CRAWD: Sampling-Based Estimation of Count-Distinct SPARQL Queries

Thi Hoang Thi Pham(✉), Pascal Molli, Brice Nédelec, Hala Skaf-Molli, and Julien Aimonier-Davat

Nantes Université, CNRS, LS2N, Nantes, France
{thi-hoang-thi.pham,pascal.molli,brice.nedelec,hala.skaf}@univ-nantes.fr

Abstract. Count-distinct SPARQL queries compute the number of unique values in the results of a query executed on a Knowledge Graph. However, counting the exact number of distinct values is often computationally demanding and time-consuming. As a result, these queries often fail on public SPARQL endpoints due to fair use policies. In this paper, we propose CRAWD, a new sampling-based approach designed to approximate count-distinct SPARQL queries. CRAWD significantly improves sampling efficiency and allows feasible execution of count-distinct SPARQL queries on public SPARQL endpoints, considerably improving existing methods.

Keywords: Semantic Web · Approximate Aggregation · Sampling · Count-Distinct

1 Introduction

How many people live in Europe? How many books have been written by Europeans? How many women head cities in Europe? How many distinct objects are in a knowledge graph? These questions can be formulated using count-distinct aggregate SPARQL queries, as shown in Fig. 1. However, these queries pose significant execution challenges as they require substantial memory to prevent double-counting and are time-consuming because they typically involve scanning large segments of knowledge graphs. All queries in Fig. 1 time out after 60 s on Wikidata, providing no results. This is due to the fair use policy of public SPARQL endpoints, which maintains servers responsive but prevents many count-distinct aggregate SPARQL queries from completing. Consequently, many downstream activities cannot be performed online, such as processing aggregate queries [7], developing indexes of SPARQL endpoints [9,15], and computing summaries for federation engines [18].

If it is not possible to return exact results in 60 s, it may be possible to return an approximate value in 60 s, which can be refined by running the count-distinct aggregate query for another round of 60 s. Approximate query processing has already been used for count SPARQL queries [1,18], but it does not support

G. Demartini et al. (Eds.): ISWC 2024, LNCS 15232, pp. 98–115, 2025.
https://doi.org/10.1007/978-3-031-77850-6_6

```
SELECT(COUNT(DISTINCT ?person)
       AS ?cd_persons )
WHERE {
  ?person wdt:P31 wd:Q5 .
  ?person wdt:P19 ?city .
  ?city wdt:P17 ?country .
  ?country wdt:P30 wd:Q46 .
}
```

(a) How many people live in Europe?

```
SELECT(COUNT(DISTINCT ?book )
       AS ?cd_books )
WHERE {
  ?book wdt:P31 wd:Q571 .
  ?book wdt:P50 ?author .
  ?author wdt:P19 ?city .
  ?city wdt:P17 ?country .
  ?country wdt:P30 wd:Q46 }
```

(b) How many books were written by Europeans?

```
SELECT(COUNT(DISTINCT ?mayor )
       AS ?cd_mayors )
WHERE {
  ?city wdt:P17 ?country .
  ?city wdt:P6 ?mayor .
  ?mayor wdt:P21 wd:Q6581072 .
  ?country wdt:P30 wd:Q46 .
}
```

(c) How many women head cities in Europe?

```
SELECT (COUNT (DISTINCT ?object ) AS ?cd_objects ) WHERE { ?subject ?predicate ?object }
```

(d) How many distinct objects are in a knowledge graph (QB5 of SPORTAL [9]) ?

Fig. 1. Count-distinct queries that time out after 60 s on Wikidata due to quotas.

Table 1. Performance of the count-distinct query QB5 on WDBench [4].

		1s	30s	60s	*Expected*
exact	Blazegraph, Jena	NA	NA	NA[a]	*304,967,140*
	Chao and Lee's [5]	5,909	247,276	561,301	*304,967,140*
approx	**CRAWD**	299,831,155	304,180,989	304,357,330	*304,967,140*
	CRAWD 16 threads	324,940,368	305,837,635	304,798,354	*304,967,140*

[a] Both engines crash after 5 h, running out of memory.

count-distinct aggregate SPARQL queries. The research question thus arises: *Is it possible to build a count-distinct aggregate estimator for SPARQL?*

Distinct-value estimators, originally developed for databases [8,22], estimate the number of distinct values from a sample, ensuring the convergence to the exact value. However, the convergence time to the expected values is very high, making them impractical in the context of RDF and online public SPARQL endpoints. To illustrate, using the real-world WDBench benchmark [4], which includes 1.2 billion triples from Wikidata, it is impossible to obtain results for the query QB5 in Fig. 1d in 60 s with Blazegraph or Apache Jena[1]. In Table 1, we present the results of the Chao-Lee count-distinct estimator [5], a representative state-of-the-art count-distinct estimator. Its provided estimate has reached 561,301 after 60 s. Although the estimate converges, it is still far from the exact value.

In this paper, we introduce CRAWD, the first count-distinct aggregate estimator for SPARQL queries. CRAWD employs a novel approach: each time CRAWD samples a value, it resamples its frequency, i.e., its number of duplicates in the query. We demonstrate that knowing these frequencies is sufficient to estimate the number of distinct values. Since the number of distinct frequen-

[1] Both engines crash after 5 h, running out of memory.

cies can be much lower than the number of distinct values, the overall sampling efficiency of CRAWD is drastically higher than that of existing methods. In our example of Table 1, CRAWD estimates QB5 to 304.3M after 60 s, and 304.7M when parallelizing sampling with 16 threads. Our scientific contributions are twofold:

- We proposed a new count-distinct estimator specifically designed for count-distinct aggregate SPARQL queries. We established its correctness, demonstrating that CRAWD consistently converges to the ground truth. CRAWD can process queries involving basic graph patterns and requires only independent draws instead of uniform sampling from the query results.
- We conducted an extensive experimental study, focusing on the sample efficiency of CRAWD compared to existing estimators. The experimental results indicate that CRAWD significantly outperforms the state-of-the-art approaches by orders of magnitude in terms of sample efficiency.

This paper is organized as follows: Sect. 2 presents preliminaries and motivations. Section 3 describes CRAWD sampling approach. Section 4 presents our experimental results conducted on challenging real and synthetic benchmarks. Sect. 5 describes our positioning compared to related works. Section 6 concludes and outlines future works.

2 Preliminaries and Motivations

We assume the reader is familiar with SPARQL 1.1 concepts and notations [12,17]. In this paper, we focus on count-distinct aggregate conjunctive SPARQL queries Q, represented as $Q = \mathit{Aggregate}(\mathbf{E}, \texttt{COUNTD}, P)$ where $\mathbf{E}$ is a set of variables, and P is a basic graph pattern (BGP), i.e., a set of triple patterns $(tp1 \bowtie tp2 \bowtie \ldots \bowtie tp_n)$. For instance, QB5 is a single triple pattern query and can be represented by $\mathit{Aggregate}(\{?o\}, \texttt{COUNTD}, (?s\ ?p\ ?o))$.

