

Local Stability of Rankings

FELIX S. CAMPBELL, Ben-Gurion University of the Negev, Israel

YUVAL MOSKOVITCH, Ben-Gurion University of the Negev, Israel

Rankings play a crucial role in decision-making. However, if minor changes to items significantly alter their rankings, the quality of the decisions being made can be compromised. The stability of ranking is a measure used to assess how modifications to the ranking algorithm or data affect results. While previous work has focused on stability of the ranking under changes to the algorithm, we introduce a novel measure we refer to as *local stability*. Local stability indicates the effect of minor changes to the values of an item in the ranking on its rank. Our proposed definition furthermore takes into account the presence of multiple items with similar qualities in the ranking, called *dense regions*, permitting minor modifications to swap the positions of items within the region. We show that computing this measure in general is hard, and in turn propose a relaxation of the definition to admit approximation.

We present (i) LStability, a sampling-based algorithm for approximating local stability, on which we make probably-approximately-correct-type guarantees through the use of concentration inequalities, and (ii) Detect-Dense-Region, an algorithm based on this approach to detect the dense region an item lies in, if it exists. We introduce a number of optimizations to our algorithms to improve their scalability and efficiency. We validate our proposed framework through an extensive suite of experiments, including case studies highlighting the utility of our definitions.

CCS Concepts: • **Information systems** → **Data management systems**; • **Social and professional topics** → **Socio-technical systems**.

Additional Key Words and Phrases: ranking stability; ranking explanation; sensitivity analysis

ACM Reference Format:

Felix S. Campbell and Yuval Moskovitch. 2026. Local Stability of Rankings. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 204 (June 2026), 27 pages. <https://doi.org/10.1145/3802081>

1 Introduction

Ranking items or individuals by their quantitative attributes often plays a key role in a wide range of domain applications, e.g., in academia [6, 31], hiring [22], or e-commerce [17]. A major underlying assumption of ranking is that a higher ranking reflects a meaningful improvement in utility over lower ranked items. However, if minor modifications to the data result in significant shifts in an item's position in the ranking, this fundamental assumption is undermined. As a simple example, consider the results of a motorsport event: if the driver finishing 1st did so with a margin of only 0.2 seconds, the distinction from the runner-up may be practically negligible, whereas a 20-second lead would indicate a clearly superior performance. While the improvement in utility is clear for these simple cases, defining margins between items in more complex settings, such as learning-to-rank (LtR) [50] or rankings based on scores computed by multiple attributes, is not straightforward, as we next illustrate.

Authors' Contact Information: Felix S. Campbell, Ben-Gurion University of the Negev, Beersheba, Israel, felixsal@post.bgu.ac.il; Yuval Moskovitch, Ben-Gurion University of the Negev, Beersheba, Israel, yuvalmos@bgu.ac.il.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/6-ART204

<https://doi.org/10.1145/3802081>

Table 1. Top-10 computer science departments ranked according to adjusted average publication count. Each group of rows highlighted by a single color represents a dense region in the ranking in which the departments are similar.

	University	AI Pubs.	Systems Pubs.	Score ↓
t_1	Lakefront University	44	36	39.2
t_2	Dempster University	42	35	37.9
t_3	Western Polytechnic	43	33	36.7
t_4	Prairie University	23	25	25.4
t_5	Ogden University	22	24	24.4
t_6	Kedzie Institute	20	24	23.8
t_7	University of Blue Island	21	22	22.7
t_8	Plainfield College	7	13	11.9
t_9	Irving University	8	11	11.0
t_{10}	Kimball College	6	10	9.6

Example 1.1. When choosing a university to enroll in, students often consult a ranking of potential universities compiled by third parties (e.g., CSRankings [5]) in order to pick a program that can best help them achieve their goals. As a running example, we consider a ranking of fictitious universities based on a simplified version of CSRankings for the sake of exposition. Table 1 shows the top-10 universities in descending order of their adjusted average publication counts, computed by

$$\varphi(t) = \sqrt[17]{(t.\text{AI Pubs.} + 1)^5 \cdot (t.\text{Systems Pubs.} + 1)^{12}}$$

Intuitively, this score is the geometric mean of adjusted publication counts for the 5 and 12 subfields of AI and systems, respectively.

In this case, while Lakefront University is ranked 1st according to the ranking function, with just four fewer publications in systems (keeping the rest of the data unchanged), it would be 3rd. As the number of publications can slightly vary over time, this may raise questions such as “*how much does this university deserve its position in the ranking?*” or “*was this university a close second in the ranking?*” and questions on a broader level, such as “*how likely are the two universities to shift in the ranking?*”

Quantifying the well-foundedness of the positions of items in a ranking is referred to as ranking *stability*, and has been studied in previous works [2, 3, 12, 18, 23, 48]. In particular, in [3], ranking stability is used to measure the robustness of the ranking to changes in the ranking function used to produce it. This definition gives a *global* ranking stability score and provides insight into how stable the ranking is with respect to uncertainty in the ranking function.

Rankings frequently feature items with comparable qualities, such as universities of comparable standing, which can create *dense regions* within the ranking, where small changes can reasonably lead to position swaps among the items. This phenomenon was recognized, for example, in the National Research Council’s ranking of doctoral programs [31], which provides ranges of possible rankings for each university to account for uncertainty in the collected data.

Example 1.2. The universities in Table 1 are divided into three dense regions. Each dense region corresponds to a group of universities with similar scores generated by the given ranking function. As shown in Example 1.1, with small (hypothetical) modifications to the number of publications (4 in total), the 1st ranked university becomes 3rd. In contrast, the change in the number of publications that would shift Lakefront University to be ranked 4th is 17. The presence of dense regions in a ranking may impact decision-making. For example, consider a student admitted to the universities ranked 4th through 7th. Although the top choice should be the highest-ranked option, since all

of these universities belong to a dense region, they may be considered as roughly equivalent in quality. Consequently, other factors, such as location or program fit, might be prioritized over the precise ranking order.

The ranking stability measure presented in [3] has a coarse-grained nature, treating all changes in the ranking equally—for instance, a transposition of a single pair of items is as significant as a complete reversal of the ranking. Thus, this measure may overlook dense regions in the data. This motivates us to propose the concept of *local stability*, which considers stability as a property of individual items within the ranking rather than the ranking as a whole. Considering stability as a property of an individual item leads to a more natural treatment of dense regions, which are themselves properties of local areas within the ranking. Our contribution may be summarized as follows.

Local stability. We define a local stability measure aiming to quantify the effect of modifications to data on the outcome of the ranking for a given tuple by characterizing the magnitudes of changes (or *refinements*) to the tuple that can change its position significantly. To account for the effects of modifications to the data on the ranking we leverage the notion of counterfactual reasoning [30, 39, 44, 45]. Intuitively, we consider small hypothetical (counterfactual) modifications to a single tuple in the data to determine its margin relative to other tuples in the ranking. Restricting attention to a single tuple yields a local stability measure that reflects a best-case analysis, assuming no uncertainty in the remaining tuples of the database, which would otherwise exacerbate instability in many cases. Intuitively, lower magnitude modifications that can change the tuple’s position significantly correspond to a lower local stability. Our definition hinges on a user-given definition of what a *significant* change in positions is, and is specified by choosing a value k representing a range of positions around the tuple’s original ranking. This parameter k can then be used to account for dense regions when evaluating the local stability. Furthermore, our definition takes into account a *set of reasonable changes* which may be determined by domain-specific knowledge or in a data-driven manner, and bounds the magnitude of the refinements under consideration. Since local stability is focused on changes in the data, we are able to treat the ranking function as a black-box, allowing our definition to accommodate any ranking function, including those derived by complex LtR models.

In a nutshell, refinements may be partially ordered by the magnitude of modifications applied to each attribute value. For instance, continuing Example 1.1, changing the number of AI publications by 3 and systems publications by 2 is considered to be a larger modification compared to only altering the number of AI publications by 2. Utilizing this partial order, we can define a *stable zone*, where, intuitively, any perturbations to the values of a given tuple do not cause its position in the ranking to change by more than the given value of k . The local stability value is then defined as the relative portion of the stable zone, compared to the overall space of reasonable changes given by the user.

While useful and informative, computing the local stability of a tuple is intractable. To circumvent this challenge, we propose a relaxed definition, allowing the stable zone to possibly contain a small number of refinements that can modify the tuple’s position in the ranking by more than k (which we refer to as *k-unstable* refinements), however keeping the probability that such a refinement is sampled (uniformly) from the stable zone low.

Estimating local stability. We present a two-stage sampling-based algorithm that first samples from the space of reasonable changes to construct (an approximate) stable zone. It then verifies that the generated stable zone is approximately stable by sampling refinements within it. We show that by taking enough samples, verifying whether the computed stable zone is indeed approximately

Table 2. Summary of notations

Notation	Description
$\varepsilon(t)$	Tuple obtained by refining t by ε
$D_{t \rightarrow t'}$	Database D where tuple t is replaced by t'
$\Delta_{f(D)}(t, t')$	Change in positions for t when replaced by t' under $f(D_{t \rightarrow t'})$
$\leq$	Refinement containment relation
$E(\cdot)$	Set of non-containing refinements
RC	Set of reasonable changes
$k\text{-SB}_{f(D)}^t$	k -stable zone boundary
$\alpha\text{-}k\text{-SB}_{f(D)}^t$	α - k -stable zone boundary

stable (i.e., includes a low number of unstable refinements) can be done with high probability. We propose three optimizations to improve the performance of our sampling-based estimator.

Detecting dense regions. The local stability of a tuple evaluates how well it aligns with its assigned position in the ranking, and may be used to answer questions such as “*how much does this university deserve its position in the ranking?*” as demonstrated in Example 1.1. Alternatively, an important related problem is to determine the potential bounds within which a tuple’s position could vary in the ranking. Towards this end, we propose Detect-Dense-Region, whose goal is to identify, given a tuple, a value k that covers the extent of the dense region of the tuple. We do so by leveraging our local stability definition to understand where the margin between tuples (in counterfactual scenarios) is small enough to imply the existence and extent of a dense region.

Experimental evaluation. Finally, we validate our definition and proposed algorithms, LStability and Detect-Dense-Region, through an extensive experimental evaluation using both real and synthetic data, showing the usefulness of the approach and the efficiency of the algorithms. We present two case studies in which our definition leads to insights for rankings on real data. For example, we show that for the ranking of NBA players, the learned ranking function overfits Joel Embiid, implying his high ranking is ill-founded. Furthermore, we find that the CSRankings [5] ranking is fairly locally stable for most of the top-10 universities, supporting its reliability. Our experimental results also indicate that Detect-Dense-Region is able to accurately identify dense regions observed in the ranking. We conclude the experimental evaluation with a comparison of our local stability measure to the global stability measure as defined in [3], demonstrating how the two definitions can lead to divergent interpretations of the ranking.

Paper organization. The rest of the paper is organized as follows. In Section 2, we provide the background and formalizations necessary and then define exact and approximate local stability for rankings. In Section 3, we develop techniques to approximate local stability of an item in a ranking, and propose optimizations to our algorithm in Section 4. We present a heuristic to detect the dense region around an item in Section 5. In Section 6, we provide case studies and empirical analysis on the performance of our approach and a survey of related work is given in Section 7. Finally, we draw conclusions and discuss future directions in Section 8.

