ACERPI-Block: Applying Blocking Techniques to the ACERPI Approach

Christian Schmitz[1], Jonathan Martins[1], Serigne K. Mbaye[2], Edimar Manica[2], Renata Galante[1]

[1] Universidade Federal do Rio Grande do Sul, Brazil
{cschmitz, jmartins, galante}@inf.ufrgs.br
[2] Instituto Federal do Rio Grande do Sul, Brazil
{serigne, edimar.manica}@ibiruba.ifrs.edu.br

Abstract. Ordinances are documents issued by federal institutions that contain, among others, information regarding their staff. These documents are accessible through public repositories that usually do not allow any filter or advanced search on documents' contents. This paper extends ACERPI (an **approach** to **collect** documents, **extract** information and **resolve** entities from **institutional ordinances**), which identifies the people mentioned in ordinances from institutions to help users find the documents of interest. ACERPI-Block focuses on the Entity Resolution step of the approach, developing blocking strategies that allow scalability to hundreds of thousands of records being resolved. Experiments show a reduction of 93.3% in the number of comparisons of similarity between records if compared to the solution without blocking, with no decrease in efficacy.

Categories and Subject Descriptors: H.2 [**Database Management**]: Miscellaneous; H.3 [**Information Storage and Retrieval**]: Miscellaneous; I.7 [**Document and Text Processing**]: Miscellaneous

Keywords: Entity Resolution, Blocking

1. INTRODUCTION

Brazilian federal institutions publish documents named Ordinances to disseminate changes in their employees' positions, such as function substitutions, requests for leave, retirement, and vacation. Ordinances are official documents issued by organs of the institutions that implement the resolutions contained therein. Today, due to Law no. 12.527 [Brasil 2011], which formalizes the disclosure of information that may be of public interest produced by federal institutions, the publication of the Ordinances by the institutions occurs publicly, allowing anyone to consult them. Access to this information is often given by making the PDF files of the documents available in individual repositories of each institution or even of different campuses within the institutions. However, with little or no filtering for advanced searches on their content, these document repositories do not allow fast (or even feasible) search for specific employees or types of documents.

Web Scraping techniques have been used to discover and extract files from repositories. This allows fast overcoming of common scraping problems, such as server-side requests' restrictions. Named Entity Recognition (NER) is applied to identify names in the documents' texts. For this, transfer learning is used to re-train a neural network to the Ordinances domain. Entity Resolution (ER) techniques are then used for matching identified names to real-world people, experimenting with different matching criteria. ACERPI (**abordagem** para a **coleta** de documentos, **extração** de informação e **resolução** de entidades em **portarias institucionais**; a translation for an **approach** to **collect** documents, **extract** information and **resolve** entities from **institutional ordinances**) [Schmitz et al. 2021] allows the user, then, to search a database using information extracted from the documents, such as the employees mentioned, their identification numbers, the publication date of the Ordinances, and the Ordinances' identification numbers.

This paper presents ACERPI-Block, an extension of ACERPI [Schmitz et al. 2021] which focuses on experimenting and evaluating Entity Resolution strategies, such as a different matching function and blocking techniques, in order to improve the ER step of the original approach. Experiments with actual data from the Federal University of Rio Grande do Sul demonstrate the solutions' efficacy, with a reduction of 93.3% of record comparisons.

Related approaches include Orion [Manica et al. 2017], which identifies entity-pages for data acquisition; [Dozier et al. 2010], where Dozier et al. apply different NER techniques, as well as ER in a set of legal documents from the United States of America; and ACERPI [Schmitz et al. 2021], the original approach from which ACERPI-Block started from. ACERPI-Block provides a flexible approach for Brazilian ordinances' collection, information extraction, and entity resolution while leveraging blocking techniques in the last step and providing structured data for analysis as a result.

This paper is structured as follows. Section 2 reviews related work. Our proposed ACERPI approach is described in Section 3. Section 4 discusses the experimental results, while Section 5 concludes the paper.

2. RELATED WORK

Four approaches are related to ACERPI-Block, proposed in this paper. The original ACERPI and other three gathered through empirical research. The first one is the Orion approach [Manica et al. 2017], which aims to discover and extract real entities and attribute values from entity pages. An entity page is a web page that publishes data describing an entity of a given type [Blanco et al. 2008]. Unlike the ACERPI-Block approach, where Natural Language Processing (NLP) is used to identify names in unstructured text, the Orion approach leverages the structure of the discovered entity pages DOM trees [World Wide Web Consortium].

In [van Dalen-Oskam et al. 2014], the authors adapted an available NER software to create a Named Entity tagger for Dutch fiction. They also applied Entity Resolution techniques to link the identified Named Entities to Wikipedia entries. They generated a Web Application that provides free-text searching, searching and metadata filtering, and visualization of search results. In ACERPI-Block, queries are available only via database clients, and the creation of a GUI is planned for future work.

Dozier et al. [Dozier et al. 2010] described NER methods using lookup techniques, context rules, and statistical models. They also described techniques employed in resolving entities, such as blocking, features for matching functions, and supervised and semi-supervised learning for the matching function. Furthermore, some techniques were used to extract and resolve entities in legal documents from the United States of America, such as jurisprudence cases, depositions, defenses, and other trial documents. The ER technique employed, as opposed to the one used in ACERPI-Block, aims at the association of each entity found to an entry in an authority file. In ACERPI-Block, ER occurs by grouping entities with similar names and contexts.

The initial approach, ACERPI [Schmitz et al. 2021], proposed a flexible pipeline to discover, obtain, convert and structure files, extract information, and solve entities from institutional Ordinances. This was achieved through the composition of more straightforward techniques in data collection, information extraction, and entity resolution. ACERPI-Block leverages the plug-in capabilities of ACERPI to explore the entity resolution step deeply and exclusively, leveraging different blocking techniques and a second matching function.

3. ACERPI-BLOCK

This section describes ACERPI-Block, an approach that, by using techniques for file discovery, retrieval, conversion, structuring, information extraction, and ER, generates a database of records and entities which allows searching professional information of public institutions staff in a categorized, filtered, and clustered manner.