The evaluation of a query Q on an RDF graph G is denoted $[\![Q]\!]_G$ and returns a multiset of mappings where each result mapping μ is a partial function $\mu : V \rightarrow T$, where V are variables and T are terms. For the sake of simplicity, let $[\![Q]\!]_G = [\![\mathit{Aggregate}(\mathbf{E}, \texttt{COUNTD}, P)]\!]_G$, we note: (i) N the number of results of $[\![P]\!]_G$, (ii) D the number of distinct results of $[\![P]\!]_G$, and (iii) F_μ the number of occurrences of a mapping μ projected on variables $\mathbf{E}$ in the results of P: $F_\mu = [\![\mathit{Aggregate}(*, \texttt{COUNT}, \mathit{Filter}(\bigwedge_{v \in \mathbf{E}} v = \mu[v], P))]\!]_G$. In other terms, F_μ is the number of duplicates of μ. For simplicity, when a single variable is projected, we denote this by writing F_t, where t is the term of the projected variable.

Many count-distinct aggregate estimators were developed for databases [8, 22]. To better understand the limitations of existing approaches, we consider a representative: Chao and Lee's count-distinct aggregate estimator [5]. This estimator is versatile and capable of operating on uniform and non-uniform samples, both with and without replacement. Chao and Lee's count-distinct estimator is defined as $\widehat{D}_{CL} = \frac{|S_d|}{\sum_{\mu \in S_d} \frac{F_\mu}{N}}$ where S_d is a sample of distinct values.

Consider a highly skewed graph G_{skew} where the first 5000 triples have o_1 as objects, while the remaining triples each have unique objects $o_2, ..., o_{5001}$, totalling 10000 triples:

$:s_1$ $:p_1$ $:o_1$	$:s_1$ $:p_2$ $:o_2$
$:s_2$ $:p_1$ $:o_1$	$:s_1$ $:p_2$ $:o_3$
...	...
$:s_{5000}$ $:p_1$ $:o_1$	$:s_1$ $:p_2$ $:o_{5001}$

The evaluation $[\![(?s\ ?p\ ?o)]\!]_{G_{skew}}$ returns 10000 solutions mappings, 5001 of which are distinct when projected on the variable `?o`. Therefore $|[\![(?s\ ?p\ ?o)]\!]_{G_{skew}}| = N = 10000$ and $[\![\text{QB5}]\!]_{G_{skew}} = D = 5001$. In the original paper [5], F_μ is assumed to be unavailable. However, in RDF, F_μ can be computed through count queries or approximated count queries [1,13]. In our example, $F_{:o_1} = 5000$, and $F_{:o_2} = F_{:3} = \ldots = F_{:o_{5001}} = 1$. The likely result of four random samples is the following:

(1) $\{?o \rightarrow :o_1\}$ $\quad \widehat{D}_{CL} = \frac{1}{\frac{5000}{10000}} = 2$

(2) $\{?o \rightarrow :o_{57}\}$ $\quad \widehat{D}_{CL} = \frac{2}{\frac{5000}{10000} + \frac{1}{10000}} \approx 4$

(3) $\{?o \rightarrow :o_1\}$ $\quad$ already seen, no change ≈ 4

(4) $\{?o \rightarrow :o_{102}\}$ $\quad \widehat{D}_{CL} = \frac{3}{\frac{5000}{10000} + \frac{1}{10000} + \frac{1}{10000}} \approx 6$

This result highlights several weaknesses of Chao and Lee's estimator: (i) After 4 draws $\widehat{D}_{CL} = 6$, which is very far from 5001. Even by sampling 50% of G_{skew} (2500 times $:o_1$ and 2500 times a value belonging to $:o_2, ..., :o_{5001}$), the best case estimate is $\widehat{D}_{CL} \approx \frac{2501}{\frac{5000}{10000} + \frac{2500}{10000}} \approx 3334.6$, which is still far from 5001. (ii) We have a 50% chance of drawing o_1, which does not result in any progress. Progress only happens when a new distinct value is sampled, which becomes exponentially difficult as the sample size increases. (iii) Chao and Lee's estimator requires remembering all observed distinct values, which may be high and may hit the memory limit of a server.

Although Chao and Lee's count-distinct estimator is well-crafted, like other existing estimators, it is intrinsically slow in the presence of such a skew. This makes such approaches impractical for large RDF datasets.

3 CRAWD: Sampling-Based Count-Distinct Estimator

CRAWD relies on an essential observation: *while the number of distinct values can be extremely high, the number of distinct frequencies of these values is often much lower.* Remarkably, CRAWD maintains its high performance even when faced with skewed data. For example, in G_{skew}, there are 5001 distinct values but only two distinct frequencies: $F_{:o_1} = 5001$ and $F_{:o_2} = ... = F_{:o_{5001}} = 1$. Here, each distinct frequency has a 50% probability of appearing in G_{skew}, 50% for the frequency of 5001, and 50% for the frequency of 1. Similarly, in the DBpedia

dataset in Fig. 2, although there are 13,619,093 distinct objects, there are only 2,141 distinct frequencies. In the following sections, we detail CRAWD; with Definition 1 as starting point:

Definition 1 (Exact count-distinct value). *Given a SPARQL count-distinct query $Q = Aggregate(\boldsymbol{E}, \texttt{COUNTD}, P)$ to be executed over a graph G, and the frequencies of all mappings of P, the exact number of distinct values D is:*

$$D = \sum_{\mu \in [\![P]\!]_G} \frac{1}{F_\mu} \tag{1}$$

When we observe a uniform sample S^{uni} of $\mu \in [\![P]\!]_G$ with their respective frequencies F_μ, we need only to apply a scaling factor to the sum of frequencies to estimate the number of distinct values.

3.1 CRAWD with Uniform Sampling ($\widehat{D}^{uni}$)

Definition 2 (Count-distinct estimator for uniform sampling). *Given a query $Q = Aggregate(\boldsymbol{E}, \texttt{COUNTD}, P)$ to be executed over a graph G, and a uniform sample S^{uni} of the results $[\![P]\!]_G$, the count-distinct estimator $\widehat{D}^{uni}$ is defined as:*

$$\widehat{D}^{uni} = \frac{\widehat{N}}{|S^{uni}|} \cdot \sum_{\mu \in S^{uni}} \frac{1}{\widehat{F}_\mu} \tag{2}$$

The first part of the equation scales the observations made by the multiset sample S^{uni} to the estimated size of the results $\widehat{N}$. The second part of the equation weights each element of the sample with the inverse number of duplicates.

To illustrate, we use the skewed example of QB5 over Graph G_{skew}, assuming an identical sample we obtain:

(1) $\{?o \rightarrow \texttt{:o}_1\}$ $\quad \widehat{D}^{uni} = \frac{10000}{1} \times \frac{1}{5000} = 2$
(2) $\{?o \rightarrow \texttt{:o}_{57}\}$ $\quad \widehat{D}^{uni} = \frac{10000}{2} \times (\frac{1}{5000} + \frac{1}{1}) = 5001$
(3) $\{?o \rightarrow \texttt{:o}_1\}$ $\quad \widehat{D}^{uni} = \frac{10000}{3} \times (\frac{1}{5000} + \frac{1}{1} + \frac{1}{5000}) \approx 3333$
(4) $\{?o \rightarrow \texttt{:o}_{102}\}$ $\quad \widehat{D}^{uni} = \frac{10000}{4} \times (\frac{2}{5000} + \frac{2}{1}) = 5001$

The first estimate is far from the expected count-distinct, like Chao and Lee's estimate. However, the second estimate has already reached it. It is normal as soon as CRAWD observes the distinct frequencies with corresponding probability to appear, it delivers the exact value. Unfortunately, it is impossible to know that all other draws are useless. However, it is important to converge to the exact value. The third estimate drifts away from the expected value but remains much closer than the first. The fourth estimate reaches the actual value again. Over the samples, the relative error decreases quickly. Eventually, CRAWD's estimate reaches the expected count-distinct value of 5001.