2 Problem Formulation

Local stability measures the effect of small modifications to a tuple on its position in the ranking while also taking into account dense regions, where multiple tuples have a similar quality and small changes can reasonably lead to position swaps among the tuples. We assume that data modifications

can be made independently, in the sense that changing one tuple does not require modifying any other tuple to preserve the semantics of the data. Under this assumption, we focus on modifications to individual tuples. We consider a *ranking* to be a permutation of a set of tuples [32]. A *ranking function* is a function f mapping a database D to a permutation of its tuples. We use $f(D)$ to denote the ranking resulting when applying f on a database D , and $f(D)[t]$ for the position of t in the ranking. When working with any database D , we consider only its numerical attributes. This assumption is standard and follows prior work (e.g., [3, 13, 32]).

2.1 Refinements

We aim to understand how changes in the values of different attributes of tuples affect their positions in the ranking. To do this, we first define *refinements* of a tuple, which describe possible modifications to the tuple and their impact on its values.

Definition 2.1 (Refinement). Given a database D with numeric attributes $a_1, \dots, a_n$, a *refinement* is a vector $\varepsilon = \langle \varepsilon_1, \dots, \varepsilon_n \rangle \in \mathbb{R}^n$. Given a refinement ε , the refined tuple $\varepsilon(t)$ is the tuple t' such that for all $i \in [n]$,

$$t'.a_i = t.a_i + \varepsilon_i$$

Example 2.2. Let us continue from Example 1.1. Considering a refinement $\varepsilon = \langle -10, 5 \rangle$, we have that $\varepsilon(t_1)$ is the tuple $(34, 41)$.

To understand how changes in the attributes of tuples affect their positions in the ranking, we define the notion of *position change* between two tuples.

Definition 2.3 (Position change). For a database D and a tuple t , let $D_{t \rightarrow t'}$ be the database obtained by replacing t with t'

$$D_{t \rightarrow t'} = D \setminus \{t\} \cup \{t'\}$$

The change of positions attained by substituting t' for t is

$$\Delta_{f(D)}(t, t') = |f(D)[t] - f(D_{t \rightarrow t'})[t']|$$

Example 2.4. Consider the refinement $\varepsilon = \langle -10, -5 \rangle$. The refined tuple $\varepsilon(t_1) = (34, 31)$ has a score of 32.9, which is less than the score of t_3 . Therefore, we have $\Delta_{f(D)}(t_1, \varepsilon(t_1)) = 2$ since $\varepsilon(t_1)$ ranks 3rd in $D_{t \rightarrow \varepsilon(t_1)}$.

Recall that our goal is to determine the effect of small modifications to a tuple on its position in the ranking, while also taking into account dense regions. To this end, we introduce a parameter k , which defines a range of positions around the tuple with which we would like to evaluate its local stability¹. This parameter can then be used to represent the size of the area around a tuple (i.e., tuples that are ranked at most k positions above or below) with approximately equal quality, i.e., the dense region. We assume k is given as an input, although we propose a heuristic in Section 5 for determining a value of k suitable for capturing the dense region around the tuple, if it exists. We are now ready to define a k -(un)stable refinement.

Definition 2.5 (k -(un)stable refinement). Given a database D , a ranking function f , a refinement ε , a value k , and a tuple $t \in D$, we say $\varepsilon(t)$ is k -stable for t over $f(D)$ if and only if $\Delta_{f(D)}(t, \varepsilon(t)) \leq k$. Otherwise, we say that $\varepsilon(t)$ is k -unstable.

¹In this paper, we discuss the symmetric case where the range extends in both directions of the ranking equally, for clarity of presentation. Extending our definitions and methods to make these ranges asymmetric is straightforward.

Example 2.6. Consider again the refinement $\varepsilon = \langle -10, -5 \rangle$, as in Example 2.4. Since $\Delta_{f(D)}(t_1, \varepsilon(t_1)) = 2$, we have that $\varepsilon(t_1)$ is 1-unstable for t_1 over $f(D)$.

2.2 Local Stability

In order to quantify the effect of small modifications to a tuple has on its position in the ranking, we use the magnitude of the changes required to do so. As smaller changes are more likely than larger changes, we are primarily interested in the *smallest* modifications that cause an item to move more than k positions in the ranking. For instance, in our running example, having ± 2 systems publications is more likely than having ± 10 systems publications. With this in mind, we first establish a partial order over refinements.

Definition 2.7 (Refinement containment). We say that a refinement ε is *contained* in a refinement ε' (and ε' *contains* ε), and denote it by $\varepsilon \leq \varepsilon'$, if for every $i \in [n]$, $|\varepsilon_i| \leq |\varepsilon'_i|$.

Refinement containment is defined with respect to absolute values as we are primarily concerned with the *magnitudes* of the refinements being made. For example, the refinement $\varepsilon' = \langle 10, 6 \rangle$ contains the refinement $\varepsilon = \langle 10, -5 \rangle$ since $|-5| \leq |6|$. Naturally, we write $\varepsilon < \varepsilon'$ when $\varepsilon \leq \varepsilon'$ and there is an i such that $|\varepsilon_i| < |\varepsilon'_i|$, i.e., it is the *strict* refinement containment relation. We can now define a k -stable zone boundary of a tuple t as follows.

Definition 2.8 (Stable zone boundary). Given a database D , a tuple $t \in D$, a ranking function f and a value k , let U be the set of refinements such that $\varepsilon(t)$ is k -unstable, and U^+ be the projection of U onto $\mathbb{R}_{\geq 0}^n$. Then, the k -stable zone boundary of t is the set

$$k\text{-SB}_{f(D)}^t = \text{Min}_{<}(U^+)$$

where

$$\text{Min}_{<}(R) = \{\varepsilon \in R \mid \nexists \varepsilon' \in R, \varepsilon' < \varepsilon\}$$

Intuitively, the stable zone boundary consists of the k -unstable refinements of *minimal* magnitude, i.e., there is no k -unstable refinement containing any refinement of the boundary (this is also known as a *skyline* [7]).

Example 2.9. Consider a dataset D with two attributes and a ranking function f over tuples in D . Figure 1a illustrates the space of refinements around a tuple $t \in D$ where each axis corresponds to the values of t in each one of the attributes. Refinements in the green region are k -stable, and the red area consists of k -unstable refinements (i.e., refinements in the set U). Figure 1b shows the projection of the refinements onto $\mathbb{R}_{\geq 0}^n$. In this illustration, all the refinements in the green area remain k -stable, whereas the lighter, cross-hatched red region corresponds to refinement magnitudes admitting both k -stable and k -unstable refinements in the projected space. Consequently, the stable zone boundary $k\text{-SB}_{f(D)}^t$ is the curved black line that separates the green zone from the red zone.

Definition 2.10 (Stable zone). For a set of refinements R , we denote by $E(R)$ the set of refinements that do not contain any refinement in R , i.e.,

$$E(R) = \{\varepsilon \in \mathbb{R}^n \mid \nexists \varepsilon' \in R, \varepsilon' \leq \varepsilon\}$$

We refer to $E(k\text{-SB}_{f(D)}^t)$ as the *stable zone*.

By Definition 2.8, any refinement in the stable zone is guaranteed to be k -stable. Continuing Example 2.9, the green region in Figure 1b is the stable zone $E(k\text{-SB}_{f(D)}^t)$.

The local stability of a tuple is defined with respect to a set of reasonable changes specified by the user, which bounds the values of possible refinements to small changes to the data for which

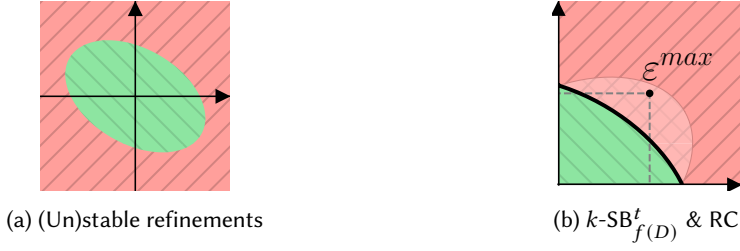


Fig. 1. Illustrated examples of definitions in Section 2

the user believes should not result in significant changes to the tuple's ranking (either leveraging domain-specific knowledge or in a data-driven manner as we do in Section 6.2). More formally, we define ϵ^{max} as a vector $\langle \epsilon_1^{max}, \dots, \epsilon_n^{max} \rangle \in \mathbb{R}_{\geq 0}^n$, and consider only refinements ϵ such that $\epsilon \leq \epsilon^{max}$.

Example 2.11. In the context of Example 1.1, the set of reasonable changes may depend on the amount of activity in the research area. For example, if we assume AI is a more active area than systems at these universities, we may assume that there is a larger variance in the number of AI papers than systems papers. Therefore, we may choose, e.g., $\epsilon^{max} = \langle 5, 3 \rangle$, encoding the assumption that each university might have ± 5 AI publications, and ± 3 systems publications.

We let RC denote the set of refinements contained within the vector ϵ^{max} (i.e., all $\epsilon \leq \epsilon^{max}$). For example, in Figure 1, RC is the set of refinements whose projection into $\mathbb{R}_{\geq 0}^n$ is in the area delineated by the dashed gray lines. We are finally ready to define the local stability of a tuple.

Definition 2.12 (Local stability). The *local stability* of a tuple t with respect to a database D , a ranking function f , a set of reasonable changes RC, and a parameter k is defined as

$$\text{Stability}_{f(D)}(t, k \mid \text{RC}) = \frac{\text{Vol}(\text{RC} \cap E(k\text{-SB}_{f(D)}^t))}{\text{Vol}(\text{RC})}$$

where $\text{Vol}(\cdot)$ is the volume of the sets (i.e., its Lebesgue measure).

Intuitively, local stability measures the relative size between the stable zone restricted to the set of reasonable changes and the set of reasonable changes.

Example 2.13. For the ϵ^{max} and RC shown in Figure 1b, the local stability is the ratio between the green area restricted to RC and the area beneath RC. In this case, the local stability would be high, since the area is mostly green.

Computing $k\text{-SB}_{f(D)}^t$ is a non-trivial task. In fact, we can show

THEOREM 2.14. *Unless $\text{FP} = \#P$, there is no polynomial-time algorithm for computing $k\text{-SB}_{f(D)}^t$ given a database D , a tuple $t \in D$, a ranking function f , and a value k .*

PROOF SKETCH. We prove this by showing that if $k\text{-SB}_{f(D)}^t$ can be computed in polynomial time, then computing $|k\text{-SB}_{f(D)}^t|$ is possible in polynomial time as well. However, we show that $\#D\text{NF}$, a well-known $\#P$ -complete problem [43], is polynomial-time reducible to computing $|k\text{-SB}_{f(D)}^t|$. Therefore, if there is such an algorithm to compute $k\text{-SB}_{f(D)}^t$, then $\text{FP} = \#P$. We relieve the details to [10]. $\square$

Moreover, we expect that even if the stable zone boundary $k\text{-SB}_{f(D)}^t$ is given, computing the stability measure remains intractable since it is closely related to the problem of computing the *hypervolume indicator*, which computes the volume of a union of hyperrectangles anchored at the origin and is known to be #P-hard [8, 24].

2.3 α -Local Stability

Following the complexity of local-stability computation, we define an approximate version of the problem. We start by defining an α - k -stable zone boundary as a set of refinements that is a k -stable boundary up to some error. Namely, the probability of sampling a k -unstable refinement from the set of refinements under an α - k -stable zone boundary is at most α when samples are drawn uniformly.