Figure 1 illustrates the data flow from its source to storage and post-processing. ACERPI-Block takes as input a set of documents' repositories. As output, a database is generated with structured information of the mentioned employees, details of the Ordinances, and their metadata. The collection step[1] includes discovering and retrieving the files, converting them to text, identifying the Ordinances published in the given document, and structuring into XML files. The information extraction step uses Named Entity Recognition [Nadeau and Sekine 2007] and Transfer Learning techniques to identify references to an employee and the related metadata and store them in a standard format. Finally, Entity Resolution techniques [Christophides et al. 2020] are used to relate the identified references to the corresponding real-world personnel and generate the final database. The final database, non-relational and document-oriented (MongoDB), can be used to obtain information about an employee, the ordinances that mention them, and the metadata extracted.

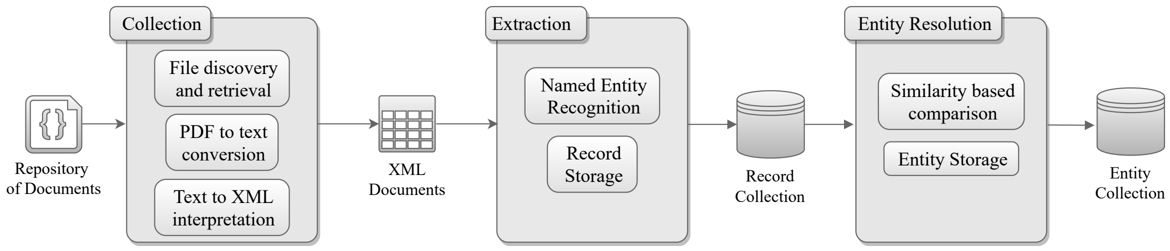


Fig. 1. Data flow overview in ACERPI-Block

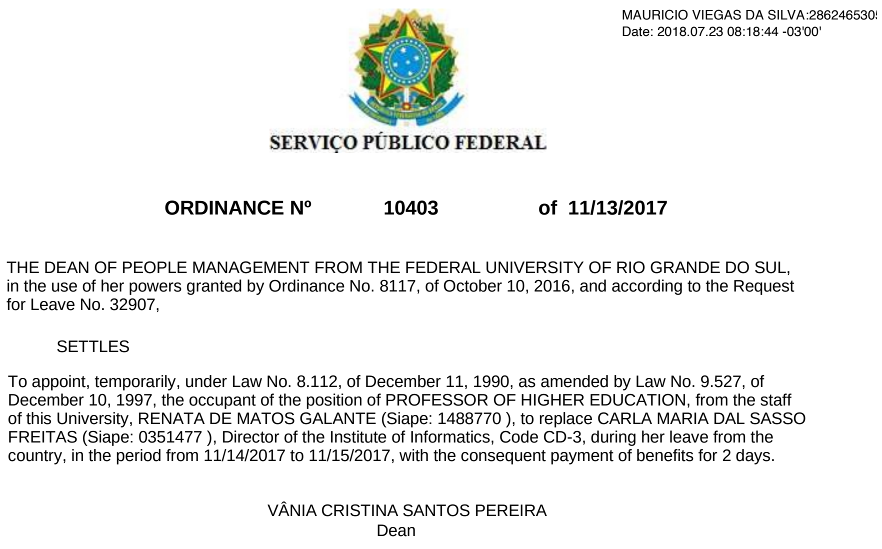
One example is the UFRGS repository[2] that contains, among other documents, Ordinance 10403 from 11/13/2017, illustrated in Figure 2. This Ordinance indicates a temporary employee replacement and refers to the employees Renata de Matos Galante and Carla Maria dal Sasso Freitas.

3.1 Collection

This section presents the strategy to discover, retrieve, convert, and parse the documents into an intermediate, structured format (XML). The initial data, PDF files of the Ordinances, are downloaded from the repositories of the Institutions. The method for **File Discovery and Retrieval** is based on Web Scraping techniques. However, one or more techniques may be used according to the repository structure and restrictions. Here, we adopted the inference of a navigation pattern [Lage et al. 2004] that, through a regular expression, generalizes the relevant URLs of the repository for automating the retrieval at a later stage.

After discovering and retrieving the PDF files and before starting the extraction step, the **Structuring** sub-step occurs. The structuring goal is to transform the data from its original format (PDF) to an intermediate format with the content of the individual ordinances. Structuring is achieved in two phases: conversion from PDF to text files and interpretation of the content of the files to one or multiple Ordinances.

[1]Developed in partnership with Serigne K. Mbaye and published in his Bachelor Thesis "Developing and Evaluating an Ordinances' Retrieval tool".
[2]Available at `https://www1.ufrgs.br/sistemas/sde/gerencia-documentos/index.php/publico/consultar/`. Last access in 06/26/2022.



MAURICIO VIEGAS DA SILVA:286246530
Date: 2018.07.23 08:18:44 -03'00'

SERVIÇO PÚBLICO FEDERAL

ORDINANCE Nº 10403 of 11/13/2017

THE DEAN OF PEOPLE MANAGEMENT FROM THE FEDERAL UNIVERSITY OF RIO GRANDE DO SUL, in the use of her powers granted by Ordinance No. 8117, of October 10, 2016, and according to the Request for Leave No. 32907,

SETTLES

To appoint, temporarily, under Law No. 8.112, of December 11, 1990, as amended by Law No. 9.527, of December 10, 1997, the occupant of the position of PROFESSOR OF HIGHER EDUCATION, from the staff of this University, RENATA DE MATOS GALANTE (Siape: 1488770), to replace CARLA MARIA DAL SASSO FREITAS (Siape: 0351477), Director of the Institute of Informatics, Code CD-3, during her leave from the country, in the period from 11/14/2017 to 11/15/2017, with the consequent payment of benefits for 2 days.

VÂNIA CRISTINA SANTOS PEREIRA
Dean

Fig. 2. Ordinance number 10403 of 11/13/2017, issued by the Central Administration of the Federal University of Rio Grande do Sul

First, the conversion of the PDF file to text format is performed. This is achieved by using Apache PDFBox [3], a PDF file manipulation library. Then, in the second phase, the text files' interpretation is performed, extracting the multiple Ordinances that may be contained in each file and their metadata into the intermediate format.