Theorem 1. *$\widehat{D}^{uni}$ is an unbiased estimator of D.*

Proof. We must prove that $\mathbb{E}[\widehat{D}^{uni}] = D$.

$$\mathbb{E}[\widehat{D}^{uni}] = \mathbb{E}\left[\frac{\widehat{N}}{|S^{uni}|} \cdot \sum_{\mu \in S^{uni}} \frac{1}{\widehat{F}_\mu}\right] = \frac{N}{|S^{uni}|} \cdot \mathbb{E}\left[\frac{1}{\widehat{F}_{\mu_1}} + \ldots + \frac{1}{\widehat{F}_{\mu_1}} + \frac{1}{\widehat{F}_{\mu_2}} + \ldots + \frac{1}{\widehat{F}_{\mu_d}}\right]$$

$$= \frac{N}{K \times N} \cdot \mathbb{E}\left[\underbrace{\frac{1}{\widehat{F}_{\mu_1}} + \ldots + \frac{1}{\widehat{F}_{\mu_1}}}_{K \times F_{\mu_1} \text{ times}} + \frac{1}{\widehat{F}_{\mu_2}} + \ldots + \frac{1}{\widehat{F}_{\mu_d}}\right] \text{with } |S^{uni}| = K \times N$$

$$= \frac{1}{K} \cdot \mathbb{E}[K \times \underbrace{(1 + 1 + \ldots + 1)}_{D \text{ times}}] = \frac{K \times D}{K} = D$$

□

Assuming a sufficiently large uniform sample of the results (typically the scaling factor $K \ll 1$ as shown in Sect. 4), our count-distinct estimator $\widehat{D}^{uni}$ converges towards the exact value. However, drawing a uniform sample over joins – especially in graphs – is tedious and often requires to reject large portions of the sampled results, resulting in poor sample efficiency [23]. Inspired by Horvitz-Thompson estimators [11], we devise an unbiased count-distinct estimator for *non-uniform* samples.

3.2 CRAWD with Non-uniform Sampling ($\widehat{D}^{non}$)

Contrarily to uniform sampling where each element has $P_{\mu_1} = P_{\mu_2} = \ldots = P_{\mu_n} = \frac{1}{N}$ chances to get picked in the sample, each mapping μ of the results may have a different probability P_μ to get picked in the sample. The probability of drawing an element is *not* reflected by its number of occurrences in the results anymore. An element with many duplicates may have few chances to appear in the sample. To counterbalance this bias, we introduce P_μ in the formula.

Definition 3 (Count-distinct estimator for non-uniform sampling). *Given a query $Q = Aggregate(\boldsymbol{E}, \texttt{COUNTD}, P)$ to execute over a graph G, a non-uniform sample S^{non} of the results $[\![P]\!]_G$ where each mapping μ has a probability P_μ of having been picked, the count-distinct estimator $\widehat{D}^{non}$ is:*

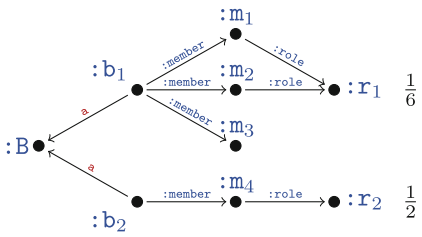
$$\widehat{D}^{non} = \frac{\widehat{N}}{\sum_{\mu \in S^{non}} P_\mu^{-1}} \cdot \sum_{\mu \in S^{non}} \frac{P_\mu^{-1}}{\widehat{F}_\mu} \tag{3}$$

To illustrate, consider the query Q^{role}_{bgp} of Fig. 2a. Q^{role}_{bgp} can be presented by $Q^{role}_{bgp} = Aggregate(role, \texttt{COUNTD}, P)$ where P is a BGP comprising three triple patterns. The evaluation $[\![P]\!]_{G_{ex}}$ returns 3 results:

(1) $[\langle :\texttt{b}_1\texttt{a:B}\rangle, \langle :\texttt{b}_1\texttt{:member:}\,\texttt{m}_1\rangle, \langle :\texttt{m}_1\texttt{:role:}\,\texttt{r}_1\rangle]$, $(?role \rightarrow :\texttt{r}_1)$;
(2) $[\langle :\texttt{b}_1\texttt{a:B}\rangle, \langle :\texttt{b}_1\texttt{:member:}\,\texttt{m}_2\rangle, \langle :\texttt{m}_2\texttt{:role:}\,\texttt{r}_1\rangle]$, $(?role \rightarrow :\texttt{r}_1)$;
(3) $[\langle :\texttt{b}_2\texttt{a:B}\rangle, \langle :\texttt{b}_2\texttt{:member:}\,\texttt{m}_4\rangle, \langle :\texttt{m}_4\texttt{:role:}\,\texttt{r}_2\rangle]$; $(?role \rightarrow :\texttt{r}_2)$.

```
SELECT (COUNT (DISTINCT ?role ) AS ?roles )
WHERE {
  ?band a :B .                          #tp1
  ?band :member ?member .               #tp2
  ?member :role ?role                   #tp3
}
```

(a) Query Q^{role}_{bgp} that requests the number of roles and the number of distinct roles in bands, respectively. In Graph G_{ex}, there are three roles but only two distinct roles.



(b) Graph representation of G_{ex}.

Fig. 2. A simple conjunctive query with 3 triple patterns.

We assume to have a non-uniform sample of $[\![P]\!]_{G_{ex}}$ over the graph G_{ex} represented in Fig. 2b, along with their associated probabilities:

(1) $[\langle{:}\,\mathtt{b_1}\,\mathtt{a}\,{:}\mathtt{B}\rangle, \langle{:}\,\mathtt{b_1}\,\mathtt{:member}\,{:}\,\mathtt{m_1}\rangle, \langle{:}\,\mathtt{m_1}\,\mathtt{:role}\,{:}\,\mathtt{r_1}\rangle]$, $P_{\mu_1} = (\frac{1}{2} \times \frac{1}{3} \times 1 = \frac{1}{6})$;
(2) $[\langle{:}\,\mathtt{b_2}\,\mathtt{a}\,{:}\mathtt{B}\rangle, \langle{:}\,\mathtt{b_2}\,\mathtt{:member}\,{:}\,\mathtt{m_4}\rangle, \langle{:}\,\mathtt{m_4}\,\mathtt{:role}\,{:}\,\mathtt{r_2}\rangle]$; $P_{\mu_2} = (\frac{1}{2} \times 1 \times 1 = \frac{1}{2})$;
(3) $[\langle{:}\,\mathtt{b_2}\,\mathtt{a}\,{:}\mathtt{B}\rangle, \langle{:}\,\mathtt{b_2}\,\mathtt{:member}\,{:}\,\mathtt{m_4}\rangle, \langle{:}\,\mathtt{m_4}\,\mathtt{:role}\,{:}\,\mathtt{r_2}\rangle]$; $P_{\mu_3} = (\frac{1}{2} \times 1 \times 1 = \frac{1}{2})$;