Definition 2.15 (α - k -stable zone boundary). Given a database D , a tuple $t \in D$, a ranking function f , a value k , and a set of reasonable changes RC , let U be the set of refinements such that $\varepsilon(t)$ is k -unstable, and U^+ be the projection of U onto $\mathbb{R}_{\geq 0}^n$. Then, an α - k -stable zone boundary of t is a set $\alpha\text{-}k\text{-SB}_{f(D)}^t \subseteq U^+$ such that

- (i) $\alpha\text{-}k\text{-SB}_{f(D)}^t = \text{Min}_{<}(\alpha\text{-}k\text{-SB}_{f(D)}^t)$
- (ii) $\mathbb{P}_{\varepsilon \sim \mathcal{U}(RC)}[\varepsilon(t) \text{ is } k\text{-unstable} \mid \varepsilon \in \alpha\text{-}k\text{-SB}_{f(D)}^t] \leq \alpha$

where $\mathcal{U}$ is the uniform distribution.

When α is small, we can assure that with high probability, refinements in $RC \cap E(\alpha\text{-}k\text{-SB}_{f(D)}^t)$ when made to the tuple will not cause its position in the ranking to shift by more than k . We can now define α -local stability.

Definition 2.16 (α -local stability). An α -local stability of a tuple t with respect to a database D , a ranking function f , an $\alpha\text{-}k\text{-SB}_{f(D)}^t$ Sb , a set of reasonable changes RC , and a parameter k is defined as

$$\alpha\text{-Stability}_{f(D)}(t, k \mid Sb, RC) = \frac{\text{Vol}(RC \cap E(Sb))}{\text{Vol}(RC)}$$

3 Computing Local Stability

We propose **LStability**, a sampling-based algorithm to estimate α -local stability given a database D , a tuple $t \in D$, a ranking function f , a set of reasonable changes RC , a value k , and parameters relating to the quality and confidence of the approximation which we discuss in the following. The algorithm relies on the fact that, despite the hardness of computing the stable zone boundary (Theorem 2.14), given a set of refinements R , we can efficiently bound α , such that R is an $\alpha\text{-}k\text{-SB}_{f(D)}^t$ with high probability. We first show that with a sufficient number of samples, we can estimate the probability that a given set R contains a k -unstable refinement.

PROPOSITION 3.1. *Given a database D , a ranking function f , a value k , a tuple $t \in D$, a confidence level δ , a confidence range η , and a set of refinements R , let*

$$p = \mathbb{P}_{\varepsilon \sim \mathcal{U}(RC \cap E(R))}[\varepsilon(t) \text{ is } k\text{-unstable}]$$

$$\hat{p} = \frac{1}{N} \sum_{\varepsilon \in S} \mathbb{1}[\varepsilon(t) \text{ is } k\text{-unstable}]$$

where S is a set of $N = \frac{1}{2\eta^2} \ln \frac{1}{\delta}$ samples drawn i.i.d. uniformly from $RC \cap E(R)$. Then we have that

$$\mathbb{P}[p - \hat{p} \leq \eta] \geq 1 - \delta$$

PROOF SKETCH. This results as an immediate application of the Hoeffding inequality [25]. $\square$

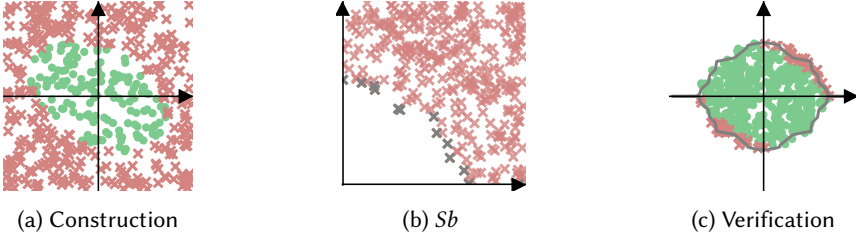


Fig. 2. Illustrated example of the steps of LStability, based on the space of refinements shown in Figure 1. (a) Green (circular) points represent sampled stable refinements, while red (cross) points represent unstable refinements. (b) The projection of the k -unstable refinements in C onto $\mathbb{R}_{\geq 0}^n$ (U_C^+). The gray (cross) points represent those along the estimated stable zone boundary. (c) Samples taken from the estimated stable zone $RC \cap E(Sb)$ for the verification step.

The following is a corollary of Proposition 3.1

COROLLARY 3.2. $N = \frac{1}{2\eta^2} \ln \frac{1}{\delta}$ samples drawn i.i.d. uniformly from $RC \cap E(R)$ are sufficient to confirm that R is an α - k - $SB_{f(D)}^t$ for $\alpha = \hat{p} + \eta$ with probability $1 - \delta$.

LStability builds on this idea. It gets as input a database D , a tuple $t \in D$, a ranking function f , a value k , a set of reasonable changes RC , a sample budget N , a confidence level δ , a confidence range η , and consists of two phases. The first generates a set of refinements Sb , which is an α - k - $SB_{f(D)}^t$ with high probability based on Corollary 3.2. In the second phase, LStability outputs an estimation of the ratio between the volume of the estimated stable zone defined by Sb and the volume of RC . Figure 2 provides an illustrative overview of LStability. In the first phase, LStability computes a set of refinements Sb , which is an α - k - $SB_{f(D)}^t$ with high probability. This is done in two sample-based steps: set *construction* and *verification*. In the construction step, illustrated in Figures 2a and 2b, the algorithm proceeds as follows:

- (1) Samples a set C of N refinements uniformly from RC and assigns to U_C^+ the projection of the k -unstable refinements in C onto $\mathbb{R}_{\geq 0}^n$ (see Figure 2a).
- (2) Computes $Sb = \text{Min}_<(U_C^+)$ (see Figure 2b).

In the verification step, illustrated in Figure 2c, the algorithm performs the following:

- (3) It samples a set V of a sufficient number of refinements ($N = \frac{1}{2\eta^2} \ln \frac{1}{\delta}$ samples) uniformly from $RC \cap E(Sb)$
- (4) It computes the proportion of k -unstable refinements in V to the overall number of samples N , thereby determining α such that Sb is α - k - $SB_{f(D)}^t$ with probability $1 - \delta$ based on Corollary 3.2

Finally, in the second phase, given Sb from the first phase, LStability applies standard Monte Carlo methods [38] to estimate α -Stability $_{f(D)}(t, k \mid Sb, RC)$. In cases where the stable zone is guaranteed to be convex (e.g., when the ranking function is based on scores computed by a linear combination of attributes), alternative methods may be used as a drop-in replacement. In particular, [16, 21] describe efficient algorithms for estimating the volumes of convex polytopes with theoretical guarantees. While these methods offer stronger theoretical guarantees for both accuracy and efficiency when applicable, the Monte Carlo approach is generally efficient and sufficiently accurate for our purposes. Moreover, we note that the convexity assumption does not hold in many practical settings.

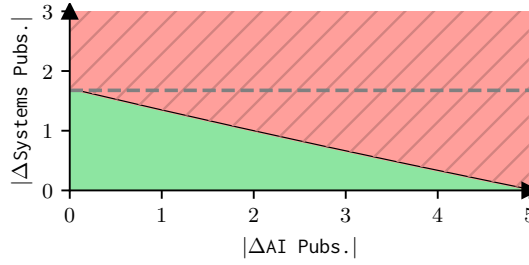


Fig. 3. The stability of refinements for Dempster Uni. and $k = 0$. Refinements in the green (solid) area are 0-stable and the red (hatched) area 0-unstable. The reduced set of reasonable changes containing all refinements on the stable zone boundary consists of those under the gray dashed line.

4 Optimizations

We next outline three optimizations to LStability. The first reduces the set of reasonable changes RC, allowing the algorithm to utilize the sample budget more effectively. The second optimization aims to reduce the cost of evaluating the ranking function f . This optimization is applicable when a modification of a tuple in the data does not affect the relative order of other tuples obtained by the ranking function, which is a common property of ranking functions. Finally, we suggest bounding α to reduce the running time. This is done by *iteratively* running the construction and verification steps with a partial sample budget in each iteration, allowing early termination if a desired level of α has been reached.

4.1 Reducing the Set of Reasonable Changes

The first optimization aims at improving the utility of the sampling budget by narrowing down the set of reasonable changes such that it is guaranteed to still contain all k -stable refinements under the stable zone boundary. Using a smaller set of reasonable changes reduces the sample space for the construction step of LStability, allowing for more informative sampling as the excluded refinements can not contribute to the computation of the stable zone boundary.

To reduce the set of reasonable changes, we use single-dimensional refinements, *i.e.*, refinements that modify a single attribute. Considering such refinements allows us to eliminate refinements that can not be within the stable zone boundary. More formally, consider an individual attribute a_i , and a refinement $\varepsilon \in \mathbb{R}^n$ such that $|\varepsilon_i| > 0$, and $\varepsilon_j = 0$ for every $j \neq i$. Furthermore, assume that $\varepsilon(t)$ is k -unstable for $f(D)$. As a consequence of the definition of $k\text{-SB}_{f(D)}^t$, we then have that any refinement ε' such that $|\varepsilon_i| \leq |\varepsilon'_i|$ cannot be in $E(k\text{-SB}_{f(D)}^t)$.

Example 4.1. Consider our running example and $\varepsilon^{\max} = \langle 5, 3 \rangle$ defined in Example 2.11. Figure 3 depicts the stability of refinements for Dempster Uni. and $k = 0$. Consider the refinement $\varepsilon = \langle 0, 2 \rangle$ as shown in Figure 3, $\varepsilon(t_2)$ is 0-unstable for $f(D)$. Thus, any refinement with more than ± 2 systems publications cannot be in $0\text{-SB}_{f(D)}^t$. In fact, any refinement that lies above the gray line cannot be part of the stable zone boundary.

To reduce RC, a portion of the sampling budget is used to sample single-dimensional refinements $\varepsilon_i^{\max}$ for every attribute a_i . Let ε_i^* denote the minimal (absolute) value of ε_i in the sample such that $\varepsilon(t)$ is k -unstable for $f(D)$. We then define RC^* as the set of reasonable changes where $\varepsilon_i^{\max} = \varepsilon_i^*$. LStability then uses RC^* instead of RC, and given an estimation of $\alpha\text{-Stability}_{f(D)}(t, k \mid Sb, \text{RC}^*)$,

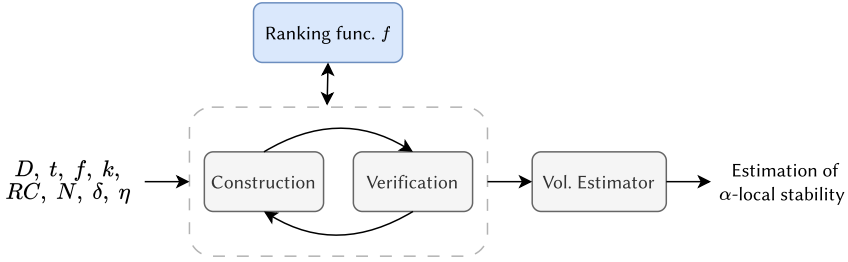


Fig. 4. Overview of the components of LStability

we scale it by a factor of $\frac{\text{Vol}(\text{RC}^*)}{\text{Vol}(\text{RC})}$ to obtain $\alpha\text{-Stability}_{f(D)}(t, k \mid Sb, \text{RC})$. When the ranking function is monotone [20], reducing RC can be done efficiently using binary search, which we detail in [10].