The intermediate format, in XML, has a root element called "document", which has as attributes the unique identifier of the document, the name of the original PDF file, and the file's location in the Institution's repository. A document can have an arbitrary number of children, called "ordinance". The Ordinance corresponds to each Ordinance that the original PDF document has, holding its number and date as attributes. The pure textual contents referring only and exclusively to the Ordinance identified by the number and date previously extracted are stored within the ordinances elements. Listing 1 shows an example of a structured XML file. The data extraction is performed through regular expressions that capture the character ranges that make up the patterns describing the format of an ordinance, its number, and publication date, respectively.

Listing 1. Intermediate structure parsed from the document displayed in Figure 2

```xml
<document id="47048" filename="47048.pdf" location="https://www1.ufrgs.br/sistemas/sde/
    gerencia-documentos/index.php/publico/ExibirPDF?documento=47048">
    <ordinance nr="10403" date="11/13/2017">
        ORDINANCE No 10403 of  11/13/2017
        THE DEAN OF PEOPLE MANAGEMENT FROM THE FEDERAL UNIVERSITY [...]
        SETTLES
        To appoint, temporarily, [...] the occupant of the position of PROFESSOR OF
            HIGHER EDUCATION, from the staff
        of this University, RENATA DE MATOS GALANTE (Siape: 1488770 ), to replace CARLA
            MARIA DAL SASSO
        FREITAS (Siape: 0351477 ), Director of the Institute of Informatics [...]
        VANIA CRISTINA SANTOS PEREIRA
        Dean
    </ordinance>
</document>
```

[3] Available at `https://pdfbox.apache.org/`. Last access in 06/26/2022.

3.2 Extraction

This section discusses the techniques used to recognize the names of the employees in the Ordinances and the structure chosen to store this relationship.

The first sub-step consists of extracting from the content of the Ordinances the names of the employees mentioned using **Named Entity Recognition**. For this, the natural language processing library Spacy [Explosion.ai 2021a], and a pre-trained model adapted for the Ordinances domain are used. The starting model is `pt_core_news_sm`[4], trained from a news database in Portuguese and convolutional neural networks. With the `pt_core_news_sm` model as a base, data annotation of the Institutions' Ordinances is performed using the Prodigy tool [Explosion.ai 2021b], which provides a Web Application that enables annotation of Named Entities using suggestions provided by the base model. After data annotation, it is necessary to retrain the base model to better interpret the files from the Ordinances domain. This stage is fundamental for improving the recognition of named entities in data with patterns previously unknown by the generic model. This is done again through Prodigy. Given the final model and the ordinance content, the NER output will be the identified Named Entities.

In addition to the name of the employees, context information is extracted from the Ordinances. The SIAPE registration numbers (when present) are extracted via regular expressions. This data corresponds to a unique identification number of the employee, which is used in the ER stage. In Ordinances, these numbers usually accompany the name of the employee, mention the term SIAPE and can be extracted from regular expressions such as `[S|s][I|i][A|a][P|p][E|e][^0-9.]{1,3}([0-9]{6,8})`. If the employee's SIAPE registration number is not precisely identified in the 120 characters following the last character of the employee's name, a list is stored for the employee containing all the registration numbers found in the Ordinance under analysis. This alternative proves useful when names of public servants are mentioned in a list, followed by another list with the respective SIAPE identifiers.

For the PDF file in Figure 2, the output of the extraction stage includes the names of the employees, RENATA DE MATOS GALANTE, CARLA MARIA DAL SASSO FREITAS, and VANIA CRISTINA SANTOS PEREIRA. Their associated SIAPE number, 1488770 for Renata de Matos Galante, 0351477 for Carla Maria Dal Sasso Freitas and both values 1488770 and 0351477 for server Vânia Cristina Santos Pereira. Besides, the association of these data with ordinance 10403 of 13 November 2017 is performed.

Listing 2. Record created for Renata de Matos Galante

```
{
    "id": 131072,
    "name": "RENATA DE MATOS GALANTE",
    "siape": ["1488770"],
    "document":
        {"name": "47048"}
}
```

Listing 3. Record created for Carla Maria Dal Sasso Freitas

```
{
    "id": 131073,
    "name": "MARIA DAL SASSO FREITAS",
    "siape": ["0351477"],
    "document":
        {"name": "47048"}
}
```

After NER, **Record Storage** occurs. In ACERPI, a main document structure named record was defined, which concentrates the information of a person identified in an ordinance. This database document has, respectively: *(i)* a unique identifier of the record; *(ii)* the name of the individual server identified in the NER step; *(iii)* A list of SIAPEs identified in ordinances related to the employee. This list is populated when it is not possible to identify a specific number for the server, and all the

[4]Available at `https://spacy.io/models/pt\#pt\_core\_news\_sm`. Last access in 03/15/2021.

values identified in the document where the respective name was found are inserted into the list; *(iv)* the identifier of the ordinance from which this record was found. Three records are created when analyzing the document in Listing 1. The record with identifier 131072 indicated in Listing 2 of the employee Renata de Matos Galante, the record with identifier 131073 indicated in Listing 3 of the employee Carla Maria Dal Sasso Freitas and a third record in the same basis for the Dean Vânia Cristina Santos Pereira.

3.3 Entity Resolution

Given the records, entity resolution is performed. This step involves identifying which records refer to the same entities in the real world (i.e., which documents refer to the same staff member, reflecting ordinances in which he/she was directly involved). For example, the records from Listings 2 and 4 both refer to the Professor Renata de Matos Galante. At the end of the ER step, it is expected that the two records are grouped in the cluster that refers to the real-world entity Renata de Matos Galante, from the Institute of Informatics of the Federal University of Rio Grande do Sul.

ER in the ACERPI-Block approach is performed by grouping the records using **Similarity Based Comparison**. The ER algorithm receives as input the set of records identified from the Ordinances and tries to match each record to any existent group. If no matching occurs, a new group is created containing the record. This loop occurs until all records are grouped, and the output consists of the identified clusters.