In this sample, $:\mathtt{r_2}$ appears more often than $:\mathtt{r_1}$ despite appearing half as often in the query results $[\![P]\!]_{G_{ex}}$. Assuming accurate estimates for $\widehat{N}$ and $\widehat{F}_\mu$, we obtain the following count-distinct estimate:

$$\widehat{D}^{non} = \frac{3}{P_{\mu_1}^{-1} + P_{\mu_2}^{-1} + P_{\mu_3}^{-1}} \times \left(\frac{P_{\mu_1}^{-1}}{F_{:\mathtt{r_1}}} + \frac{P_{\mu_2}^{-1}}{F_{:\mathtt{r_2}}} + \frac{P_{\mu_3}^{-1}}{F_{:\mathtt{r_2}}} \right) = \frac{3}{6+2+2} \times \left(\frac{6}{2} + \frac{2}{1} + \frac{2}{1} \right) = \frac{21}{10}$$

Again, increasing the sample size increases the accuracy of the estimate. It eventually converges to the exact count-distinct value.

Theorem 2 *$\widehat{D}^{non}$ is an unbiased estimator of D.*

Proof The demonstration is analogous to that of Theorem 1 with probability of appearance P_μ being neutralized by their respective weights P_μ^{-1}. □

3.3 Implementing CRAWD

Algorithm 1 provides the general instructions for computing count-distinct estimates using CRAWD under budget b. CRAWD requires (i) (ii) a count estimator for $\widehat{N}$ under budget b_N (Line 3); (iii) a sampler returning each sampled mapping $\mu \in [\![P]\!]_G$ with its inverse probability of having been picked P_μ^{-1} (Line 5); and (iv) a count estimator for $\widehat{F}_\mu$ under budget $b_{\widehat{F}_\mu}$ (Line 9). Sampled solutions become part of the estimate by updating the two sums on Lines 10 and 11 until the cost c exceeds the budget b. The function finally returns an estimate based on Eq. 3.

While this algorithm highlights that any sampling method and any count estimator [16] would work, this paper focuses on the implementation of Algorithm 2 based on RAW-JENA [1] for non-uniform sampling with replacement, and on Wander Join [13] for its count estimates.

Algorithm 1: Generic implementation of CRAWD.

1 **Function** CRAWD$(G, P, E, b, b_N, b_{F_\mu})$:
2 $(sum_{P_\mu^{-1}}, sum_{P_\mu^{-1}/F_\mu}) \leftarrow (0, 0)$
3 $(\widehat{N}, c) \leftarrow$ CountEstimate(G, P, b_N)
4 **while** $c < b$ **do**
5 $(\mu, P_\mu^{-1}, c_\mu) \leftarrow$ Sample(G, P)
6
7 // Sample provides only $\mu \in [\![P]\!]_G$
8 $P' \leftarrow Filter(\bigwedge_{v \in E} v = \mu[v], P)$
9 $(\widehat{F}_\mu, c_{F_\mu}) \leftarrow$ CountEstimate(G, P', b_{F_μ})
10 $sum_{P_\mu^{-1}} \leftarrow sum_{P_\mu^{-1}} + P_\mu^{-1}$
11 $sum_{P_\mu^{-1}/F_\mu} \leftarrow sum_{P_\mu^{-1}/F_\mu} + \frac{P_\mu^{-1}}{\widehat{F}_\mu}$
12 $c \leftarrow c + c_\mu + c_{F_\mu}$
13 **if** $sum_{P_\mu^{-1}} = 0$ **then return** 0
14 **return** $\frac{\widehat{N}}{sum_{P_\mu^{-1}}} \times sum_{P_\mu^{-1}/F_\mu}$

Algorithm 2: CRAWD based on random walks and Wander Join.

1 **Function** CRAWD$(G, P, E, b, b_N, b_{F_\mu})$:
2 $(S_{wj}, sum_{P_\mu^{-1}/F_\mu}) \leftarrow (0, 0)$
3 $c \leftarrow 0$ // b_N included in the loop
4 **while** $c < b$ **do**
5 $(\mu, P_\mu^{-1}, c_\mu) \leftarrow$ RandomWalk(G, P)
6 $S_{wj} \leftarrow S_{wj} + 1$
7 **if** $\mu \in [\![P]\!]_G$ **then**
8 $P' \leftarrow Filter(\bigwedge_{v \in E} v = \mu[v], P)$
9 $(\widehat{F}_\mu, c_{F_\mu}) \leftarrow$ WanderJoin(G, P', b_{F_μ})
10 $sum_{P_\mu^{-1}/F_\mu} \leftarrow sum_{P_\mu^{-1}/F_\mu} + \frac{P_\mu^{-1}}{\widehat{F}_\mu}$
11 $c \leftarrow c + c_{F_\mu}$
12 $c \leftarrow c + c_\mu$
13 **if** $S_{wj} = 0$ **then return** 0
14 **return** $\frac{1}{S_{wj}} \times sum_{P_\mu^{-1}/F_\mu}$
// simplified

Sampling. RAW-JENA uses random walks to provide mappings μ along with their exact probability of getting picked P_μ. For a conjunctive query $Q = (tp_1 \bowtie tp_2 \bowtie \dots tp_n)$, a random walk $\gamma_i = (t_1, \dots, t_n)$ is computed by randomly picking t_1 in $[\![tp_1]\!]_G$, and each subsequent t_i $(i > 1)$ in $[\![t_{i-1} \bowtie tp_i]\!]_G$.

For example, assuming an augmented B-Tree indexes such as Blazegraph's, which enables picking a triple uniformly at random from a triple pattern, and retrieving the cardinality of a triple pattern; with the query Q_{bgp}^{role} on Graph G_{ex} of Fig. 2b, Line 5 of Algorithm 2 performs a first random walk:

(1) $\gamma_1 = [\langle : \mathtt{b}_1\ \mathtt{a}\ :\mathtt{B}\rangle, \langle : \mathtt{b}_1\ :\mathtt{member} : \mathtt{m}_1\rangle, \langle : \mathtt{m}_1\ :\mathtt{role} : \mathtt{r}_1\rangle]$
(2) $\mu_1 = \{?role \rightarrow : \mathtt{r}_1\}$
(3) $P_{\mu_1}^{-1} = |[\![?b\,\mathtt{a}\,:\mathtt{B}]\!]_{G_{ex}}| \times |[\![: \mathtt{b}_1\,:\mathtt{member}\,?m]\!]_{G_{ex}}| \times |[\![: \mathtt{m}_1\,:\mathtt{role}\,?r]\!]_{G_{ex}}| = 2 \times 3 \times 1 = 6$

Since the found mapping μ_1 is a valid solution of Query P over Graph G_{ex}, Line 7 of Algorithm 2 states that CRAWD should evaluate $\widehat{F}_{\mu_1}$ to update its dedicated sum. Therefore, Line 8 binds the variable $?role$ of Query P with the found mapping $: \mathtt{r}_1$ in order to evaluate $\widehat{F}_{\mu_1}$ as a regular `COUNT` query: $Aggregate(*, \mathtt{COUNT}, (?b\ \mathtt{a}\ :\mathtt{B} \bowtie ?b\ :\mathtt{member}\ ?m \bowtie ?m\ :\mathtt{role} : \mathtt{r}_1))$.