4.2 Reduce Re-ranking Cost

To determine whether a refinement ε is k -stable, LStability needs to compute the value $\Delta_{f(D)}(t, \varepsilon(t))$, which in turn depends on the value of $f(D_{t \rightarrow \varepsilon(t)})[\varepsilon(t)]$. The function f may be arbitrarily complex in its dependence on the size of the database D , e.g., in the case of learning-to-rank functions. Thus, evaluating $f(D_{t \rightarrow \varepsilon(t)})$ may be costly, in particular, when the sample budget N is large. However, we note that determining whether $\Delta_{f(D)}(t, t') \leq k$ can be done without the cost of computing $f(D_{t \rightarrow \varepsilon(t)})$ for tuple-independent ranking functions which we next define.

Definition 4.2 (Tuple-independent ranking function). A ranking function is **tuple-independent** if for any $t \in D$ and t' obtained by refining t , and all $a, b \in D \setminus \{t, t'\}$ such that $f(D)[a] < f(D)[b]$, then $f(D_{t \rightarrow t'})[a] < f(D_{t \rightarrow t'})[b]$.

Tuple independence is a common feature of ranking functions. For instance, the scoring function based on the geometric mean of attributes in Example 1.1 is tuple-independent, since altering the attributes of one tuple only modifies the score of that tuple, leaving the relative ordering between the rest of the tuples unchanged.

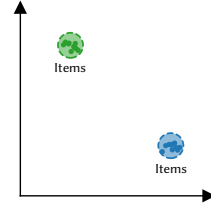
Given a tuple t and a refinement ε , let $t^\uparrow$ ($t^\downarrow$) be the tuple $k+1$ positions above (below) t in $f(D)$ where f is a tuple-independent ranking function. Since the relative ordering between all tuples except for the refined tuple $\varepsilon(t)$ remains the same, to determine whether $\Delta_{f(D)}(t, \varepsilon(t)) \leq k$, it is enough to check whether the relative order between $\varepsilon(t)$ and $t^\uparrow$ ($t^\downarrow$) is different than that of t and $t^\uparrow$ ($t^\downarrow$). Therefore, we only need to evaluate f over three tuples (a constant number), which can be significant when the database is large and the ranking function evaluation heavily depends on the data size (e.g., in Ltr models), as we show in our experimental evaluation (Section 6.5).

4.3 Reduce Sampling for Bounded α

LStability returns the level of minimality α attained by the refinements boundary it computes. However, it is natural to opt for a desired α , and it may be unnecessary for the construction step to sample as many times as it does in order to obtain a set of refinements Sb such that Sb is a α - k -SB $_{f(D)}^t$ with the desired α . To this end, we extend LStability with the ability to apply iteratively the *construction* and *verification* steps as depicted in Figure 4. In each iteration, the algorithm computes a set Sb using a limited amount of samples, and then checks whether it is α - k -SB $_{f(D)}^t$ for $\alpha \leq$ the given bound. If so, the algorithm continues to the volume estimation of the Sb and RC. Otherwise, the next construction step merges the unstable refinements from the previous verification step into Sb to improve its estimation using these counterexamples, and samples only

X	Y	Score ↓
9.71	0.86	10.57
1.38	9.15	10.53
9.30	1.20	10.51
...	...	...
8.31	0.52	8.83

(a) Scores of dataset



(b) Dataset visualized

Fig. 5. Example showing two distant clusters in the space of attributes can map to similar scores, suggesting they are all part of a single dense region in the ranking

from $RC \cap E(Sb)$ which ensures the sampled refinements can only improve our estimation of Sb . The process runs for ℓ iterations, as determined by the user.

The choice of the number of samples taken per iteration by the construction step introduces a trade-off. The goal is to take just enough samples to get a good enough estimated stable zone boundary, namely, a set which is α - k - $SB_{f(D)}^t$ with the desired α . However, when more than a single iteration is required to obtain a set Sb such that it is α - k - $SB_{f(D)}^t$ for the given α , the verification step is evaluated multiple times. We propose apportioning the budget equally to each iteration, taking into account the samples taken by the verification step in each iteration. Specifically, if N samples were allocated to the construction step and V samples to the verification step in the vanilla version of LStability, we allocate $\frac{N+V}{\ell} - V$ samples to the construction step in each iteration. Under this allocation, we take as many samples as was originally budgeted in the worst case.

5 Detecting Dense Regions

The local stability of a tuple t is defined with respect to a given k , defining the range of positions with respect to which local stability is evaluated. This parameterization allows our definition of local stability to be useful in the presence of *dense regions* in the ranking. Thus far, we have focused on computing the local stability of a given tuple in a ranking. In our running example, the local stability measure evaluates how well a particular university aligns with its assigned position in the ranking. An interesting alternative perspective is to explore, for a given university, the potential bounds on its possible position within the ranking, based on the local stability measure. Namely, identifying the dense region around a tuple in the ranking, if it is in one.

Intuitively, in order to identify dense regions, we need to cluster items in the ranking with similar utilities. However, this may be impossible when the ranking is not score-based (e.g., some list-wise LtR models [27]), so there is no meaningful measure of utility for a given item. Another naïve solution involves clustering based on the tuples' attributes. However, this approach assumes that all tuples ranked similarly must be similar in their attributes, which may not always be the case, as we next illustrate.

Example 5.1. Figure 5a shows items consisting of two attributes x and y ranked in descending order of $x + y$, each represented as a point in Figure 5b. As demonstrated in Figure 5b, two clusters may be observed based on their attribute values and are depicted as blue and green dashed circles in the figure. However, their scores, as shown in Figure 5a, are all similar, indicative of all items belonging to a single dense region.

To this end, we propose **Detect-Dense-Region**, a heuristic for computing a value k for a given tuple in the database, following the intuitive meaning of dense regions viewed through the lens

Algorithm 1: Detect-Dense-Region

Input: Database D , a tuple $t \in D$, a ranking function f , a set of reasonable changes RC and a sample budget N

Output: A value k (the size of the dense region of t)

```

1  $S \leftarrow \emptyset$ 
2 for  $i \leftarrow 1$  to  $N$  do
3    $\varepsilon \leftarrow \text{Sample}(\mathcal{U}(RC))$ 
4    $t' \leftarrow \varepsilon(t)$ 
5    $S \leftarrow S \cup \{(\varepsilon, \Delta_{f(D)}(t, t'))\}$ 
6  $k^* \leftarrow \max(\{p \mid (\varepsilon, p) \in S\})$  ► Largest change of positions induced by samples from RC
7  $\widehat{\text{Stability}}(t, k^*) \leftarrow 1$ 
8 for  $k \leftarrow 0$  to  $k^* - 1$  do
9    $S \leftarrow S \setminus \{(\varepsilon, p) \in S \mid p \leq k \wedge (\nexists (\varepsilon', p') \in S, (p' > k) \wedge (\varepsilon' \leq \varepsilon))\}$ 
10   $\widehat{\text{Stability}}(t, k) \leftarrow 1 - \frac{|S|}{N}$ 
11  $d_0 \leftarrow \widehat{\text{Stability}}(t, 0)$ 
12 for  $k \leftarrow 1$  to  $k^*$  do
13    $d_k \leftarrow \widehat{\text{Stability}}(t, k) - \widehat{\text{Stability}}(t, k - 1)$ 
14  $C_s, C_l \leftarrow \text{Cluster}(d_0, \dots, d_k)$  ► Cluster returns a partitioning of  $d_0, \dots, d_k$  into two groups: small and large
15 return  $\min_{k \in [k^*], d_k \in C_l} k$ 

```

of our definition of local stability. The high-level idea of Detect-Dense-Region is to compare the differences in the stability values of a given tuple for different k values and identify the last k value before a large difference occurs. The pseudocode of Detect-Dense-Region is shown in Algorithm 1. The algorithm first estimates the local stability for different values of k (lines 1-10). Then (lines 11-15), the algorithm uses clustering to partition the differences in the computed stability values into small and large changes, and returns as suggested k the position where the first large difference was observed. We next provide details on each phase.

Estimating local stability for multiple values of k . To determine the first value of k for which we observe a large difference in estimated local stability, we first need to estimate the local stability for different values of k . The challenge here is to do so efficiently. Evaluating LStability for all possible values of k is unnecessarily expensive when we only need rough estimates of the local stability.

In order to do so, Detect-Dense-Region samples refinements according to a given sample budget N and stores each sampled refinement ε together with the change in position $\Delta_{f(D)}(t, \varepsilon(t))$ in a set of samples S (lines 1-5). After sampling, the algorithm determines the maximum change in positions induced by any of the samples and denoted by k^* (line 6). These samples are used to estimate the local stability for every k between 0 and k^* , denoted by $\widehat{\text{Stability}}(t, k)$. k^* is assumed to be the maximum change in positions induced by *any* refinement contained in the set of reasonable changes RC (which is true for a sufficiently large number of samples), thus the estimated stability for k^* is 1 (line 7). For every value of k from 0 to $k^* - 1$ the stability is estimated using the set of samples S . Intuitively, the stability value for a given k can be estimated as the number of samples (ε, p) in S such that $\varepsilon(t)$ is k -stable (namely, $p \leq k$), and there is no $(\varepsilon', p') \in S$ such that $\varepsilon' \leq \varepsilon$ and $\varepsilon'(t)$ is k -unstable (namely, $p' > k$), divided by the total number of samples N . Note that the estimated $(k + 1)$ -stable zone boundary is above the estimated k -stable zone boundary. Thus, the estimation can be done iteratively (lines 8-10) as follows: for each k , we remove from S the

k -stable refinements not containing any sampled k -unstable refinement (line 9) and estimate the local stability for k as $1 - \frac{|S|}{N}$ (line 10).

Clustering-based suggestion of dense region. Given an estimation of the local stability for the observed values of k , according to our notion of a dense region, the extent of the dense region in which the tuple lies is the first value of k for which the succeeding value of k has a significantly larger value of local stability. Therefore, Detect-Dense-Region computes the difference in stability (lines 11 - 13) for each value of k , d_k , as the local stability for k minus the local stability for $k - 1$ (and fixing $d_0 = \text{Stability}(t, 0)$). It then employs a Fisher-Jenks natural breaks [19] clustering to partition the differences into two groups, C_s and C_l of small and large differences, respectively (lines 14). Finally, Detect-Dense-Region returns k , where k is the smallest value such that d_k is in the set of large differences C_l (line 15).

6 Experiments

In this section, we present an experimental evaluation. We start with two case studies whose goal is to demonstrate the usefulness of our local stability definition. We then study the effect of the sample budget N , the confidence level δ , and the confidence range η on the performance of LStability. We examine the performance of Detect-Dense-Region using real and synthetic data, and show the scalability of the algorithm and the usefulness of our proposed optimizations. We conclude with a comparison to the definition of stability that was presented in [3].

6.1 Experimental Setup

Implementation details. Our experiments were evaluated on Google Cloud Compute Engine c4d-highcpu-4 servers, with 4 vCPUs (AMD EPYC 9005 series) and 7 GB of memory. We implement the algorithms as an extension of [9] in Python 3.13, using Polars² 1.29 to load and manipulate data. We repeat each experiment 10 times, and show the averages of the quantities of interest (e.g., runtime, estimated (α) -local stability) in the figures.

We implement the uniform sampling over the refinements under the estimated boundary Sb , used in the verification and optimized construction step of LStability, by rejection sampling. More concretely, we sample uniformly from RC, and reject refinements containing any refinement of Sb . The efficiency of rejection sampling can be poor when the volume of the area we want to sample from is relatively small compared to RC. Therefore, we use a threshold over the estimated volume τ_v , such that when the estimated ratio between the volume of the area and its containing hyperrectangle is below τ_v at the end of the construction step, we skip the verification step and stop early. We use $\tau_v = 0.05$, as improving the estimation below this point yields diminishing returns with respect to the verification cost. We evaluated 10,560 experiments in total, where the verification step was skipped due to τ_v in 7.6% of them.