The match function is the central part of the algorithm since it defines whether the new record is or is not part of a cluster (i.e., it measures how similar the new record and the records already belonging to a cluster are). It can be simple, as a direct comparison of the named entities of the records, or complex, using methods that compare substrings of the named entities and records' metadata. The ACERPI-Block approach uses a technique that also analyses the context for entity resolution, which in the case of ordinances occurs through the SIAPE identification number. When unique and identical, the SIAPE implies references to the same real-world entity. If the SIAPE numbers are not unique and identical, the comparison of the named entities is performed by cleaning the records' names and comparing them directly. The cleaning procedure consists of characters undercapitalization and trimming. Thus, the records in Listings 2 and 3 would not be grouped because although both have only one SIAPE registration number associated, they differ. On the other hand, the records on listings 2 and 4 would be grouped because they have only one SIAPE registration number each, and they are identical.

The clusters resulting from ER proceed to **Entity Storage**. An entity is generated for each cluster, representing a real-world entity. Each entity has a reference to the identifiers of the records that compose it and a set of names and SIAPE registration numbers found in the records to reduce the computational cost of ER. For the records of Listings 2 and 4, after the step of solving entities, the entity of Listing 5 is generated.

Listing 4. Another record created for the employee Renata de Matos Galante

```
{
    "id": 4630,
    "name": "RENATA DE MATOS GALANTE",
    "siape": ["1488770"],
    "document":
        {"name": "50216"}
}
```

Listing 5. Entity generated from entity resolution of the records from Listings 2 and 4

```
{
    "records": [47048, 50216],
    "names": ["RENATA DE MATOS GALANTE
        "],
    "siapes": ["1488770"]
}
```

4. EXPERIMENTAL EVALUATION

This section describes three experiments: (1) establishment of ground truth metrics for the current implementation, to be used for comparison in the other experiments; (2) evaluation of letter-based blocking techniques in references' names to improve entity resolution performance; and (3) evaluation of combined letter and size-based blocking techniques in references' names in order to improve entity resolution performance. It also highlights the main aspects of the evaluation algorithm developed for the experiments.

4.1 Data Source

DOCS-UFRGS, comprised of public documents from the Federal University of Rio Grande do Sul. This source consists of extracted documents, mostly Ordinances, from the University's repository. The data was collected until March 3, 2020, with a total storage of 7.99Gb and 44865 PDF files. From these files, 194 thousand records were generated as the output of the Extraction step from experiments developed in ACERPI [Schmitz et al. 2021], which is the starting point for the following experiments.

During the experiments, two data sets are mentioned. One is called **full data set** and the other is called **test data set**. The test data set consists of 613 records for which the real-world entities they refer to are known, i.e., the groupings that are to be generated are known. These are references to the Institute of Informatics' professors. The test data set is a subset of the full data set. It is important to note that the real-world entities are not known for all these records, only for the ones from the test data set. All efficacy metrics for the full data set are based on the hits and misses of the elements in the test subset groupings.

4.2 Environment

All experiments were developed in Google Cloud Platform machines n2-standard-4 model with 4 CPUs on Intel Cascade Lake architectures, 16 GB of memory, local SSD, and 10 Gbps internet bandwidth. The machines were configured with Linux Debian 10 and ran MongoDB Community Server, the extension for Python cProfile[5] for collecting data about function calls, and the Linux time package[6] for collecting runtime and CPU utilization data.

4.3 Methodology

The methodology applied to the experiments is the following:

(1) Run the entity resolution algorithm once with the test data set with the *cProfile* extension disabled. The purpose of this run is only to collect efficacy metrics on the application of the algorithm to a small data set.
(2) Run the entity resolution algorithm once on the complete data set with the *cProfile* extension enabled to collect function-level runtime data and the number of calls for each executed function.
(3) Run the entity resolution algorithm ten times on the full dataset, this time with the *cProfile* extension disabled to prevent the *overhead* of this extension from influencing the algorithm's runtime.

The execution with the *cProfile* extension enabled happens only once because the purpose of using this extension is to collect the number of comparisons performed in each experiment and analyze the points in the code that use most of the algorithm's execution time. Since the data set used is the

[5]Available at `https://docs.python.org/3/library/profile.html`. Last access in 02/12/2022.
[6]Available at `https://man7.org/linux/man-pages/man1/time.1.html`. Last access in 02/12/2022.

same in every run, and the algorithm is deterministic, the program always performs the same number of comparisons and therefore does not need to be run multiple times with the extension enabled.

The analysis of the execution time and CPU utilization of each version of the algorithm is based on the metrics collected by the Linux *time* utility in all ten runs with the complete data set and with the *cProfile* extension disabled.

4.4 Metrics

The metrics chosen to evaluate Entity Resolution are based on the pairwise comparison, where each pair indicates a relationship between entities (to refer to the same entity in the real world). The metrics used for this experiment were: *(i)* precision, the percentage of identified pairs that actually refer to the same entity in the real world; *(ii)* recall, the percentage of all pairs referring to the same entity that were correctly identified; and *(iii)* F1-Score. In addition, the variables were defined as *(i)* true positives, the pairs that actually refer to the same entity in the real world; *(ii)* false positives, the pairs that were identified as references to the same entity, but are not; and *(iii)* false negatives, the pairs that were not identified as references to the same entity, but are.

Algorithm 1 is used for evaluating the results of the entity resolution algorithm. Unlike the evaluation algorithm used in ACERPI [Schmitz et al. 2021], Algorithm 1 uses sets, and the binomial coefficient formula to calculate correct and incorrect pairs efficiently and with no need to generate all pairs for the gold set and the entity being evaluated.

Algorithm 1: Algorithm for Entity Resolution efficacy evaluation

```
Input  : gold standard, existing records in gold standard, groupings
Output: true positives, false positives, false negatives
1  gold_std ← gold standard;
2  gold_records ← existing records in gold standard;
3  entities ← groupings;
4  true_positives ← 0;
5  false_positives ← 0;
6  false_negatives ← 0;
7  for each entity e in entities do
8      assessable_entity ← e ∩ gold_std;
9      for each entity gold_entity em gold_std do
10         true_members ← gold_records ∩ assessable_entity;
11         assessable_entity ← assessable_entity - true_members;
12         true_positives ← true_positives + binomial_coefficient(size(true_members), 2);
13         false_positives ← false_positives + size(true_members) · size(assessable_entity);
14     end
15 end
16 possible_true_positives ← 0;
17 for each entity gold_entity in gold_std do
18     possible_true_positives ← possible_true_positives + binomial_coefficient(size(gold_entity), 2);
19 end
20 false_negatives ← possible_true_positives - true_positives;
21 return true_positives, false_positives, false_negatives;
```

4.5 Experiment 1 - Entity Resolution without blocking

To evaluate the effectiveness of the blocking techniques, it was first necessary to evaluate the entity resolution algorithm without applying any blocking techniques. In this experiment, the ER algorithm's performance using two techniques for string comparison is analyzed: exact match and Jaro-Winkler. The threshold used for Jaro-Winkler was 0.9, which has been previously experimented with and defined as optimal in ACERPI-Link [Eich 2021], although the data sets used not being identical.