Estimating $\widehat{F}_\mu$. According to the G-Care Benchmark [16], Wander Join [13] is an accurate and unbiased cardinality estimator on RDF data. Being unbiased, increasing the dedicated budget b_{F_μ} improves the expected accuracy of the provided estimates. Assuming a conjunctive query $Q = (tp_1 \bowtie tp_2 \bowtie \dots tp_n)$ and a set of a random walks $S_{wj} = \{(t_1, \dots, t_n)\}$, Wander Join estimates the cardinality $\widehat{N}_{wj}$ of a query Q as follows:

$$\widehat{N}_{wj} = \sum_{\gamma \in S_{wj}} \frac{P_\gamma^{-1}}{|S_{wj}|} \text{ where } P_\gamma^{-1} = \begin{cases} |[\![tp_1]\!]| \prod_{i=2}^{n} |[\![t_{i-1} \bowtie tp_i]\!]| & \text{if } \gamma \text{ succeeded,} \\ 0 & \text{if } \gamma \text{ failed.} \end{cases}$$

For example, with a budget b_{F_μ} of two random walks, and the built query $P' = (?b\ \texttt{a}\ \texttt{:B} \bowtie ?b\ \texttt{:member}\ ?m \bowtie ?m\ \texttt{:role}\ \texttt{:r}_1)$, Wander Join on Line 9 computes $\widehat{F}_{\mu_1}$ as follows:

(1) $[\langle \texttt{:b}_1\ \texttt{a}\ \texttt{:B}\rangle, \langle \texttt{:b}_1\ \texttt{:member}\ \texttt{:m}_1\rangle, \langle \texttt{:m}_1\ \texttt{:role}\ \texttt{:r}_1\rangle]$, $\widehat{F}_{\mu_1} = \frac{2\times 3\times 1}{1} = 6$
(2) $[\langle \texttt{:b}_1\ \texttt{a}\ \texttt{:B}\rangle, \langle \texttt{:b}_1\ \texttt{:member}\ \texttt{:m}_3\rangle]$, $\widehat{F}_{\mu_1} = \frac{6+0}{2} = 3$

Line 10 of Algorithm 2 aggregates this result into $sum_{P_\mu^{-1}/F_\mu} = \frac{6}{3} = 2$. At this point, if the cost c exceeds the budget b, CRAWD's estimate is $\widehat{D}^{non} = \frac{2}{1} = 2$. Contrary to Line 14 of the generic Algorithm 1, the term $\widehat{N}$ does not appear. By using Wander Join to estimate $\widehat{N}$ as well, we have $S^{non} \subset S_{wj}$, therefore $\sum_{\mu\in S_{wj}} P_\mu^{-1} = \sum_{\mu\in S^{non}} P_\mu^{-1}$ which simplifies $\frac{\widehat{N}_{wj}}{\sum_{\mu\in S^{non}}}$ by $\frac{1}{|S_{wj}|}$.

Overall. CRAWD provides a range of different implementations, allowing for trade-offs depending on (i) the chosen approach to estimate N, (ii) the chosen approach to estimate F_μ, and (iii) the chosen approach to provide sampled mappings μ. Being unbiased, increasing CRAWD's budget improves its accuracy. For most queries, CRAWD converges quickly towards the expected value, even in highly skewed settings where other estimators might fail. However, some complex queries may require budget adjustments as demonstrated in our experimental results.

4 Experimental Study

We want to answer the following questions empirically:

1. What is the sample efficiency of CRAWD compared to other count-distinct estimators for single triple pattern queries? For these queries, provided F_μ are exact thanks to the indexes of SPARQL engines. Therefore, this experiment shows the performance of count-distinct estimators without the noise induced by underlying count estimators.
2. What is the sample efficiency of CRAWD for basic graph pattern queries? The presence of joins makes the estimation of F_μ approximate. This experiment measures how the approximation of F_μ impacts CRAWD.

CRAWD is implemented in Java on top of Blazegraph's storage. The code for reproducible experiments is public and available on the GitHub platform at: https://github.com/GDD-Nantes/crawd-experiments.

4.1 Experimental Setup

Datasets: We experiment on the three datasets summarized in Table 2:
- *WatDiv10M*: A synthetic dataset with 10 million triples based on Waterloo SPARQL Diversity Test Suite [3].
- *DBpedia*: A subset of DBpedia within LargeRDFBench [19] sourced from Wikipedia and containing around 43 million triples.

Table 2. Characteristics of datasets (NA: Not Available).

	triples	distinct (value/frequency)		
		subjects	predicates	objects
WatDiv10M	10,916,457	521,585/308	86/74	1,005,832/641
DBpedia	42,849,609	9,494,331/121	1,058/739	13,619,093/2,141
WDBench	1,257,169,959	92,498,623/NA	8,604/8,195	304,967,140/NA

– *WDBench* [4]: A real-world large dataset extracted from Wikidata containing around 1.25 billion triples.

Queries: As there is no dedicated benchmark for count-distinct aggregate queries, we used the synthetic benchmark WatDiv and the real-world benchmark WDBench to build two workloads. We transformed the original queries from these benchmarks into count-distinct aggregate queries and selected the most challenging queries for the estimators, i.e., those producing a large number of distinct results. For both workloads, we added single triple pattern queries from SPORTAL [9]: QB3 (count-distinct predicates), QB4 (count-distinct subjects), and QB5 (count-distinct objects), the most basic VoID statistics[2]. From SPORTAL, we added ten queries that count distinct objects per class for the top 10 classes with the highest number of distinct objects per class.
The result is a workload of 58 queries ranging from 1 to 12 triple patterns for WatDiv and a workload of 43 queries from 1 to 5 triple patterns for WDBench. The methodology used to build our workloads is detailed on the project website.

Baseline Estimators: For single pattern count-distinct queries, we compare the performance of CRAWD with the following count-distinct estimators [8]:

– *Horvitz-Thompson* [11,20]: is a specialisation of the general Horvitz-Thompson statistical estimator to count distinct items. In this case, the challenge is to compute the inclusion probability of a distinct element. This specialisation is different from the Horvitz-Thompson estimator for cardinality estimation used in Wander Join [13].
– *Smoothed Jackknife* [8]: accounts for true bias structures.
– *Chao-Lee* [5]: implements the Chao and Lee estimator using a natural coverage estimator that assumes F_μ to be unavailable.
– *Chao-Lee-F_μ* [23]: is a variant of Chao-Lee where F_μ is computed using rewriting and cardinality estimation. Out of fairness, it uses the same cardinality estimator as CRAWD.
– *NDV estimator* [22]: is a learned estimator. According to the authors, the model has been trained with synthetically generated training data and can be deployed on unseen tables and workloads.

For BGP count-distinct queries, we only compare CRAWD and Chao-Lee-F_μ as they are the sole representative supporting non-uniform sampling over joins [23].

Evaluation Metrics: We measure the *mean relative error* over five runs to evaluate the accuracy of count-distinct estimators. The relative error for an

[2] https://www.w3.org/TR/void/.

