after T, which is with high probability before the arrival of the infinity edge.

We now get into details of our proof approach. We first choose a parameter $x \in [n]$ (thinking of the claws as labeled 1 through n from left to right). We will undercount the algorithm's failure, noting that it fails whenever any of the following happens:

- The upper edge of some $(-A)$ to the left of x arrives during $[T, T+\ell]$, *and* I_t does not include any blocker to the left of x for any $t \in [T, T+\ell]$.
- The upper edge of some $(-A)$ arrives at an unprotected $t \in [T, T+\ell]$.

In other words, we are zeroing in on two potential sources of failure: the upper edge of *any* $(-A)$ claw could arrive during an unprotected time, or the upper edge of an $(-A)$ claw to the left of x could arrive before the algorithm accepts a blocker to the left of x. Note that these are very narrow possibilities for failure, but they suffice for our analysis.

So there are three probabilities to analyze. The first part of the first bullet is independent of the algorithm,[12] and simply considers the probability that the upper edge of a $(-A)$ to the left of x arrives during $[T, T+\ell]$.

Lemma 3. *With probability at least* $1 - 2^{-\ell^2 x/2}$*, the upper edge of a* $(-A)$ *claw to the left of* x *arrives between* T *and* $T+\ell$*.*

[12] Recall that the first edge of a $(--)$ claw to arrive must always be accepted since I_t must span V_t.

The next two probabilities are significantly more involved, as they consider decisions made by the algorithm. Note that the algorithm can decide *adaptively* when to go unprotected, based on the current ratio of $(-A)$s (potential (AA)s) versus $(S-)$s (potential blockers) to the left of x. To this end, we will let the algorithm adaptively choose any (measurable) subset of $[T, T+\ell]$ to go unprotected, and let y denote the total measure of this interval.[13] y captures the aforementioned tradeoff: small y means that the algorithm is likely to fail bullet one, while large y means the algorithm is likely to fail bullet two. Lemma 4 quantifies the cost of keeping y small, lowerbounding the probability of the second part of the first bullet.

Lemma 4. *Conditioned on the upper edge of a $(-A)$ claw to the left of x arriving between T and $T+\ell$ (i.e. Lemma 3 happening), any greedy algorithm which goes unprotected for a total measure of y during $[T, T+\ell]$ fails to accept a blocker to the left of x with probability at least:*

$$(1 - -2x\ell e^{-\frac{2x}{3}})(1-y)^{4x}.$$

Finally, we analyze the second bullet, lower bounding the probability that the upper edge of a $(-A)$ claw (anywhere) arrives during a period when the algorithm is unprotected (while the precise form is complicated, recall the intuition that as y gets larger, the probability of this particular bad event goes up, and y is at most ℓ):

Lemma 5. *Any greedy algorithm which goes unprotected for a total measure of $y \geq n^{-0.4}/2$ during $[T, T+\ell]$ has the upper edge of a $(-A)$ claw arrive during an unprotected t with probability at least:*

$$1 - \left(1 - \frac{2y - n^{-0.4}}{2\ell}\right)^{\frac{n^{0.6}(4\ell - n^{-0.4})}{32\ell^2}}.$$

Finally, we just need to combine the three bounds in Lemmas 3, 4, 5. We will choose a value of ℓ and x for the analysis, and then the algorithm (knowing x) can adaptively allocate the unprotected intervals within $[T, T+\ell]$ for a total measure of y. More formally, we let $f(y) = (1-2x\ell e^{-\frac{2x}{3}})(1-y)^{4x}\left(1 - (\frac{1}{2})^{\ell^2 x/2}\right)$ denote the lowerbound on failure probability derived in Lemma 4. Furthermore, we let

$$g(y) = \begin{cases} 1 - \left(1 - \frac{2y - n^{-0.4}}{2\ell}\right)^{\frac{n^{0.6}(4\ell - n^{-0.4})}{32\ell^2}}, & y \geq \frac{n^{0.4}}{2}; \\ 0, & y < \frac{n^{0.4}}{2}. \end{cases}$$

The first case follows from Lemma 5, and setting g to 0 elsewhere only strengthens our lower bound. Overall, the algorithm fails with probability at least $\min_y \{\max\{f(y), g(y)\}\}$. The next lemma sets parameters to lower bound this expression.