We evaluate LStability with all optimizations described in Section 4 enabled and compare its performance to the basic version of LStability without optimizations as presented in Section 3.

Parameters setting. For the construction step, we set the total number of samples to be at most 750,455, where in each iteration 20,000 samples are taken when the maximal number of iterations ℓ is 20. We chose this number empirically from the sample budget parameter study in Section 6.3. For the verification step, unless otherwise specified, we set the bound on α to be 0.05 and the confidence bound $\delta = 0.05$. That is, we are satisfied if we are 95% sure that 95% of refinements contained in the estimated boundary are k -stable, for the given value of k . In order to have a decent chance of attaining $\alpha \leq 0.05$, we set the confidence range $\eta = 0.01$.

²<https://pola.rs>

Datasets & ranking functions. We evaluated our algorithms using both real and synthetic datasets, utilizing different types of ranking functions as follows.

NBA (2023-2024): We consider a ranking of the top-100 players according to a popular sports website [37]. Since the ranking methodology is not provided, we learn a ranking function from the reference ranking using player statistics from the 2023-2024 season (collected from [34]) as the input features. We use LightGBM [28] to learn a regressor based on the same five statistics used in [13]: points (PTS), total rebounds (TRB), assists (AST), steals (STL), and blocks (BLK). For these five statistics, we consider the season totals since we are interested in ranking overall season performance. By default, we let the set of reasonable changes (RC) be approximately 5% of the maximum difference in each attribute.

CSRankings: We use a dataset and ranking function from [32] (based on [5]), comprising 188 universities and the counts of their faculty and publications for four different fields of computer science: artificial intelligence (AI), systems (Sys), theory (Thry), and interdisciplinary research (Intdsc). The count of publications is adjusted to account for the number of authors in each publication by dividing the credit equally, *e.g.*, a paper with 2 authors from different institutions counts as 0.5 publications for each of the involved institutions (*i.e.*, fractional changes in the publication count for a given institution are valid and meaningful). The universities are ranked in descending order of their scores, which are computed as a variation of geometric mean over the adjusted publication counts in each area—specifically,

$$\varphi(t) = \sqrt[27]{(t.AI + 1)^5 \cdot (t.Sys + 1)^{12} \cdot (t.Thry + 1)^3 \cdot (t.Intdsc + 1)^7}$$

We let RC be approximately 5% of the maximum difference in each attribute—about ± 4 adjusted publication count in AI and ± 1 adjusted publication count in the other fields.

Synthetic: We generate synthetic datasets with dense regions specifically in mind. In order to generate a synthetic dataset of N tuples and d attributes, we iteratively generate dense regions with a random number of tuples until N tuples are in the dataset. Each dense region is assigned a score s , which is separated from the preceding dense region by a constant margin c . We then draw tuples from a multivariate Gaussian distribution whose mean is the one-vector scaled by $\frac{s}{d}$, and the covariance matrix is a scaled identity matrix. We rank the resulting dataset by the ranking function, sorting the tuples in descending order of the sum of their attributes. We let RC be defined by the vector ϵ^{max} whose components are proportional to $\frac{c}{d}$, ensuring that local stability values are comparable across varying attribute counts for a fixed margin c .

6.2 Case Studies

We start with two case studies on the NBA and CSRankings datasets.

6.2.1 Case Study: NBA Player Rankings. Rankings are frequently featured in sports, and can drive decision-making in a number of contexts, *e.g.*, in fantasy leagues, drafts, and estimating the strength of a given team. In this case study, we investigate the local stability of NBA players in the 2023-2024 season as ranked by a learned ranking function, with the goal of ranking players by their performance that season. Since the top of the ranking receives significantly more attention [17, 42], we focus on the local stability of the top-10 ranked players. The local stability of the top-10 players for different values of k is depicted in Figure 6a.

Assume we wish to choose the season's most valuable player (MVP). According to the ranking, a natural choice would be the top-ranked player of this season, Nikola Jokić. However, Figure 6a shows that his ranking in 1st is very unstable with local stability of 0.02 for $k = 0$. Indeed, with minor modifications to his total statistics that season, *e.g.*, +10.04 in PTS, −3.04 in TRB, −0.56 in AST, −1.01 in STL, and −1.11 in BLK, he would be ranked 2nd. Notice that these changes are particularly

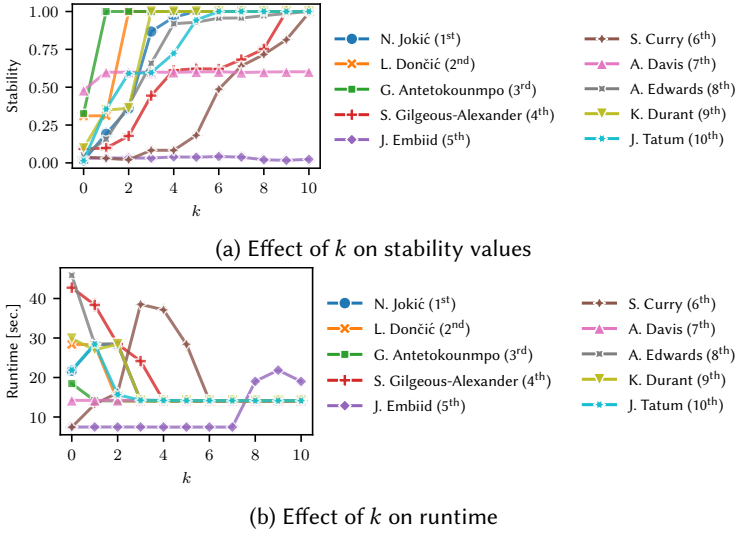


Fig. 6. Case study results for top-10 NBA players

small in comparison to his overall season totals: in this season, Jokić had 2,085 PTS, 976 TRB, 708 AST, 108 STL, and 68 BLK. Furthermore, his ranking among the top-3 is also unstable, with local stability of 0.36 for $k = 2$. This suggests that naming Jokić as this season's MVP may not be well-founded under this ranking function. Based on this ranking function, we may instead conclude that choosing Luka Dončić is more justifiable given that he is certainly in the top-4 (more likely than Jokić). Alternatively, we may refine the ranking function (*e.g.*, by considering only a subset of the attributes) to obtain a ranking with higher local stability values that can be used to justify the chosen MVP.

We observed that most of the top-10 players are not stable within their own position: the highest estimated stability when $k = 0$ is 0.46, in the case of Anthony Davis. While most players are unstable for $k = 0$, we observed that the majority of players are stable within ± 3 ranks of their initial ranking. Therefore, we may infer this particular ranking function is reasonably locally stable for the players in the top-10, and confirm that it is overall *well-founded*: namely, highly-ranked players remain highly-ranked even under small modifications in their statistics for this season. A notable exception is Joel Embiid, whose stability is very low for all evaluated values of k . This suggests that the learned ranking function has overfit to Embiid's statistics, as even very small modifications to his statistics cause him to drop out of the top-10 players. This interpretation is supported by the fact that Embiid played far fewer games than other players in the 2023-2024 season (only 39, compared to more than 70 for most other players), due to injuries. Therefore, his total statistics for the season are much lower than they would have been otherwise, making them more similar to lower-ranked players.

Figure 6b shows the runtime of (the optimized) LStability for different values of k for each of the top-10 players. Broadly speaking, the runtime is decreasing for increasing values of k . Intuitively, this is explained by the fact that there are necessarily more k -unstable refinements for lower values of k , often resulting in more iterations taking place in LStability to produce a good enough estimate. Notable exceptions are Stephen Curry and Joel Embiid, in which the runtime starts low (because they are below the estimation volume threshold τ_v), rises when just above τ_v , and then decreases again once their local stability increases. When compared to the basic version of LStability, as

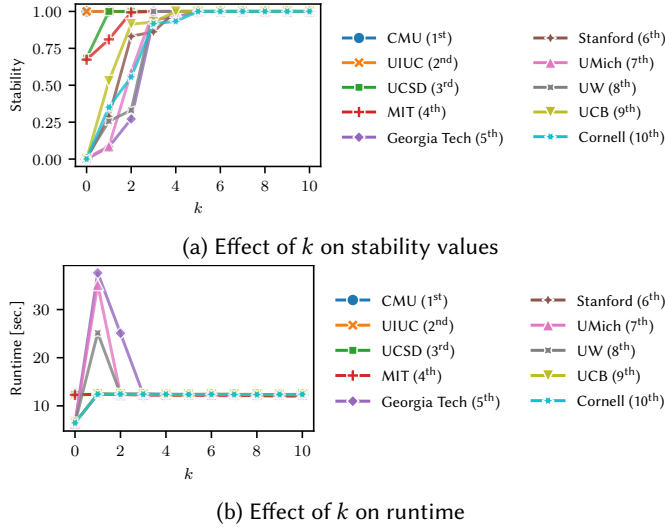


Fig. 7. Case study results for top-10 universities

presented in Section 3, our optimized algorithm is $51.6\times$ faster in the best case, and $25.4\times$ faster on average. We show the full results in [10].

6.2.2 Case Study: CSRankings. Rankings also feature prominently in higher education, and have been the subject of significant study in order to develop methodologies that accurately reflect the perceptions of the quality of the institutions when compared with each other [6, 31]. We consider evaluating the local stability of institutions in a more specialized ranking, CSRankings [5], which ranks institutions based on publication metrics in computer science conferences. Given the focus placed on high positions in the ranking, we again consider the local stability of the top-10 universities.

Figure 7a shows that the ranking is substantially locally stable. In fact, the top 2 universities, Carnegie Mellon University (CMU) and the University of Illinois at Urbana-Champaign (UIUC), are estimated to be completely locally stable: no change made to them from the set of reasonable changes causes them to rank outside of 1st or 2nd, respectively. For the remaining universities, we see that for $k = 3$, all of the universities have an estimated local stability of more than $\frac{1}{2}$. Furthermore, from $k \geq 5$, all of the universities are estimated to be completely locally stable. This lends credence to the claim of these universities fielding the top-ranked computer science departments, since small changes in the ranking do not result in drastic changes in their position.

Figure 7b shows the runtime of (the optimized) LStability depending on the value of k . As expected, the runtime in general decreases with k . As in the NBA case study, there are universities with estimated stabilities just above the threshold τ_v , leading to higher runtimes for those values of k . We observed a speedup of up to $35.2\times$ in the best case, and of $19.1\times$ on average compared to the performance of the basic version of LStability. We show the full results in [10].