4.5.1 *Results.* Table I shows the runtime metrics, in seconds, of the two versions of the algorithm. It demonstrates that the Jaro-Winkler *strings* comparison algorithm harms execution time, which was expected. Despite having all other aspects of the algorithm in common, using Jaro-Winkler causes the program to take, on average, 5.88 times longer to execute. Table II shows the most relevant metrics collected by the *cProfile* extension, number of name comparisons and the total time each version spent performing these comparisons.

Table I. Runtime metrics - Entity Resolution without blocking

Runtime Metric	Exact Match	Jaro-Winkler
Average	7585.8s	44571.1s
Maximum	7838.0s	46720.0s
Minimum	7386.0s	43402.0s
Standard Deviation	164.5s	1154.1s

Table II. Number of Comparisons - Entity Resolution without blocking

Metric	Exact Match	Jaro-Winkler
Comparisons	1,259,040,135	1,439,499,577
Time spent in comparison operations	3553s	60720s
Total time	12866s	69348s

Table III shows the CPU usage related metrics of the ER algorithm's process using both *strings* comparison algorithms. The ER program uses only one *thread* and does not implement any parallelism. This means that every time it queries the database, it waits for the response before resuming execution, causing several idle moments to occur.

Table III. CPU Utilization - Entity Resolution without blocking

CPU Usage Metric	Exact Match	Jaro-Winkler
Average	79.5%	96.5%
Maximum	80.0%	97.0%
Minimum	79.0%	95.0%
Standard Deviation	0.5%	0.7%

Tables IV and V describe the efficacy metrics for each version of the algorithm and for the full and test datasets. Table IV shows the results when using the Jaro-Winkler algorithm, whose assumption is a tradeoff between time and efficacy. The efficacy of the result generated by the algorithm using Jaro-Winkler for name comparison is better than using an exact match. This is because the Jaro-Winkler algorithm can group elements even when they have small differences in their names, which an exact match cannot. For the test data set, this increases the number of True Positives and decreases the number of False Negatives without increasing the number of False Positives.

The results in Table IV show the strengths of using the Jaro-Winkler algorithm for entity resolution. Nevertheless, these results were generated from a data set that does not contain many references and therefore had less opportunity for incorrect groupings. Therefore, for better analysis and a more

Table IV. Efficacy Metrics - Entity Resolution without blocking - Test Data Set

Metric	**Exact Match**	**Jaro-Winkler**
Precision	100.00%	100.00%
Recall	82.38%	94.33%
F1	90.34%	97.08%

realistic scenario for evaluation, the algorithm was also run for the complete data set, and its respective efficacy was analyzed. Table V describes the efficacy of the results for each algorithm using the full data set.

Table V. Efficacy Metrics - Entity Resolution without blocking - Full Data Set

Metric	**Exact Match**	**Jaro-Winkler**
Precision	100.00%	99.82%
Recall	82.38%	64.53%
F1	90.34%	78.39%

In Table V, a result that is considered unexpected can be seen. The efficacy of the exact match is higher than the one from the match using Jaro-Winkler on all metrics. What occurred here is that the same aspect that made Jaro-Winkler produce excellent results in a small set is responsible for a drop in efficacy when a larger data set is used. By looking at the algorithm's results using an exact comparison of *strings* in both cases, it can be seen that the exact comparison remained immune to the increase in the number of references due to the rigidity of its comparison.

4.6 Experiment 2 - Applying letter-based blocking techniques

In this experiment, blocking techniques are applied to the algorithms used in Experiment 1 and analyze the effect these techniques have on runtime, number of comparisons, CPU utilization, and efficacy of the result.

Two different blocking techniques are applied:

—**Blocking by name initials:** when querying the database for entities to compare against a given element. Only entities whose first name in the name list begins with any of the element's name initials are considered, regardless of whether it is upper or lower case.

—**Blocking by the first letter of name** when querying the database for entities to compare against a given element. Only entities whose first name in the name list begins with the same letter, regardless of whether it is upper or lower case, as the element name are considered.

4.6.1 *Results.* Table VI presents the results of the runtime metric for the entity resolution algorithm using an exact match of strings and Jaro-Winkler. This metric is collected by the *time* application. It is based on the runtime of ten executions of the program, with the *cProfile* extension disabled and with the full dataset. From these runtime metrics, it is possible to see a reduction in runtime by using blocking to filter the comparison entities. It can also be seen that although both versions of the algorithm had a reduction in their average execution time, the one using Jaro-Winkler for string comparison had a greater reduction in execution time than the version using exact string matching. The version using Jaro-Winkler had a reduction of **79.2%** between the non-blocking version and the version with blocking based on the first letter of the name, and of **90.3%** between the non-blocking version and the version with blocking based on the initials of the name. In contrast, these reductions were **66.8%** and **75.2%** in the versions using exact match.

This difference does not come from the number of comparisons, as it can be seen in Table VII that the number of comparisons has been reduced similarly between the two versions. The number

Table VI. Runtime Metrics - Letter-based blocking - Matching Function Variations

Runtime Metric	**No Blocking**	**Initials**	**First Letter**
	Exact Match		
Average	7585.8s	2520.0s	1882.0s
Maximum	7838.0s	2563.0s	1976.0s
Minimum	7386.0s	2458.0s	1742.0s
Standard Deviation	164.5s	32.0s	86.0s
	Jaro-Winkler		
Average	44571.1s	9265.0s	4330.8s
Maximum	46720.0s	10190.0s	4353.0s
Minimum	43402.0s	8935.0s	4300.0s
Standard Deviation	1154.1s	430.8s	20.8s

of comparisons in the version using the Jaro-Winkler algorithm was reduced by **81.2%** with the application of initials-based blocking and **93.3%** with the application of first-letter-based blocking, while the version using exact comparison of *strings* had a reduction of **82%** and **93.3%**. The number of comparisons was reduced almost identically in both versions, so this was not the factor leading to the difference in the reduction in average execution time.