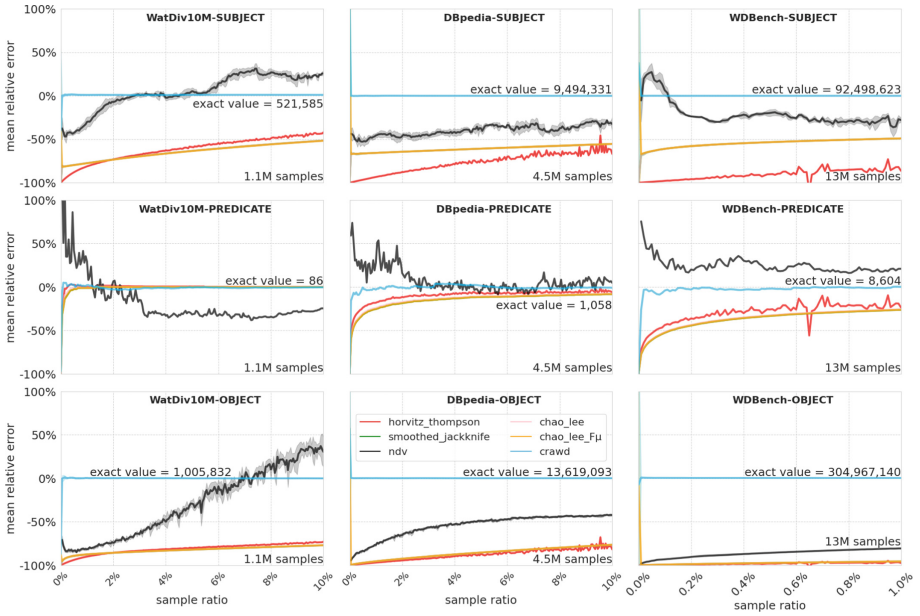


Fig. 3. Relative error of count-distinct estimators for single triple pattern queries.

estimation $\widehat{D}$ with the ground-truth value D is defined as $\frac{\widehat{D}-D}{D} \times 100$. For single triple pattern count-distinct queries, we keep the sign of relative errors to highlight the over estimations and underestimations of reference values.
The *sample efficiency* is given by comparing the relative error of two count-distinct estimators for the same sample size.
Hardware: We use a local cloud instance with Ubuntu 20.04.4.LTS, an AMD EPYC 7513-Core processor with 16 vCPUs allocated to the VM, 1 TB SSD, and 64 GB of RAM.

4.2 Experimental Results

What is the Sample Efficiency of CRAWD Compared to Other count-Distinct Estimators for Single Triple Pattern Queries? For single triple pattern queries, provided $\widehat{N}$ and F_μ are exact, thanks to the indexes of the SPARQL engines. Therefore, this experimentation shows the performance of the estimator itself without the noise induced by underlying count estimators. Therefore, the budget b_{F_μ} dedicated to count estimates for both Chao-Lee-F_μ and CRAWD is 0.

To enable fair comparison, we draw for all different approaches uniformly with replacement 10% of subjects, predicates, and objects from WatDiv10M and DBpedia, and 1% of subjects, predicates, and objects from WDBench. Concerning WDBench, we limit the sampling ratio to 1% as it represents 12M triples, which is already greater than the whole WatDiv10M dataset.

Figure 3 presents the sample efficiency for our three single pattern queries over our three datasets. On the x-axis, we have the sampling ratio, and on the y-axis the mean relative error.

The overall results show that whatever the query and the dataset, CRAWD converges to the ground truth extremely fast compared to other estimators, i.e., with 1% of sampling ratio, CRAWD has a relative error under 2%.

When we examine the queries, the performance disparity between CRAWD and the competitors is less pronounced in queries about distinct predicates (QB3). This is normal as the number of distinct frequencies for predicates is closer to the number of distinct values as described in Table 2.

However, the performance gap significantly increases when counting the distinct subjects and reaches its peak in queries involving distinct objects. The primary reason is the varying number of distinct elements (see Table 2). The performance of existing estimators tends to degrade as the number of distinct elements increases. CRAWD performs extremely well as the number of distinct frequencies is significantly lower compared to the number of distinct elements[3].

Focusing on the different datasets, it is evident that changes in dataset size predominantly affect the performance of counting the distinct predicates. For example, WatDiv10M has 86 predicates, DBpedia has 1,058 predicates, and WDBench has 8,604 predicates. Consistent with earlier findings, the performance of existing estimators deteriorates as the number of distinct elements in a dataset increases, leading to slower convergence.

Focusing on the estimators, the performance of Chao-Lee and Chao-Lee-F_μ is comparable, indicating that the two methodologies for estimating F_μ yield similar results. The performance of Smoothed Jackknife is also akin to that of Chao-Lee-F_μ. However, the performance of Horvitz-Thompson appears to deteriorate more rapidly than that of Chao-Lee's as the number of distinct elements increases. Finally, the performance of NDV is erratic. Being a learned estimator, it remains unclear whether NDV's inconsistent results are due to generalization issues.

In summary, for single triple pattern queries, CRAWD outperforms other estimators on all the datasets for count-distinct subjects, predicates, and objects.

What is the Sample Efficiency of CRAWD in the Presence of Basic Graph Patterns? For the BGP count-distinct queries, we only compare CRAWD and Chao-Lee-F_μ, as they are the sole representatives of count-distinct estimators that support non-uniform sampling over joins [23].

To ensure a fair comparison, each count-distinct aggregate query is executed with an identical global budget b in terms of number of scans over Blazegraph's indexes. For this experiment, the global budget b is set to 10K, 100K, and 1M scans. This budget includes a frequency budget $b_{\widehat{F}_\mu}$, which is the number of scans allocated for frequency estimation F_μ per sampled result μ. In the experiment, the frequency budget $b_{\widehat{F}_\mu}$ is set to 1, 10, 100 multiplied by the number of triple

[3] We were unable to compute frequencies of distinct subjects and objects for WDBench with our computing resources.

(a) Workload of WatDiv10M. (b) Workload of WDBench.

Fig. 4. Aggregated mean relative error for the different combinations of $b/b_{\widehat{F}_\mu}$.

patterns in the query. For instance, a query with two triple patterns will consume 2, 20, or 200 scans to estimate each $\widehat{F}_\mu$, depending on the configuration. Overall, this represents nine configurations with different budget allocations b and $b_{\widehat{F}_\mu}$ for each query.

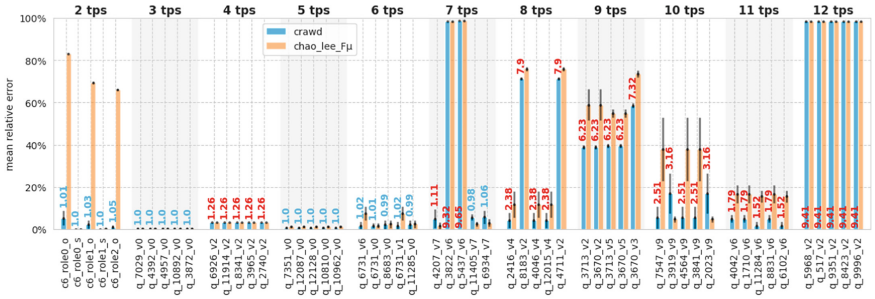
Figure 4 presents the average relative error for all queries per configuration. As expected, the error decreases as the budget for sampling increases for both Chao-Lee-F_μ and CRAWD. In this experiment, CRAWD consistently outperforms Chao-Lee-F_μ.