6.3 Effect of Parameters

The next set of experiments study the effect of the sampling budget N and verification parameters δ (confidence level) and the η (confidence range) on the running time of LStability and the computed stability values. The observed effects of the parameters on the stability values were negligible, and

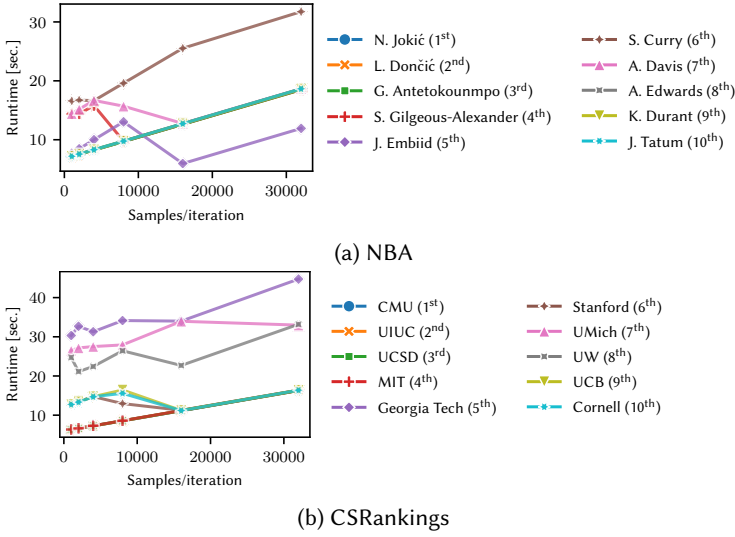


Fig. 8. Effect of the number of samples per round taken by the construction step on runtime

thus are left to [10]. For each dataset, we set k such that a wide variety of values of local stability are observed across different tuples. Specifically, for NBA we chose $k = 5$, and $k = 1$ for CSRankings.

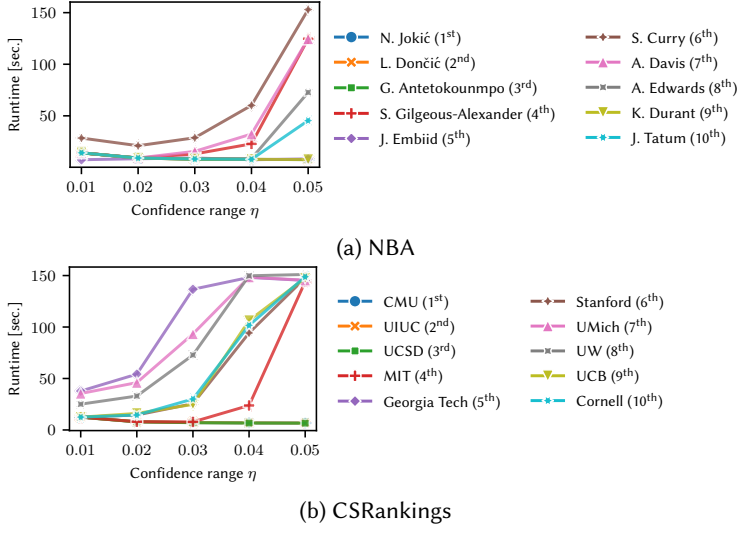
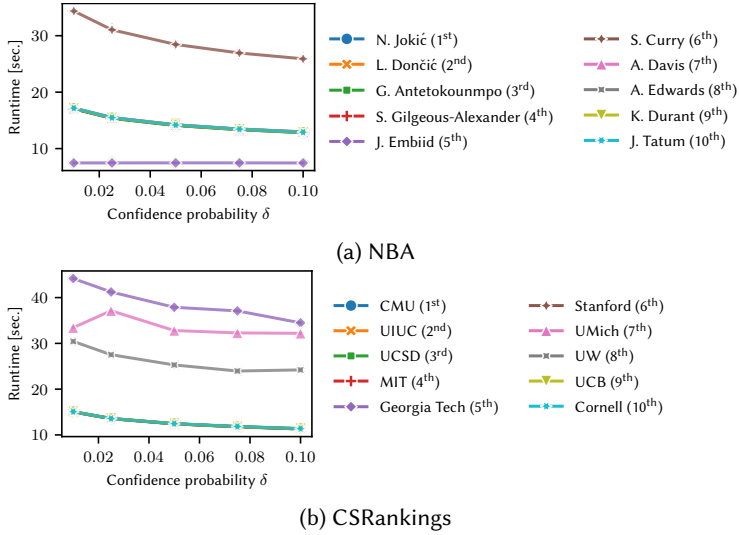
Sample budget. Figure 8 depicts the running time for increasing sampling budget N between 1,000 and 32,000. We observed a tradeoff in running time between small and large per-iteration sample budgets. When the number of samples in a single iteration is insufficient to achieve the desired α , multiple iterations are required, incurring the cost of multiple executions of the verification step. This phenomenon is more pronounced for tuples with low values of local stability (e.g., Cornell in Figure 8b).

Verification parameters (η, δ). To assess the effect of η , we vary its value from 0.01 to 0.05. The results are shown in Figure 9. We observed an increase in running times. This can be explained as follows. Recall that $\alpha = \hat{p} + \eta$ where $\hat{p}$ is the estimated probability that the computed Sb contains an unstable refinement as defined in Proposition 3.1. Thus, when α is fixed, as η increases, the acceptable upper bound on $\hat{p}$ decreases, meaning the portion of sampled unstable refinements under Sb should decrease. Achieving a lower $\hat{p}$ may require an increase in the number of samples, and as a result growing number of iterations, which incurs higher computation time. Finally, Figure 10 shows the running time for values of δ from 0.01 to 0.1. As expected, the runtime decreases with increasing δ values, i.e., for low confidence level, the running time is lower.

6.4 Dense Region Detection

We next evaluate Detect-Dense-Region in terms of accuracy and runtime. To this end, we use the CSRankings dataset and the Synthetic dataset to assess the quality of the output given the ground truth over dense regions in the data. We do not consider the NBA dataset in this experiment, as the learned ranking function treats items with similar statistics in a substantially disparate manner (e.g., the case of Joel Embiid in the NBA dataset), leading to the absence of clear dense regions in the rankings.

Table 3 presents the top-10 universities in CSRankings along with their computed score. We observe a clear separation between different groups in the data based on their scores, which is

Fig. 9. Effect of the confidence range η on runtimeFig. 10. Effect of the confidence probability δ on runtime

highlighted in the table. Specifically, there is a notable gap in the scores for tuples ranked 1st-4th, whereas tuples ranked 5th-8th and tuples ranked 9th-10th have a more comparable score to each other, forming dense regions in the data. *E.g.*, with only one less adjusted publication count in systems, Georgia Tech would be ranked 8th.

The evaluation of Detect-Dense-Region was able to identify the dense regions. In particular, the k value computed for each one of the first four tuples was 0, indicating their stability with respect to their surrounding tuples. For Stanford, the k value computed by Detect-Dense-Region was 1 while one might reasonably expect $k = 2$ so as to include the University of Washington

Table 3. Top-10 computer science departments according to CSRanking [32], with dense regions highlighted

	University	AI	Sys	Thry	Intdsc	Score ↓
1 st	CMU	71.4	11.9	21.1	13.8	19.53
2 nd	UIUC	46.1	12.6	16.0	7.2	15.39
3 rd	UCSD	31.6	9.0	10.1	10.3	13.00
4 th	MIT	28.1	8.6	16.2	7.9	12.33
5 th	Georgia Tech	28.5	7.8	6.9	10.2	11.58
6 th	Stanford	36.7	5.4	13.3	11.5	11.56
7 th	UMich	30.4	9.0	9.3	5.9	11.26
8 th	UW	28.0	6.2	12.2	10.0	11.13
9 th	UCB	23.2	7.4	15.9	6.4	10.69
10 th	Cornell	42.0	5.7	12.8	6.8	10.66

(UW). Indeed, the local stability value for $k = 0$ is $7 \cdot 10^{-6}$, for $k = 1$ is 0.29 and for $k = 2$ is 0.83, i.e., an increase of 0.29 from $k = 0$ to $k = 1$, while the increase from $k = 1$ to $k = 2$ is 0.54, which is significantly larger. However, the former is classified as being large by the clustering algorithm due to the smaller differences between other values of k (which are all less than 0.15). The observed k values for the rest of the tuples matched the dense regions as they are depicted in Table 3. These results highlight the usability of Detect-Dense-Region to detect dense regions in the rankings as a reasonable approach to solving the problem. When accounting for dense regions, the local stability of the ranking becomes evident. Specifically, when choosing k to span the dense area of the universities, we find that all the universities (except Cornell) have a local stability of at least $\frac{1}{2}$. In particular, Cornell has lower stability within its dense region than UC Berkeley (UCB) since UCB is buoyed by their relative strength in systems publications while Cornell is not, therefore making it feasible for Cornell to rank 11th.

To further evaluate the performance of Detect-Dense-Region, we utilized the Synthetic dataset with $d = 2$ and 100 tuples, which was generated with dense regions and thus provides us with ground truth for k values that cover the dense region to which the tuple belongs. We executed Detect-Dense-Region on all 100 tuples of the Synthetic dataset and compared the output k value for each tuple with the ground truth value of k . For all 100 tuples, Detect-Dense-Region was correctly able to recommend the value of k that fits the dense region according to the data generation process.

Efficiency of Detect-Dense-Region. To show the efficiency of Detect-Dense-Region, we compared the runtime of Detect-Dense-Region to that of a variant where the stability values are computed using LStability. The evaluation was done on all 100 tuples of the Synthetic dataset. The total runtime for Detect-Dense-Region was 7 minutes, compared to 141 minutes when using LStability for the local stability values computation. We observed a 20.3× improvement overall, where Detect-Dense-Region was up to 32× faster with an average of 20.4× improvement.

6.5 Scalability & Optimizations

We next examine the scalability of the optimized version of LStability (LSt) compared to the basic version presented in Section 3 (Basic), and quantify the effect of our proposed optimizations.

Scalability. We evaluate the scalability of our methods with growing data sizes and numbers of attributes using synthetic datasets. Figure 11 shows the runtime of both the optimized and basic versions of LStability as a function of the data size. As expected, LSt outperforms Basic. We observed a linear increase in the Basic’s running time, with increasing data size, while the impact on

LSt's running time was negligible. This confirms that the dependence on data size can be eliminated for tuple-independent ranking functions by the optimization to reduce the re-ranking cost.

The runtime as a function of the number of attributes is depicted in Figure 12. We note that the running time can be affected by the stability of the tuples, *i.e.*, tuples with stability below the threshold τ_v typically incur shorter runtimes due to early termination. Due to the randomness involved in the data generation process and tuple selection, it is likely that each evaluated dataset has a mixture of high and low stabilities. Consequently, we report the maximum observed running time per tuple. For Basic, the runtime increases linearly with the number of attributes as each additional attribute incurs more overhead in *e.g.*, computing $\text{Min}_<$ of a set of refinements. Furthermore, the runtime of LSt increases exponentially with the number of attributes, as additional iterations must be performed in order to attain the desired α . Interestingly, for more than 6 attributes, the performance of LSt degrades and becomes slower than Basic in the worst cases. In these cases, LSt is performing close to the maximum number of allotted iterations, and incurs overhead from *e.g.*, the increasing cost to perform the rejection sampling as the estimated stable zone shrinks. However, we emphasize that this is only true in the worst case (Figure 12 shows *max* runtimes). On average, LSt is 11.8 $\times$ faster for 6 to 10 attributes.

Ablation Study. We compare the runtime of different variants of LStability: (LSt) LStability equipped with all the optimizations, (LSt-I), denoting LStability where the iteration optimization for bounded α is disabled, (LSt-C), denoting LStability without the optimization for reducing the set of reasonable changes, (LSt-R), denoting LStability with no reduction of the re-ranking cost, and (Basic) denoting LStability with no optimizations enabled, as described in Section 3. To demonstrate the difference in performance, we report the overall running time for computing the stability values for every tuple in the top-10 for the values of k chosen in Section 6.3 for each variant of the algorithm. The results are shown in Figure 13.