[13] The algorithm does not need to commit to the value of y in advance or choose it deterministically.

Lemma 6. *When* $x = n^{0.3}$ *and* $\ell = n^{-0.1}$*, we have*

$$\lim_{n\to\infty} \min_{y\in[0,\ell]} \{f(y), g(y)\} = 1.$$

The proof of Theorem 2 now follows from the four lemmas of this section.

4 Randomized Partition Algorithms

This section is devoted to a class of algorithms based on partition matroids. These are generalizations of an algorithm by Korula and Pal [19] for the secretary problem on graphic matroids. We show that this algorithm and natural generalizations of it fail to provide good probability-competitive performance.

4.1 Defining Randomized Partition Algorithms

The algorithm by Korula and Pal [19] was phrased in the language of graphs. Let us try to generalize it in a language applicable to all matroids. Before seeing any weights, their algorithm restricts itself (potentially randomly) to accepting only a subset of independent sets. More specifically, the algorithm will restrict its attention to the disjoint union[14] of solutions to simpler subproblems. The algorithm must ensure that for all feasible solutions to the subproblems, their union is a feasible solution to the main problem. In the case of the Korula-Pal algorithm, the smaller subproblems are instances of 1-uniform matroid secretary problems. (Several other algorithms for the Matroid Secretary Problem use similar high-level techniques, where the "simpler" matroids are not 1-uniform [9,15,20,24], and this idea is also used for the related prophet inequality [10].)

More concretely, we say that a partition is *valid* if the union of what is accepted by the instances of Dynkin's algorithm is an independent set (regardless of the weights and order of arrivals). Now we consider the following class of algorithms based on partition matroids:

Algorithm 2. Randomized Partition

1. Before looking at any weights, (perhaps randomly) validly partition the elements into parts S_i.
2. Within each part, run Dynkin's algorithm, and output the union of the selected elements.

One can ask whether any algorithm in this framework can be constant-*probability*-competitive. Theorem 4 shows that the answer is 'no'.

[14] This disjointness is why we refer to these generalizations as algorithms based on "partition matroids."

4.2 Randomized Partitions

In this section, we will rule out all algorithms based on partition matroids as candidates for achieving a constant probability-competitive ratio for the matroid secretary problem.

For the algorithm to always output a feasible solution, any partition it uses must be valid. Recall that a *valid* partition is one for which the union of what is accepted by the instances of Dynkin's algorithm is always independent. We say a distribution over partitions is *valid* if every partition in its support is valid.

Without loss of generality, we can assume the input graph is always complete. Otherwise, one can consider a modified weight-function that assigns a weight of zero to every edge that is not present. Since the algorithm cannot see the weights of the edges in advance, it will have to choose a partition of the complete graph at the start.

Theorem 4. *Any algorithm that draws a partition from a valid distribution $\mathcal{D}$ in Algorithm 2 is not α-probability-competitive for any $\alpha = \omega(n^{-1/8})$.*

The high-level plan in the proof of Theorem 4 is to plant a random *broom*, illustrated in Fig. 3, and show that with high probability, its handle is not accepted. We will refer to the lone neutral edge $\{u, w\}$ connecting the two stars as the *handle* of the broom. Note that the edges of non-zero weight in the broom form an acyclic subgraph and are therefore the unique max-weight basis of this graphic matroid.

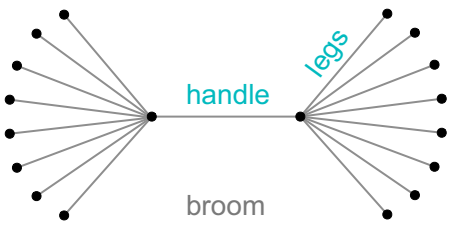

Fig. 3. Two stars connected by an edge form a broom. We call the bridge between the two stars the handle of the broom, and we the other edges of the broom as its legs.

Before proving this theorem, we characterize valid partitions.

Characterizing Valid Partitions. In this section we give a few characterizations of what valid partitions look like, which serve to provide intuition into why validity is a strong enough condition that prevents partition-based algorithms from probability-competitiveness.

We define a *valid* partition to be one where the union of what is accepted by the instances of Dynkin's algorithm is always an independent set, even for adversarial weights and arrival orders. We first give several equivalent

descriptions of what valid partitions should look like in the case of graphic matroids, which provides certain structural properties enforced by validity. It will be later used to prove our Theorem 4.