For a fixed budget, the average relative error also decreases as the frequency budget increases. However, an increased frequency budget results in a decrease in the number of sampled results. There is a trade-off between the overall budget and the frequency budget, which can vary depending on the query.

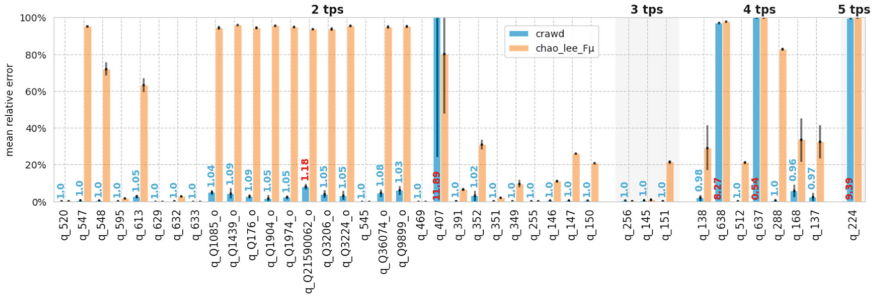
By comparing the results on WatDiv10M and WDBench, we observe that CRAWD and Chao-Lee-F_μ perform better on WDBench, which is 100 times larger than WatDiv10M. This is due to the nature of the WatDiv workload, which includes queries with up to 12 triple patterns. Queries with a high number of triple patterns tend to deliver less accurate estimations, thereby degrading the overall performance of the WatDiv workload.

Figure 5 presents the results per query for each workload for $b = 1M$ and $b_{\widehat{F}_\mu} = \#tp \times 100$. The queries are grouped on the x-axis by their number of triple patterns. As shown, CRAWD significantly outperforms Chao-Lee-F_μ in most cases. For some queries, both estimators have similar performance, and only a few times Chao-Lee-F_μ performs better than CRAWD: `q_2023_v9`, `q_3919_v9` on WatDiv, and `q_407` for WDBench.
When CRAWD estimates poorly, there can be a few causes:

1. The estimation of F_μ is performed with random walks, making join ordering critical. An incorrect join order negatively impacts CRAWD's estimation. We observed this issue with the queries `q_407` and `q_637`. After forcing the good join order, CRAWD delivered nearly perfect estimations.



(a) Mean relative errors of 55 queries for WatDiv10M.



(b) Mean relative errors of 40 queries for WDBench.

Fig. 5. Mean relative errors of estimates provided by CRAWD and Chao-Lee-F_μ with a global budget $b = 1M$ and a frequency budget $b_{\widehat{F}_\mu} = \#tp \times 100$.

2. Compared to the original query, the query for F_μ can be highly selective. For instance, to estimate the query $Q = Aggregate(\{?y\}, \texttt{COUNTD}, (\texttt{:foo :knows } ?x \bowtie ?x \texttt{ :knows } ?y))$, with a sampled mapping $\{y \rightarrow \texttt{:bar}\}$, we generate the frequency query $F_{\texttt{:bar}} = Aggregate(*, \texttt{COUNT}, (\texttt{:foo :knows } ?\text{x} \bowtie ?\text{x } \texttt{:knows :bar}))$. If $F_{\texttt{:bar}}$ is highly selective, it may require more scans than the allocated frequency budget to be estimated accurately. The queries `q_224` with four triple patterns and `q_638` with five triple patterns of WDBench have this issue.

Regardless of the causes of poor estimation accuracy, we know that F_μ is an unbiased cardinality estimator. It eventually converges to the expected values. Consequently, CRAWD also eventually converges to the correct value. The key question is: *What budget is required to achieve such a convergence?*

Given an overall budget b and a frequency budget $b_{\widehat{F}_\mu}$, we can assess if they need to be increased by comparing the provided count-distinct estimates with the estimates provided by a smaller configuration with a lower global budget and a smaller frequency budget.

To illustrate, consider the query `q_9962_v2` with 12 triple patterns of WatDiv that should return 8,321. With $b = 1M$ and $b_{\widehat{F}_\mu} = 1{,}200$, the estimate is approx-

imately 130. With a smaller $b = 100K$ and $b_{\widehat{F}_\mu} = 120$, the estimate is approximately 14. The ratio between these two estimates is $\frac{130}{14} \approx 9.2$ indicates that the estimation still evolves significantly and requires more sampling. Conversely, the query `q_137` of WDBench should return 223,432. CRAWD returns 218,308.8 with $b = 1M$ and $b_{\widehat{F}_\mu} = 400$; and 202,951.6 with $b = 10K$ and $b_{\widehat{F}_\mu} = 40$. The ratio is 1.07, indicating a stable value.

In Fig. 5, we computed for CRAWD, the estimation ratio for all queries, dividing the distinct values obtained with $b = 1M$ and $b_{\widehat{F}_\mu} = \#tp \times 100$ by the distinct values obtained with $b = 1M$ and $b_{\widehat{F}_\mu} = \#tp \times 10$. The estimation ratio appears on top of the CRAWD bars. As we can see, when the estimation ratio is high, the relative error is high, and when the estimation ratio is close to 1, the relative error is low.

To confirm the correlation, we calculated the Spearman's correlation coefficient for both datasets, WatDiv10M and WDBench, with the two configurations: $b = 1M, b_{\widehat{F}_\mu} = 100$ and $b = 100K, b_{\widehat{F}_\mu} = 10$. The coefficients are 0.87 and 0.57, respectively, with p-values close to 0. This statistically significant result confirms the correlation between the estimation ratio and relative error. Consequently, an end-user can utilize CRAWD by progressively increasing the budget until stable values are found.

5 Related Work

Executing count-distinct aggregate SPARQL queries poses significant challenges, as they can quickly exhaust quotas in time and memory of public SPARQL endpoints. Consequently, the execution of count-distinct aggregate queries often gets interrupted by fair use policies [9,15].

To address the memory issue, one can use sketch-based approaches such as HyperLogLog [6,10], which offer accurate estimates with error-bound. While sketches resolve the memory issue, they are still time-consuming as they require scanning all results of the count-distinct aggregate queries. One strategy to overcome the timeout issue is to rely on web preemption [2,7]. Web preemption allows splitting the processing of count-distinct aggregate queries into several quanta of time, making the processing compatible with fair use policies. Unfortunately, web preemption still requires to scan all results of the count-distinct aggregate queries.

Online Approximative Query Processing (AQP) [14] is an alternative approach for computing aggregate queries. Unlike sketches, AQP does not require scanning all results of a count-distinct aggregate query, i.e., results can be delivered by scanning only a tiny fraction of the dataset. However, it cannot guarantee error-bound as sketches; the only guarantee of AQP is to converge to the correct result in a pay-as-you-go scenario. AQP has already been successful for SPARQL aggregate queries but without the solution modifier `DISTINCT` [1,18,21]. These aggregates can be handled with random walks as proposed in Wander Join [13]. However, it cannot handle count-distinct aggregate queries as the Wander Join statistical estimator is not designed to count distinct values. To the best of

our knowledge, CRAWD is the first proposal to sample SPARQL count-distinct queries.