For the NBA dataset, the basic version was overall 28.6 $\times$ slower than LSt and 13.2 $\times$ slower for CSRankings. Interestingly, the performance gap between Basic and LSt-I is negligible on CSRankings, whereas on the NBA dataset Basic is 1.51 $\times$ slower. This is because Basic and LSt-I draw the same number of samples, but LSt-I additionally incorporates an optimization that reduces re-ranking costs, which is more effective for the NBA dataset than CSRankings due to the higher cost of the learned ranking function used for the former. The use of iterative computation to achieve a bounded α value was shown to be the most effective, improving the runtime for a single tuple by a factor of 37 $\times$ in the best case, and 18.9 $\times$ overall for the NBA dataset, and 13.2 $\times$ for CSRankings.

A notably smaller effect was observed for the reduction of reasonable changes optimization: no change in the NBA dataset, and 1.23 $\times$ faster for CSRankings. The higher effect on CSRankings is because small refinements to a *single* attribute can render a tuple locally unstable in this dataset, whereas in the NBA dataset, relatively complex interactions between attributes are learned by the LtR model, and so typically refinements to multiple attributes are necessary to move the tuple in the ranking. Additionally, we note that the ranking function in CSRankings is monotone, allowing for more efficient computation of the reduced RC.

Finally, we see a higher gain for the optimization of reducing the cost of re-ranking in the NBA dataset, 1.51 $\times$ in total runtime compared to no change in CSRankings. This is due to the fact that the ranking function's runtime depends on the data size. For CSRankings, the optimization makes almost no difference, since the runtime of evaluating a simple scoring function for a small dataset is negligible when compared to the other phases of LStability.

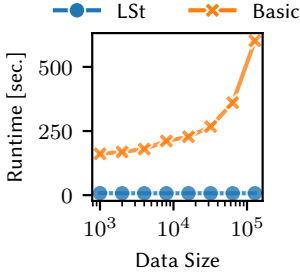


Fig. 11. Effect of data size on runtime

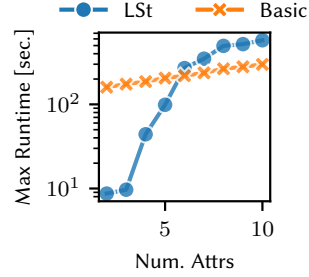
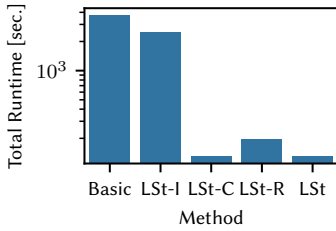
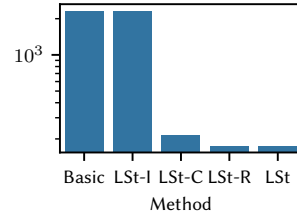


Fig. 12. Effect of num. attrs. on runtime



(a) NBA



(b) CSRRankings

Fig. 13. Total runtimes for experiment suites under ablation of optimizations

6.6 Local vs. Global Ranking Stability

We next demonstrate the difference between our proposed definition for local stability and the (global) stability of ranking as was defined in [3]. We implement the Monte Carlo algorithm described in the paper for 2 dimensions, and sample 500,000 function weights to determine the global stability of the Synthetic ranking subject to changes of a, b in the scoring function used to rank $\varphi(t) = a \cdot t.x + b \cdot t.y$ where $a, b \geq 0$. The original ranking, obtained by setting $a = 1, b = 1$, has a global stability of 0.03 (out of 1) while according to our definition, when taking into account the marked dense regions, most tuples are considerably locally stable (on average 0.45) given the margins between dense regions and the set of reasonable changes described in Section 6.1.

In contrast, consider a variant of the Synthetic dataset with 10 tuples, where each of the attributes of the tuple ranked i^{th} is greater by 1 than the tuple ranked at the $i + 1$ position in each attribute. Clearly, this ranking is considered perfectly stable, as each tuple dominates all the others ranked below it. However, according to our local stability measure, the stability of each tuple is at most 0.08 for values of k that don't cover the entire ranking.

7 Related Work

Ranking stability. The stability of ranking has been studied in a line of work [2, 3, 12, 18, 23, 48]. Closest to our work is [3], which studies stability as the property of a *ranking* with respect to a family of ranking functions, which we deemed *global* stability. At a high level, stability measures how much the output may change in response to small changes in the input. Any concrete definition of stability, therefore, depends on how modifications are defined and on the notion of output effect. The specific choices for these components determine the semantics of the stability measure. Technically, [3] defines modifications as changes to the *ranking function*, and considers *any change* in the ranking order to constitute an output effect. In contrast, we study modifications to the *data*

and quantify the output effect as the *magnitude of the change* in the position (*i.e.*, the value k) of a given tuple. Conceptually, the stability notions in [3] are geared toward assessing how likely the (entire) ranking of the tuples is, and can be used to determine, *e.g.*, whether the ranking function was cherry-picked. In contrast, our definitions aim to measure how close tuples are in the ranking under the *original ranking function*, which is especially important in scenarios where no scores are available, or the scores are non-linear in the input attributes. Note that by focusing on data changes rather than methodology, we can treat the ranking process as a black box, making our framework model-agnostic. Moreover, our definitions overlook changes within the dense region, thereby making our measure more useful when similar tuples are present in the ranking.

A form of local stability was studied in [12]; however, the analysis in [12] relies on strong assumptions about both the ranking function and the dataset, which together enable a closed-form characterization of the perturbation magnitude required to alter the ranking. In particular, beyond restricting the class of ranking functions considered, their results apply only to a *perfect season* dataset - one in which every team plays every other team exactly once, with no upsets: the top-ranked team defeats all others, the second-ranked team loses only to the first, and so forth. These assumptions render the analysis tractable but substantially limit its generality. Extending the framework of [12] to more general datasets or to alternative ranking functions is non-trivial and would require significant additional work. In contrast, our approach is model-agnostic and imposes no assumptions on the underlying data. Moreover, [12] does not explicitly provide a method to measure the local stability of a tuple when multiple attributes contribute to the score. Finally, we note that [12] considers perturbations affecting exactly two tuples, while we examine perturbations made to a single tuple.

Ranking explanations. Beyond ranking stability, the problem of explaining ranking outcomes has been studied extensively [1, 13–15, 20, 32]. Many of these works explain rankings by identifying feature importance [1, 13, 14, 20, 32]. While related to stability, such approaches do not explicitly quantify the margin between tuples in the ranking, as we propose. In [13, 14], linear scoring functions are fit to a dataset to either match the top- k of a given ranking or put a single tuple among the top- k . The weights of the resulting function may be interpreted as an explanation of feature importance. ShaRP [32] adopts Shapley values to explain *why* a given item ranks the way it does according to the contributions made by each of the individual attributes. Similarly, [1] employs existing feature importance methods (*e.g.*, LIME [35]) to explain rankings. The work of [20] exploits monotonicity assumptions to derive importance measures of feature importance. Orthogonally, [15] finds sufficient subsets of attributes (called an abductive explanation) which, when agreed upon, result in the same ordering between tuples.

Robustness verification in ML. Verifying the robustness of machine learning models has been the subject of much recent attention [4, 11, 26, 41, 47, 51, 52]. Essentially, robustness quantifies how likely a perturbation to an input will change its prediction [11]. Recent work has focused on *probabilistic* definitions of robustness, similar to our definition of α -local stability; we refer readers to [52] for a recent survey. Many works (*e.g.*, [4, 26, 41, 47, 51]) employ a framework similar to ours, leveraging sampling and concentration inequalities to estimate robustness with guarantees on their reliability and accuracy [41, 47, 51]. However, a key difference separates our definition from this line of work: the space of perturbations under consideration. Most often, robustness is considered as a function of an ℓ_p -ball [4]. This presupposes a cost function on the perturbations enabling the determination of a bound on the maximum cost of changes for which the model is robust. However, our notion of stable zone boundaries replaces this assumed cost function with a weaker assumption: that the cost of a change is monotone non-decreasing. This enables including parts of the stable area which are not included in any ℓ_p -ball (*e.g.*, the arms of an “L”-shaped stable zone). Furthermore, in these works, the inference tasks are independent of other points (*i.e.*, classification and regression),

while this is not the case in the ranking setting—local stability is not only a property of the ranking function, but depends on the other tuples in the ranking as well (e.g., in the case of dense regions), motivating optimizations specific to this setting (e.g., Section 4.2).

Counterfactual explanations. Counterfactual explanations provide concrete examples of how to achieve *recourse* for an undesirable classification [30, 33, 39, 40, 44, 46, 49]. Most of the time, these methods focus on providing a few actionable options for recourse based on minimizing the cost of changing the input [39, 46]. Instead, we focus on trying to characterize a broader set of options for recourse, essentially finding minimal recourse without specifying a cost function up front. Furthermore, many methods such as [30, 44] are tailored to specific models assuming white-box access, while our sampling-based approach allows us to be model-agnostic.

Local model-agnostic explanations. Local model-agnostic explanations [29, 32, 35, 36] answer *why* a certain decision was made for the given input for any given model by treating it as a black box. In many cases, they take the form of *feature importance* measures, as in [29, 32, 35]. While feature importance correlates with the local stability of a single attribute, we principally consider changes to combinations of attributes instead.

8 Conclusion

We recognized the need for a *local view* of stability in rankings, due to the existence of dense regions in which small amounts of instability are reasonably expected and therefore tolerated. We defined this notion formally as a ratio between the refinements whose magnitude cannot incur a large change in positions to a user-defined set of reasonable changes chosen with the help of domain-specific knowledge. We showed that computing this set of changes exactly is intractable in the general case, leading us to propose a relaxed definition of local stability. We proposed LStability, a sampling-based estimation algorithm and showed a probably-approximately-correct type guarantee by applying a concentration inequality. We further proposed Detect-Dense-Region to suggest a range of positions around a given item in the ranking with which we may reasonably expect small changes to move the item due to being ranked next to similar items. Finally, we demonstrated that our definition of local stability can provide interesting insights through case studies on real-life datasets, demonstrated the suitability of Detect-Dense-Region for its intended purpose, and provided an experimental analysis of the parameters affecting the performance of our algorithms and optimizations.

Our work introduces many avenues for further study into the local stability of rankings. Future work may consider additional classes of refinements, such as adding or removing data, or requiring refinements to satisfy constraints (e.g., denial constraints or tuples with correlated values), and handling categorical data. Extending our framework to categorical attributes may be possible in certain cases, but doing so is non-trivial, and modeling refinement magnitude appropriately depends on whether the categorical domain admits a natural order (e.g., small, medium, and large sizes).

Acknowledgments

This research was supported by the Israel Science Foundation grant 2121/22, and the Frankel Center for Computer Science at BGU.