Lemma 7. *Let $\{S_1, \ldots, S_k\}$ partition the edges of a complete graph K_n, and let part(e) denote the S_i containing edge e. The following are equivalent:*

(a) ***Matroid condition:*** *$\{S_1, \ldots, S_k\}$ is valid.*
(b) ***Graph condition (i):*** *Every* cycle *has at least two edges in the same part.*
(c) ***Graph condition (ii):*** *Every* triangle *has at least two edges in the same part.*

Proof of Theorem 4. We provide a counterexample in the case of graphic matroids using the broom. Consider a partition $\mathcal{S} = \{S_1, \ldots, S_k\}$ of the edges of the complete graph. We say an edge $e \in S_i$ is "high-degree" if the sum of the degrees of its endpoints within the same part S_i is large. More concretely, we define the part-i degree of a vertex v as $\deg_i(v) = |\{e = \{a, b\} \in S_i : v \in \{a, b\}\}|$. Given an edge $e = \{a, b\}$ in part S_i, its degree is given by $\deg(e) = \deg_i(a) + \deg_i(b) - 1$, which intuitively means that we are counting all the incident edges in that part and the edge itself. An edge e is said to be *high-degree* if $\deg(e) \geq C$ for some C that we will choose later.

We will show that a $1 - o(1)$ fraction of the edges are high-degree for super-constant C. Therefore, an adversary can plant a random broom by assigning weights according to the following distribution: Pick a random edge $\{u, v\}$ in the graph, and randomly partition the vertices $V \backslash \{u, v\}$ into two parts X and Y of equal size (we assume $|V|$ is even). Assign a weight of 1 to every edge $\{u, x\}$ and $\{v, y\}$ for all $x \in X, y \in Y$, and a weight of zero to everything else. We will show that no matter what partition an algorithm chooses, the random edge $\{u, v\}$ will have a high-degree with high probability. The algorithm must therefore choose at most one edge from at least C elements of OPT. Hence, it cannot be better than $1/C$-probability-competitive.

It remains to show that a $1 - o(1)$ fraction of the edges are high-degree for some super-constant C in any valid partition $\mathcal{S}$ of the edges of the complete graph. A partition of the edges of K_n can be thought of as a coloring of its adjacency matrix $A \in M^{n \times n}$ (ignoring diagonal entries) in the obvious way (i.e., assign a different color to each S_i, and the color part(e) to the entry of A corresponding to e). In this notation, an entry of A is low-degree if there are fewer than C entries of the same color in its row or column. Note that by Lemma 7, a partition is valid iff every triangle has at least two edges in the same part. In the matrix language, a partition is valid iff for every three row indices u, v, w, at least two of $A(u, v)$, $A(u, w)$ and $A(v, w)$ are the same color. We will show using this interpretation of feasibility that each row and column must mostly consist of high-degree entries. More specifically, we will fix a vertex v, and consider any other two vertices u and w.

Proposition 1. *Let $C \leq (n-1)/2$ and let $T(n)$ be the maximum possible number of low-degree edges in any valid coloring of the complete graph on n vertices. For*

any vertex v, let x_i denote the number of edges adjacent to v in partition i. $T(n) \leq$

$$\max_{\boldsymbol{x} \in \mathbb{N}_{\geq 0}^{n-1}, \sum_i x_i = n,} \min \left\{ \sum_i T(x_i) + 2C(n-1),\ T(n-1) + 2(n-1-\max_i\{x_i\}) \right\}.$$

Proof. There are two steps: for any $\boldsymbol{x}$, we show that both the left term and the right term are always upper bounds (and therefore their minimum is a valid upper bound too).

Intuitively, the left term is better when $\max_i\{x_i\}$ is not too large. To see that the left term is always an upper bound, consider the following cases. Below, let X_i denote the set of nodes z such that (z, v) is in partition i (and therefore $x_i := |X_i|$).