In the field of databases, a large number of count-distinct estimators are proposed [8,22]. These estimators are used to estimate the number of distinct values in a column of a table. As highlighted in this paper, these approaches deliver very poor sample efficiency in the context of RDF, i.e., in the presence of a table with three columns: subject – predicate – object. CRAWD outperforms other count-distinct estimators in this context. Moreover, existing databases count-distinct estimators are not evaluated over query results, mainly to avoid the challenging problem of sampling over joins [23]. When sampling over joins, results become non-uniform, and many count-distinct estimators cannot be used. Different algorithms have been proposed to draw uniform samples over joins [23], such as online exploration, Extended Olken (EO), and exact weight.

However, providing uniform sampling over joins has a very high cost in terms of sample efficiency [23]. CRAWD avoids this issue by using a non-uniform sampling. This is an essential feature for making the whole approach usable in practice.

6 Conclusion

In this paper, we proposed CRAWD, the first count-distinct estimator for SPARQL queries. By leveraging the relatively small number of distinct frequencies compared to distinct values, CRAWD significantly improves sample efficiency, even in the presence of joins. Our comprehensive experimental evaluations empirically demonstrate that CRAWD significantly outperforms state-of-the-art approaches in terms of sample efficiency. In addition, we provide an engine that runs on Blazegraph storage, enabling Blazegraph and CRAWD to be used on the same physical data.

Our future work will focus on extending the support of our approximate count-distinct estimator CRAWD to more complex operators such as `OPTIONAL`, `SET MINUS`, and `NOT EXISTS`. Our experiments also highlighted a trade-off in CRAWD's sampling budget allocation between result sampling and sampling cardinality estimation of results. Exploring an intelligent strategy for optimizing this allocation is an interesting direction for future research. Finally, combining CRAWD with a cardinality estimator with better performance than Wander Join [13] could further enhance its performance.

Supplemental Material Statement: All datasets and code required to run and reproduce the experiments are publicly available on the GitHub platform at: https://github.com/GDD-Nantes/crawd-experiments.

Acknowledgements. This work is funded by the French ANR project MeKaNo (ANR-22-CE23-0021) and the French Labex CominLabs project MiKroloG (The Microdata Knowledge Graph).

References

1. Aimonier-Davat, J., Nédelec, B., Dang, M.H., Molli, P., Skaf-Molli, H.: RAW-JENA: approximate query processing for SPARQL endpoints. In: 22nd International Semantic Web Conference, ISWC 2023, Athens, Greece, 6–10 November 2023 (2023)
2. Aimonier-Davat, J., Skaf-Molli, H., Molli, P., Grall, A., Minier, T.: Online approximative SPARQL query processing for COUNT-DISTINCT queries with Web Preemption. Semantic Web - Interoperability, Usability, Applicability (2022)
3. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 197–212. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_13
4. Angles, R., Aranda, C.B., Hogan, A., Rojas, C., Vrgoč, D.: WDBench: a Wikidata graph query benchmark. In: Sattler, U., et al. (eds.) The Semantic Web, ISWC 2022. LNCS, vol. 13489, pp. 714–731. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19433-7_41
5. Chao, A., Lee, S.M.: Estimating the number of classes via sample coverage. J. Am. Stat. Assoc. **87**(417), 210–217 (1992)
6. Flajolet, P., Fusy, É., Gandouet, O., Meunier, F.: HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In: Conference on Analysis of Algorithms, AofA 07, DMTCS Proceedings, AH, pp. 137–156. Discrete Mathematics and Theoretical Computer Science, Juan les Pins, France (2007)
7. Grall, A., Minier, T., Skaf-Molli, H., Molli, P.: Processing SPARQL aggregate queries with web preemption. In: Kirrane, S., et al. (eds.) ESWC 2020. LNCS, vol. 12123, pp. 235–251. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49461-2_14
8. Haas, P.J., Naughton, J.F., Seshadri, S., Stokes, L.: Sampling-based estimation of the number of distinct values of an attribute. In: Proceedings of VLDB'95: 21th International Conference on Very Large Data Bases, Zurich, Switzerland, 11–15 September 1995, pp. 311–322. Morgan Kaufmann (1995)
9. Hasnain, A., Mehmood, Q., e Zainab, S.S., Hogan, A.: SPORTAL: profiling the content of public SPARQL endpoints. Int. J. Semant. Web Inf. Syst. **12**(3), 134–163 (2016)
10. Heule, S., Nunkesser, M., Hall, A.: HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. In: Proceedings of the 16th International Conference on Extending Database Technology, pp. 683–692 (2013)
11. Horvitz, D.G., Thompson, D.J.: A generalization of sampling without replacement from a finite universe. J. Am. Stat. Assoc. **47**(260), 663–685 (1952)
12. Kaminski, M., Kostylev, E.V., Grau, B.C.: Query nesting, assignment, and aggregation in SPARQL 1.1. ACM Trans. Database Syst. **42**(3), 1–46 (2017)
13. , Li, F., Wu, B., Yi, K., Zhao, Z.: Wander Join: online aggregation via random walks. In: Proceedings of the 2016 International Conference on Management of Data, SIGMOD 2016, pp. 615–629. ACM, New York (2016)
14. Li, K., Li, G.: Approximate query processing: what is new and where to go? Data Sci. Eng. **3**(4), 379–397 (2018)
15. Maillot, P., Corby, O., Faron, C., Gandon, F., Michel, F.: IndeGx: a model and a framework for indexing RDF knowledge graphs with SPARQL-based test suits. J. Web Semant. **76**, 100775 (2023)

16. Park, Y., Ko, S., Bhowmick, S.S., Kim, K., Hong, K., Han, W.: G-CARE: a framework for performance benchmarking of cardinality estimation techniques for subgraph matching. In: International Conference on Management of Data, SIGMOD Conference 2020, pp. 1099–1114. ACM (2020)
17. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 1–45 (2009)
18. Pham, T.H.T., Skaf-Molli, H., Molli, P., Nédelec, B.: Online sampling of summaries from public SPARQL endpoints. In: Companion Proceedings of the ACM on Web Conference 2024, WWW 2024, pp. 617–620, ACM, New York (2024)
19. Saleem, M., Hasnain, A., Ngomo, A.C.N.: LargeRDFBench: a billion triples benchmark for SPARQL endpoint federation. J. Web Semant. **48**, 85–125 (2018)
20. Särndal, C.E., Swensson, B., Wretman, J.: Model Assisted Survey Sampling. Springer, New York (2003)
21. Soulet, A., Suchanek, F.M.: Anytime large-scale analytics of linked open data. In: Ghidini, C., et al. (eds.) ISWC 2019. LNCS, vol. 11778, pp. 576–592. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30793-6_33
22. Wu, R., et al.: Learning to be a statistician: learned estimator for number of distinct values. Proc. VLDB Endow. **15**(2), 272–284 (2021)
23. Zhao, Z., Christensen, R., Li, F., Hu, X., Yi, K.: Random sampling over joins revisited. In: Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, 10–15 June 2018, pp. 1525–1539. ACM (2018)