References

- [1] Hadis Anahideh and Nasrin Mohabbati-Kalejahi. 2022. Local Explanations of Global Rankings: Insights for Competitive Rankings. *IEEE Access* 10 (2022), 30676–30693. doi:10.1109/ACCESS.2022.3159245
- [2] Abolfazl Asudeh and H. V. Jagadish. 2020. Fairly Evaluating and Scoring Items in a Data Set. *Proc. VLDB Endow.* 13, 12 (2020), 3445–3448. doi:10.14778/3415478.3415566

- [3] Abolfazl Asudeh, H. V. Jagadish, Gerome Miklau, and Julia Stoyanovich. 2018. On Obtaining Stable Rankings. *Proc. VLDB Endow.* 12, 3 (2018), 237–250. doi:10.14778/3291264.3291269
- [4] Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel, and Prateek Saxena. 2021. Scalable Quantitative Verification For Deep Neural Networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 312–323. doi:10.1109/ICSE43902.2021.00039
- [5] Emery D. Berger. 2025. CSRankings. <https://csrankings.org/>.
- [6] Emery D. Berger, Stephen M. Blackburn, Carla E. Brodley, H. V. Jagadish, Kathryn S. McKinley, Mario A. Nascimento, Minjeong Shin, Kuansan Wang, and Lexing Xie. 2019. GOTO rankings considered helpful. *Commun. ACM* 62, 7 (2019), 29–30. doi:10.1145/3332803
- [7] Stephan Börzönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, Dimitrios Georgakopoulos and Alexander Buchmann (Eds.). IEEE Computer Society, 421–430. doi:10.1109/ICDE.2001.914855
- [8] Karl Bringmann and Tobias Friedrich. 2010. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Comput. Geom.* 43, 6-7 (2010), 601–610. doi:10.1016/J.COMGEO.2010.03.004
- [9] Felix S. Campbell and Yuval Moskovitch. 2025. Locator: Local Stability for Rankings. In *Companion of the 2025 International Conference on Management of Data, SIGMOD/PODS 2025, Berlin, Germany, June 22-27, 2025*, Volker Markl, Joseph M. Hellerstein, and Azza Abouzied (Eds.). ACM, 55–58. doi:10.1145/3722212.3725113
- [10] Felix S. Campbell and Yuval Moskovitch. 2026. Local Stability of Rankings. arXiv:2603.09724 [cs.DB] <https://arxiv.org/abs/2603.09724>
- [11] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 39–57. doi:10.1109/SP.2017.49
- [12] Timothy P. Chartier, Erich Kreutzer, Amy N. Langville, and Kathryn E. Pedings. 2011. Sensitivity and Stability of Ranking Vectors. *SIAM Journal on Scientific Computing* 33, 3 (2011), 1077–1102. doi:10.1137/090772745
- [13] Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. 2023. Why Not Yet: Fixing a Top-k Ranking that Is Not Fair to Individuals. *Proc. VLDB Endow.* 16, 9 (2023), 2377–2390. doi:10.14778/3598581.3598606
- [14] Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. 2025. Synthesizing Scoring Functions for Rankings Using Symbolic Gradient Descent. In *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 3849–3862. doi:10.1109/ICDE65448.2025.00287
- [15] Francesco Chiariello and João Marques-Silva. 2025. Formal Explanations of Black-Box Ranking Functions. In *Logics in Artificial Intelligence - 19th European Conference, JELIA 2025, Kutaisi, Georgia, September 1-4, 2025, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 16093)*, Giovanni Casini, Besik Dundua, and Temur Kutsia (Eds.). Springer, 28–44. doi:10.1007/978-3-032-04587-4_3
- [16] Ben Cousins and Santosh S. Vempala. 2016. A practical volume algorithm. *Math. Program. Comput.* 8, 2 (2016), 133–160. doi:10.1007/S12532-015-0097-Z
- [17] Mahsa Derakhshan, Negin Golrezaei, Vahideh Manshadi, and Vahab Mirrokni. 2022. Product Ranking on Online Platforms. *Management Science* 68, 6 (2022).
- [18] Siddhartha Devic, Aleksandra Korolova, David Kempe, and Vatsal Sharan. 2024. Stability and Multigroup Fairness in Ranking with Uncertain Predictions. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. <https://openreview.net/forum?id=YiblhkVL2w>
- [19] Walter D. Fisher. 1958. On Grouping for Maximum Homogeneity. *J. Amer. Statist. Assoc.* 53, 284 (1958), 789–798. arXiv:<https://www.tandfonline.com/doi/pdf/10.1080/01621459.1958.10501479> doi:10.1080/01621459.1958.10501479
- [20] Abraham Gale and Amélie Marian. 2020. Explaining Monotonic Ranking Functions. *Proc. VLDB Endow.* 14, 4 (2020), 640–652. doi:10.14778/3436905.3436922
- [21] Cunjing Ge and Feifei Ma. 2015. A Fast and Practical Method to Estimate Volumes of Convex Polytopes. In *Frontiers in Algorithmics - 9th International Workshop, FAW 2015, Guilin, China, July 3-5, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9130)*, Jianxin Wang and Chee-Keng Yap (Eds.). Springer, 52–65. doi:10.1007/978-3-319-19647-3_6
- [22] Sahin Cem Geyik, Stuart Ambler, and Krishnaram Kenthapadi. 2019. Fairness-Aware Ranking in Search & Recommendation Systems with Application to LinkedIn Talent Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, Ankur Teredesai, Vipin Kumar, Ying Li, Römer Rosales, Evimaria Terzi, and George Karypis (Eds.). ACM, 2221–2231. doi:10.1145/3292500.3330691
- [23] Yifan Guan, Abolfazl Asudeh, Pranav Mayuram, H. V. Jagadish, Julia Stoyanovich, Gerome Miklau, and Gautam Das. 2019. MithraRanking: A System for Responsible Ranking Design. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1913–1916. doi:10.1145/3299869.3320244

- [24] Andreia P. Guerreiro, Carlos M. Fonseca, and Luís Paquete. 2021. The Hypervolume Indicator: Computational Problems and Algorithms. *ACM Comput. Surv.* 54, 6, Article 119 (July 2021), 42 pages. doi:10.1145/3453474
- [25] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. Springer.
- [26] Wei Huang, Xingyu Zhao, Gaojie Jin, and Xiaowei Huang. 2023. SAFARI: Versatile and Efficient Evaluations for Robustness of Interpretability. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 1988–1998. doi:10.1109/ICCV51070.2023.00190
- [27] Jiarui Jin, Xianyu Chen, Weinan Zhang, Mengyue Yang, Yang Wang, Yali Du, Yong Yu, and Jun Wang. 2023. Replace Scoring with Arrangement: A Contextual Set-to-Arrangement Framework for Learning-to-Rank. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, Ingo Frommholz, Frank Hopfgartner, Mark Lee, Michael Oakes, Mounia Lalmas, Min Zhang, and Rodrygo L. T. Santos (Eds.). ACM, 1004–1013. doi:10.1145/3583780.3615031
- [28] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 3146–3154. <https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html>
- [29] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 4765–4774. <https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>
- [30] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. 607–617.
- [31] Jeremiah P Ostriker, Charlotte V Kuh, and James A Voytuk. 2011. *A data-based assessment of research-doctorate programs in the United States*. National Academies Press.
- [32] Venetia Plattsika, Joao Fonseca, Kateryna Akhynko, Ivan Shevchenko, and Julia Stoyanovich. 2024. ShaRP: A Novel Feature Importance Framework for Ranking. arXiv:2401.16744
- [33] Rafael Poyiadzi, Kacper Sokol, Raúl Santos-Rodríguez, Tijl De Bie, and Peter A. Flach. 2020. FACE: Feasible and Actionable Counterfactual Explanations. In *AIES '20: AAAI/ACM Conference on AI, Ethics, and Society, New York, NY, USA, February 7-8, 2020*, Annette N. Markham, Julia Powles, Toby Walsh, and Anne L. Washington (Eds.). ACM, 344–350. doi:10.1145/3375627.3375850
- [34] Sports Reference. 2024. *2023-24 NBA Player Stats: Totals*. Retrieved July 29, 2024 from http://basketball-reference.com/leagues/NBA_2024_totals.html
- [35] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 1135–1144. doi:10.1145/2939672.2939778
- [36] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). AAAI Press, 1527–1535. doi:10.1609/AAAI.V32I1.11491
- [37] The Ringer. 2024. *The NBA, Ranked*. Retrieved July 29, 2024 from <https://nbarankings.theringer.com>
- [38] Christian P. Robert and George Casella. 2004. *Monte Carlo Statistical Methods*. Springer. doi:10.1007/978-1-4757-4145-2
- [39] Maximilian Schleich, Zixuan Geng, Yihong Zhang, and Dan Suciu. 2021. GeCo: Quality Counterfactual Explanations in Real Time. *Proc. VLDB Endow.* 14, 9 (2021), 1681–1693. doi:10.14778/3461535.3461555
- [40] Shubham Sharma, Jette Henderson, and Joydeep Ghosh. 2020. CERTIFAI: A Common Framework to Provide Explanations and Analyse the Fairness and Robustness of Black-box Models. In *AIES '20: AAAI/ACM Conference on AI, Ethics, and Society, New York, NY, USA, February 7-8, 2020*, Annette N. Markham, Julia Powles, Toby Walsh, and Anne L. Washington (Eds.). ACM, 166–172. doi:10.1145/3375627.3375812
- [41] Karim Tit, Teddy Furon, and Mathias Rousset. 2021. Efficient Statistical Assessment of Neural Network Corruption Robustness. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 9253–9263. <https://proceedings.neurips.cc/paper/2021/hash/4d215ab7508a3e089af43fb605dd27d1-Abstract.html>

- [42] Raluca M. Ursu. 2018. The Power of Rankings: Quantifying the Effect of Rankings on Online Consumer Search and Purchase Decisions. *Marketing Science* 37, 4 (2018), pp. 530–552. <https://www.jstor.org/stable/48748468>
- [43] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421. doi:10.1137/0208032
- [44] Peter M. VanNostrand, Huayi Zhang, Dennis M. Hofmann, and Elke A. Rundensteiner. 2023. FACET: Robust Counterfactual Explanation Analytics. *Proc. ACM Manag. Data* 1, 4 (2023), 242:1–242:27. doi:10.1145/3626729
- [45] Sahil Verma, Varich Boonsanong, Minh Hoang, Keegan Hines, John Dickerson, and Chirag Shah. 2024. Counterfactual Explanations and Algorithmic Recourses for Machine Learning: A Review. *ACM Comput. Surv.* 56, 12, Article 312 (Oct. 2024), 42 pages. doi:10.1145/3677119
- [46] Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. 2017. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *CoRR* abs/1711.00399 (2017). arXiv:1711.00399 <http://arxiv.org/abs/1711.00399>
- [47] Lily Weng, Pin-Yu Chen, Lam M. Nguyen, Mark S. Squillante, Akhilan Boopathy, Ivan V. Oseledets, and Luca Daniel. 2019. PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6727–6736. <http://proceedings.mlr.press/v97/weng19a.html>
- [48] Ke Yang, Julia Stoyanovich, Abolfazl Asudeh, Bill Howe, H. V. Jagadish, and Gerome Miklau. 2018. A Nutritional Label for Rankings. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 1773–1776. doi:10.1145/3183713.3193568
- [49] Wenzhuo Yang, Jia Li, Caiming Xiong, and Steven C. H. Hoi. 2022. MACE: An Efficient Model-Agnostic Framework for Counterfactual Explanation. *CoRR* abs/2205.15540 (2022). arXiv:2205.15540 doi:10.48550/ARXIV.2205.15540
- [50] Meike Zehlike, Ke Yang, and Julia Stoyanovich. 2023. Fairness in Ranking, Part I: Score-Based Ranking. *ACM Comput. Surv.* 55, 6 (2023), 118:1–118:36. doi:10.1145/3533379
- [51] Tianle Zhang, Wenjie Ruan, and Jonathan E. Fieldsend. 2022. PRoA: A Probabilistic Robustness Assessment Against Functional Perturbations. Springer-Verlag, Berlin, Heidelberg, 154–170. doi:10.1007/978-3-031-26409-2_10
- [52] Xingyu Zhao. 2025. Probabilistic Robustness in Deep Learning: A Concise yet Comprehensive Guide. arXiv:2502.14833 [cs.LG] <https://arxiv.org/abs/2502.14833>

Received October 2025; revised January 2026; accepted February 2026