Table VII. Comparisons' Metrics - Letter-based Blocking - Full Data Set - Matching Function Variations

Metric	**No Blocking**	**Initials**	**First Letter**
	Exact Match		
Comparisons	1,259,040,135	226,749,153	84,487,924
Comparisons' Runtime	3553s	665s	248s
Total Runtime	12866s	3450s	2425s
	Jaro-Winkler		
Comparisons	1,439,499,577	261,655,437	96,432,691
Comparisons' Runtime	60720s	11553s	4077s
Total Runtime	69348s	14474s	6113s

Table VII also pinpoint metrics that help understand the cause of the disparity in the average time reductions for each version, such as execution time spent on comparisons and total execution time. For example, without blocking, the Jaro-Winkler version of the entity resolution algorithm spends about **87.5%** of its runtime performing comparisons between references and entities. In contrast, the version that uses an exact match of strings spends only **27.6%** of its runtime on making these comparisons. This means that, assuming all other program instructions remain the same, halving the number of comparisons for the Jaro-Winkler version would reduce the total execution time by **43.75%**, while the same reduction in the number of comparisons would reduce the total execution time for the version that uses exact match by only **13.8%**.

This difference is a consequence of the high cost of the string comparison operation using the Jaro-Winkler algorithm. The execution time of the database queries is almost constant because nearly identical queries are being made for both algorithms. This means that the increase in comparison time caused by using the Jaro-Winkler algorithm causes a change in the distribution of the work generated by this program between the program itself and the database.

Using the data in Table VIII it is possible to see another consequence of the change in workload distribution. The program's CPU utilization represents the percentage of its execution time spent running the program and not waiting for responses from other processes. The higher the CPU utilization, the higher the percentage of time spent running the program. It can be seen in Table VIII that the version of the program that uses Jaro-Winkler spends a much higher percentage of its total execution time executing the algorithm's operations than the version that uses exact string comparison.

Table VIII. CPU Usage - Letter-based Blocking - Full Data Set - Matching Function Variations

CPU Usage	**No Blocking**	**Initials**	**First Letter**
Exact Match			
Average	79.5%	54.2%	37.9%
Maximum	80.0%	55.0%	39.0%
Minimum	79.0%	54.0%	37.0%
Standard Deviation	0.5%	0.4%	0.9%
Jaro-Winkler			
Average	96.5%	90.1%	75.0%
Maximum	97.0%	87.0%	75.0%
Minimum	95.0%	54.0%	75.0%
Standard Deviation	0.7%	1.4%	0.0%

It is also necessary to take into account the efficacy metrics, described in Table IX. Increasing the efficiency of the entity resolution algorithm would be of little value if it required reducing its efficacy to do so. The first two scenarios demonstrate a positive environment for the applied blocking techniques. For both versions of the code, there was no change in efficacy when applied to the test data set. This could lead to the conclusion that these techniques can be considered filtering techniques since there is no drop in efficacy at all, which would indicate that all the comparisons that were not done would not generate a match anyway. Nevertheless, it is important to remember that the test data set has few instances, and it is possible to have missed instances of clustering by applying any of the techniques explored on a larger data set. For the version of the program that uses exact string matching, there was no change in the efficacy of the result. A more in-depth and detailed analysis of the effects of the blocking techniques described in this paper would require a more comprehensive test dataset.

Table IX. Efficacy Metrics - Letter-based Blocking - Matching Function and Data Set Variations

Metric	**No Blocking**	**Initials**	**First Letter**
Exact Match - Test Data Set			
Precision	100.00%	100.00%	100.00%
Recall	82.38%	82.38%	82.38%
F1	90.34%	90.34%	90.34%
Jaro-Winkler - Test Data Set			
Precision	100.00%	100.00%	100.00%
Recall	94.33%	94.33%	94.33%
F1	97.08%	97.08%	97.08%
Exact Match - Full Data Set			
Precision	100.00%	100.00%	100.00%
Recall	82.38%	82.38%	82.38%
F1	90.34%	90.34%	90.34%
Jaro-Winkler - Full Data Set			
Precision	99.82%	100.00%	99.87%
Recall	64.53%	66.84%	66.81%
F1	78.39%	80.13%	80.06%

The last scenario from Table IX holds the most exciting results on the effect of applying blocking techniques. In general, in an entity resolution algorithm, it is expected that the addition of blocking techniques will harm the efficacy of the result. In the case of the algorithm presented here, the opposite is observed. The implementation of the blocking techniques has a positive effect on all the efficacy metrics. This is due to the comparison using Jaro-Winkler, which uses very permissive parameters to group the references. This means that this version of the code is very susceptible to being misled by noise. Moreover, when blocking is applied, a noise filter is also created that helps the algorithm avoid false clustering, improving the efficacy of the result. It is also important to remember that defining the minimum distance between two names to be clustered was done before having an evaluation algorithm capable of evaluating the result of applying the algorithms to the complete data set.

4.7 Experiment 3 - Applying letter-based and length-based blocking techniques

In this experiment, the blocking techniques based on the name size of the reference being analyzed are explored, combined with the blocking techniques explored in Section 4.6. For this experiment, the following blocking techniques were used:

—**Unbounded** represents the results when no blocking by reference name size is applied.

—**33% to 300%** represents the results of applying blocking in which only entities whose first name in the name list has size **greater than 33% and less than 300%** of the reference name size.

—**50% to 200%** represents the results of applying a blocking where only the entities whose first name in the list of names has size **greater than 50% and less than 200%** of the size of the reference name.

—**66% to 150%** represents the results of applying blocking where only the entities whose first name in the list of names has size **more than 66% and less than 150%** of the size of the reference name.