- First, consider each X_i, and consider the induced subgraph on just these x_i nodes. The number of low-degree edges *just counting those between two nodes in* X_i is at most $T(x_i)$, by definition of $T(\cdot)$. Clearly, a node must be low-degree in the induced subgraph to possibly be low-degree in the full graph. This means there are at most $\sum_i T(x_i)$ low-degree edges between two nodes in the same X_i.
- Next, consider an edge between two nodes x, y both $\neq v$ which are *not* in the same X_i. This means that the edges (v, x) and (v, y) are *not* colored the same, and therefore the edge (x, y) *must* share a color with one of them for A to be valid. Whichever edge shares its color, we will charge its non-v endpoint (e.g. if (x, y) shares a color with (v, x), we charge x). Observe that once a vertex is charged C times, this means there are $C + 1$ edges adjacent to it which share the color of (v, x). This means that none of these edges are low-degree. Therefore, an edge can be low-degree only if its non-v endpoint is charged at most C times, and therefore there can be at most $C(n-1)$ such low-degree edges.
- Finally, consider an edge adjacent to v. We will lazily upper bound the number of low-degree edges by just the total number of edges, $n-1$, and further upper bound it by $C(n-1)$ for cleanliness of the expression.

This establishes the left term, which holds for any $\boldsymbol{x}$. Now we establish the right term. Intuitively, the right term is a better bound whenever $\max_i\{x_i\}$ is large. Let $j := \arg\max_i\{x_i\}$. If $x_i > C$, then there can be no low-degree edges adjacent to v in X_1. Therefore, there are at most $(n-1-x_j)$ low-degree edges adjacent to v. On the subgraph induced by the $n-1$ nodes other than v, there are clearly at most $T(n-1)$ low-degree edges by definition of $T(\cdot)$, and again any edge which is low-degree in the full graph must be low-degree in every induced subgraph. On the other hand, if $x_j \leq C$, then perhaps all edges adjacent to v are low-degree, and we can only use this technique to give an upper bound of $T(n-1) + n - 1$. In both cases, our bound is at most $T(n-1) + 2(n-1-\max_i\{x_i\})$ as long as $C \leq (n-1)/2$.

We will show inductively in Lemma 8 that $T(n) \leq b \cdot C \cdot n^{1+a}$, where a is a constant, and b and C are super-constant in n, as long as a few conditions hold. Corollary 1 lists values that satisfy these conditions, concluding that for all $0 < \varepsilon < 1/2$, there are valid assignments to the variables that achieve $T(n) \leq n^{3/2+\varepsilon}$.[15] Furthermore, Corollary 1 ensures that C is super-constant (and in particular polynomial in n), implying that with probability at least $\frac{\binom{n}{2}-n^{3/2+\varepsilon}}{\binom{n}{2}}$, the handle of the randomly planted broom will be high-degree for super-constant C. It can therefore only be selected with a sub-constant probability.

Lemma 8. *Consider the following recurrence when $C \leq (n-1)/2$. $T(n) \leq$*

$$\max_{\boldsymbol{x} \in \mathbb{N}_{\geq 0}^{n-1}, \sum_i x_i = n,} \min \left\{ \sum_i T(x_i) + 2C(n-1),\ T(n-1) + 2(n-1-\max_i\{x_i\}) \right\}.$$

with a base case of $T(n) = n(n-1)/2$ when $(n-1)/2 < C$. For all N, $T(N) \leq b \cdot C \cdot N^{1+a}$, as long as

1. *$a \in (0,1)$ is a constant;*
2. *C is a super-constant function of N;*
3. *b is a super-constant function of N such that $b(N) \geq 1$ for all N;*
4. *for all $n < N$, the following is satisfied: $\frac{2(n-1)}{abn^{1+a}} < \frac{(1+a)bC}{2n^{1-a}}$.*

As an immediate corollary, we get the following.

Corollary 1. *Let $T(n)$ be defined as in Lemma 8. Then for all $0 < \varepsilon < 1/2$, $T(n) \leq n^{3/2+\varepsilon}$.*

Now we can complete the proof of Theorem 4. Corollary 1 together with Proposition 1 establishes that for any $\varepsilon > 0$, there are at most $n^{3/2+\varepsilon}$ edges with degree at most $C := n^{\varepsilon/3}$. This means that with probability $1 - n^{-1/2+\varepsilon}$, a randomly selected edge (u, v) of the complete graph has degree at least $n^{\varepsilon/3}$. Conditioned on (u, v) having high-degree, we know that $n^{\varepsilon/3}$ edges of the max-weight spanning tree are in the same partition as (u, v). Therefore, at least one of them is selected with probability at most $n^{-\varepsilon/3}$. Setting $\varepsilon = 3/8$, we conclude that except with probability $n^{-1/8}$, there is some edge selected with probability at most $n^{-1/8}$, and therefore no randomized partition algorithm can be $\omega(n^{-1/8})$-probability-competitive.