4.8 Results

One can see a positive impact by analyzing the program execution times before and after the application of blocking by the size of the reference name in Table X. One can also see in Table XI that the behavior is consistent with what is expected. The more restrictive the blocking condition is, the greater the reduction in average time. The same effect can also be seen in Section 4.6.1, where the algorithm has a greater reduction in its runtime when it uses the Jaro-Winkler algorithm to perform the name comparison.

Table X. Runtime Metrics - Full Data Set - Matching Function and Blocking Variations

Runtime Metric	**Unbounded**	**33% to 300%**	**50% to 200%**	**66% to 150%**
Exact Match - First Letter				
Average	1882.3s	1749.8s	1619.5s	1562.3s
Maximum	1976.4s	1764.0s	1629.2s	1571.7s
Minimum	1742.0s	1740.3s	1609.5s	1555.0s
Standard Deviation	86.0s	7.0s	6.3s	4.7s
Exact Match - Initials				
Average	2519.6s	2431.0s	2188.5s	1916.7s
Maximum	2562.9s	2565.3s	2202.3s	1925.6s
Minimum	2458.4s	2325.0s	2178.0s	1910.2s
Standard Deviation	32.1s	101.4s	7.7s	5.9s
Jaro-Winker - First Letter				
Average	4330.8s	3808.5s	3473.4s	2705.8s
Maximum	4353.0s	3859.0s	3563.1s	2753.5s
Minimum	4300.0s	3752.0s	3416.0s	2693.5s
Standard Deviation	20.8s	31.6s	50.6s	17.9s
Jaro-Winkler - Initials				
Average	9264.6s	8929.4s	7948.3s	6333.6s
Maximum	10190.0s	9148.0s	8045.0s	6523.0s
Minimum	8935.0s	8732.0s	7797.0s	6263.0s
Standard Deviation	430.9s	149.6s	78.7s	74.5s

Table XIII, based on the results shown in Table XII, supports the finding that the biggest factor for the difference in the impact of applying blocking by reference name size on total execution time is the percentage of execution time that is spent performing comparisons. This happens because the comparison reduction is relatively similar across all algorithm versions.

Table XI. Percentage reduction in average runtime

Version	33% to 300%	50% to 200%	66% to 150%
Exact Match - First Letter	7.04%	13.96%	17.00%
Exact Match - Initials	3.52%	13.14%	23.92%
Jaro-Winkler - First-Letter	12.06%	19.80%	37.52%
Jaro-Winkler - Initials	3.62%	14.21%	31.63%

Table XII. Number of Comparisons - Full Data Set - Matching Function and Blocking Variations

Metric	Unbounded	33% to 300%	50% to 200%	66% to 150%
Exact Match - First-letter				
Comparisons	84,487,924	78,058,713	64,007,610	43,958,753
Comparison's Runtime	248.0s	227.5s	185.1s	127.4s
Total Time	2424.8s	2153.7s	1953.6s	1823.4s
Exact Match - Initials				
Comparisons	226,749,153	210,091,823	179,150,783	130,962,611
Comparisons' Runtime	664.6s	615.4s	517.8s	377.6s
Total Time	3449.7s	3445.4s	2987.5s	2524.2s
Jaro-Winkler - First-letter				
Comparisons	96,432,691	90,408,057	72,704,794	48,783,963
Comparisons' Runtime	4077.3s	3785.1s	3174.7s	2187.8s
Total Time	6112.8s	5534.1s	4930.1s	3655.2s
Jaro-Winkler - Initials				
Comparisons	261,655,437	245,840,640	203,783,357	146,326,897
Comparisons' Runtime	11553.2s	10751.7s	9313.9s	7007.6s
Total Time	14474.1s	13229.4s	11992.9s	9171.9s

Table XIII. Percentage reduction in number of comparisons

Version	33% to 300%	50% to 200%	66% to 150%
Exact Match - First-letter	7.61%	24.24%	47.97%
Exact Match - Initials	7.35%	20.99%	42.24%
Jaro-Winkler - First-letter	6.25%	24.61%	49.41%
Jaro-Winkler - Initials	6.04%	22.12%	44.08%

Table XIV shows that as the rules of blocking by reference name size get more restrictive, the database becomes a bottleneck for the program. It can be seen that from applying blocking by reference name size with bounds of 50% and 200% both versions of the algorithm that use exact comparison spend more time waiting for a response from the database than running the program. These versions have excellent potential for improving efficiency through parallel and asynchronous computing, as they are well below optimal CPU utilization. From Table XV it can be seen that there is a cost to applying blocking by the size of the reference name. The drop in efficacy is likely caused by discarding comparisons that would generate clusters. This result is worse in terms of efficacy than blocking based on the first letter of the entity's first name.

From Table XVI it can be observed that, as seen in Section 4.6.1, applying more restrictive blocking causes an increase in the F1 measure for the algorithms using Jaro-Winkler. This is due to the fact that the calibration of the distance thresholds used in the Jaro-Winkler algorithm that generates a cluster is too lax, and the blocking techniques act as noise reduction, which prevents many of the erroneous clusters that the algorithm would generate. The result for the algorithm that uses an exact string match is the same as seen for the test data set. This is because the algorithm using exact string comparison is much more restrictive and, therefore, much more resistant to noise.

Two failure scenarios were observed during the experiment. The first one is where similar names exist for different entities, and ACERPI-Block cannot differ. This happens, for example, for *Felipe Caron* and *Felipe Gaskin Cardon*. The mismatch happens due to the permissive parameter for the

Table XIV. CPU Usage - Full Data Set - Matching Function and Blocking Variations

CPU Usage	Unbounded	33% to 300%	50% to 200%	66% to 150%
Exact Match - First-letter				
Average	37.9%	35.0%	33.0%	28.0%
Maximum	39.0%	35.0%	33.0%	28.0%
Minimum	37.0%	35.0%	33.0%	28.0%
Standard Deviation	0.9%	0.0%	0.0%	0%
Exact Match - Initials				
Average	54.2%	51.4%	49.1%	44.0%
Maximum	55.0%	52.0%	50.0%	44.0%
Minimum	54.0%	50.0%	49.0%	44.0%
Standard Deviation	0.4%	0.7%	0.3%	0.0%
Jaro-Winkler - First-letter				
Average	75.0%	74.3%	69.9%	65.1%
Maximum	75.0%	75.0%	70.0%	66.0%
Minimum	75.0%	74.0%	69.0%	65.0%
Standard Deviation	0.0%	0.5%	0.3%	0.3%
Jaro-Winkler - Initials				
Average	90.0%	87.6%	85.5%	82.1%
Maximum	91.0%	88.0%	86.0%	83.0%
Minimum	87.0%	87.0%	85.0%	82.0%
Standard Deviation	1.4%	0.5%	0.5%	0.3%

Table XV. Efficacy Metrics - Test Data Set - Matching Function and Blocking Variations

Metric	Unbounded	33% to 300%	50% to 200%	66% to 150%
Exact Match - First-letter				
Precision	100.00%	100.00%	100.00%	100.00%
Recall	82.38%	82.37%	80.89%	80.89%
F1	90.34%	90.33%	89.44%	89.44%
Exact Match - Initials				
Precision	100.00%	100.00%	100.00%	100.00%
Recall	82.38%	82.37%	80.89%	80.89%
F1	90.34%	90.33%	89.44%	89.44%
Jaro-Winkler - First-letter				
Precision	100.00%	100.00%	100.00%	100.00%
Recall	94.33%	94.32%	92.31%	87.28%
F1	97.08%	97.08%	96.00%	93.21%
Jaro-Winkler - Initials				
Precision	100.00%	100.00%	100.00%	100.00%
Recall	94.33%	94.32%	92.31%	87.28%
F1	97.08%	97.08%	96.00%	93.21%

Jaro-Winkler algorithm. It could be solved by either making it more restrictive or using more data, such as the entities' department, in the ER step. The second failure case is related to mistakenly identified entity names during the NER step. For example, it is hard to appropriately match entities such as *NAIRA MARIA BALZARETTI RENATA JENISCH BARBOSA TANIRA*, as it refers to at least two different employees, *Naira Maria Balzaretti* and *Renata Jenisch Barbosa*.

5. CONCLUSION

In this paper, an extension for ACERPI [Schmitz et al. 2021], an approach for processing Ordinances from federal institutions, was presented, which leverages the initial work's flexibility to explore improvements in the entity resolution step. The effectiveness of our approach was proven through experiments on a real data source with over 40 thousand files and thousands of employees mentioned. Results demonstrated the efficacy of the approach, with an improvement of more than 93%

Table XVI. Efficacy Metrics - Full Data Set - Matching Function and Blocking Variations

Metric	**Unbounded**	**33% to 300%**	**50% to 200%**	**66% to 150%**
Exact Match - First-letter				
Precision	100.00%	100.00%	100.00%	100.00%
Recall	82.38%	82.37%	80.89%	80.89%
F1	90.34%	90.33%	89.44%	89.44%
Exact Match - Initials				
Precision	100.00%	100.00%	100.00%	100.00%
Recall	82.38%	82.37%	80.89%	80.89%
F1	90.34%	90.33%	89.44%	89.44%
Jaro-Winkler - First-letter				
Precision	99.87%	99.87%	100.00%	100.00%
Recall	66.81%	65.66%	68.26%	76.35%
F1	80.06%	79.23%	81.14%	86.59%
Jaro-Winkler - Initials				
Precision	100.00%	100.00%	100.00%	100.00%
Recall	66.84%	65.69%	68.26%	76.35%
F1	80.13%	79.29%	81.14%	86.59%

in comparison numbers in the entity resolution process. The main contribution of this paper is the experimentation and evaluation of two different blocking techniques, based on the letters included in the names of the records and their sizes, as well as the implementation of the Jaro-Winkler similarity measure as a matching function.

For future work, we intend *(i)* the exploration of parallel and distributed programming to enable faster processing of the ER step in ACERPI-Block; *(ii)* a graphic interface for advanced search regarding staff and Ordinances; *(iii)* the classification of the content of the Ordinances into known categories; *(iv)* and the experimentation of various thresholds for the Jaro-Winkler matching function.

REFERENCES

Blanco, L., Crescenzi, V., Merialdo, P., and Papotti, P. Supporting the automatic construction of entity aware search engines. In *Proc. of the 10th ACM Workshop on WIDM*. NY, USA, pp. 149–156, 2008.

Brasil. Lei nº 12.527/2011. *Diário Oficial da República*, 2011.

Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G., and Stefanidis, K. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.* 53 (6), Dec., 2020.

Dozier, C., Kondadadi, R., Light, M., Vachher, A., Veeramachaneni, S., and Wudali, R. *Named Entity Recognition and Resolution in Legal Text.* Springer-Verlag, 2010.

Eich, L. Acerpi-link: Uma alternativa de resolução de entidades em portarias institucionais utilizando linkagem de registros. Bachelor Thesis, UFRGS, 2021.

Explosion.ai. Industrial-strength natural language processing. `https://spacy.io/`, 2021a. Accessed: 2021-09-05.

Explosion.ai. Prodigy · radically efficient machine teaching. an annotation tool powered by active learning. `https://prodi.gy/`, 2021b. Accessed: 2021-09-05.

Lage, J. P., Silva, A. S., Golgher, P. B., and Laender, A. H. F. Automatic generation of agents for collecting hidden web pages for data extraction. *DKE* vol. 49, pp. 177–196, 2004.

Manica, E., Dorneles, C. F., and Galante, R. Orion: A cypher-based web data extractor. In *DEXA*. Springer, Cham, pp. 275–289, 2017.

Nadeau, D. and Sekine, S. A survey of named entity recognition and classification. *Lingvisticæ Investigationes* 30 (1): 3–26, 2007.

Schmitz, C., Mbaye, S., Manica, E., and Galante, R. Acerpi: An approach for ordinances collection, information extraction and entity resolution. In *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados*. SBC, Porto Alegre, RS, Brasil, pp. 97–108, 2021.

van Dalen-Oskam, K., de Does, J., Marx, M., Sijaranamual, I., Depuydt, K., Verheij, B., and Geirnaert, V. Named entity recognition and resolution for literary studies. *Computational Linguistics in the Netherlands Journal* vol. 4, pp. 121–136, Dec., 2014.

World Wide Web Consortium. What is the document object model? `https://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/introduction.html`. Accessed: 2022-07-02.