References

1. Azar, P.D., Kleinberg, R., Weinberg, S.M.: Prophet inequalities with limited information. In: Chekuri, C. (ed.) Proceedings of SODA, pp. 1358–1377. SIAM (2014)
2. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: A knapsack secretary problem with applications. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX/RANDOM -2007. LNCS, vol. 4627, pp. 16–28. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74208-1_2

[15] It can be shown that this is in fact tight.

3. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: Online auctions and generalized secretary problems. SIGecom Exch. **7**(2), 1–11 (2008)
4. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: Matroid secretary problems. J. ACM **65**(6), 35:1–35:26 (2018)
5. Babaioff, M., Immorlica, N., Kleinberg, R.: Matroids, secretary problems, and online mechanisms. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 434–443 (2007)
6. Dinitz, M.: Recent advances on the matroid secretary problem. ACM SIGACT News **44**(2), 126–142 (2013)
7. Dynkin, E.B.: The optimum choice of the instant for stopping a Markov process. Sov. Math. **4**, 627–629 (1963)
8. Edmonds, J.: Matroids and the greedy algorithm. Math. Program. **1**(1), 127–136 (1971)
9. Feldman, M., Svensson, O., Zenklusen, R.: A simple O (log log (rank))-competitive algorithm for the matroid secretary problem. In: Proceedings of SODA, pp. 1189–1201 (2015)
10. Feldman, M., Svensson, O., Zenklusen, R.: Online contention resolution schemes. In: Proceedings of SODA, pp. 1014–1033 (2016)
11. Gale, D.: Optimal assignments in an ordered set: an application of matroid theory. J. Comb. Theory **4**(2), 176–180 (1968)
12. Gupta, A., Singla, S.: Random-order models. In: Roughgarden, T. (ed.) Beyond the Worst-Case Analysis of Algorithms. Cambridge University Press (2020)
13. Hajiaghayi, M.T., Kleinberg, R.D., Parkes, D.C.: Adaptive limited-supply online auctions. In: Proceedings 5th ACM Conference on Electronic Commerce (EC-2004), 17–20 May 2004, pp. 71–80. ACM, New York (2004)
14. Hoefer, M., Kodric, B.: Combinatorial secretary problems with ordinal information. In: Proceedings of ICALP, pp. 133:1–133:14 (2017)
15. Huynh, T., Nelson, P.: The matroid secretary problem for minor-closed classes and random matroids. arXiv preprint arXiv:1603.06822 (2016)
16. Jaillet, P., Soto, J.A., Zenklusen, R.: Advances on matroid secretary problems: free order model and laminar case. In: Goemans, M., Correa, J. (eds.) IPCO 2013. LNCS, vol. 7801, pp. 254–265. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36694-9_22
17. Kleinberg, R., Weinberg, S.M.: Matroid prophet inequalities. In: Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, pp. 123–136. ACM (2012)
18. Korte, B., Vygen, J.: Combinatorial Optimization, Volume 21 of Algorithms and Combinatorics. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-662-56039-6
19. Korula, N., Pál, M.: Algorithms for secretary problems on graphs and hypergraphs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 508–520. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02930-1_42
20. Lachish, O.: O (log log rank) competitive ratio for the matroid secretary problem. In: 2014 IEEE 55th Annual Symposium on Foundations of Computer Science, pp. 326–335. IEEE (2014)
21. Oxley, J.G.: Matroid Theory, vol. 3. Oxford University Press, New York (2006)
22. Rado, R.: Note on independence functions. Proc. London Math. Soc. **3**(1), 300–320 (1957)
23. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency, vol. 24. Springer, Heidelberg (2003)

24. Soto, J.A.: A simple PTAS for weighted matroid matching on strongly base orderable matroids. Electron. Notes Discrete Math. **37**, 75–80 (2011)
25. Soto, J.A., Turkieltaub, A., Verdugo, V.: Strong algorithms for the ordinal matroid secretary problem. In: Proceedings of SODA, pp. 715–734 (2018)