Partial Orders and Progressive Blocking: A Matching-based Framework for Large-scale Entity Resolution in Bibliographic Data

Inaugural-Dissertation

zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Tobias Backes
aus Köln

Köln, August 2022

aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Berichterstatter:

1. Prof. Dr. Stefan Dietze

2. Prof. Dr. Stefan Conrad

3. Prof. Dr.-Ing. Laura Dietz

Tag der mündlichen Prüfung: *14. Februar 2023*

Versicherung an Eides Statt

Ich versichere an Eides Statt, dass die Dissertation von mir selbständig und ohne unzulässige fremde Hilfe unter Beachtung der “Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Heinrich-Heine-Universität Düsseldorf” erstellt worden ist.

Köln, 1. August 2022 Tobias Backes

Ich erkläre hiermit, dass ich die Dissertation **nicht** bereits einer anderen Fakultät vorgelegt habe. Ich habe zum Zeitpunkt der Abgabe dieser Dissertation weder vorherige erfolglose, noch vorherige erfolgreiche Promotionsversuche unternommen.

Köln, 1. August 2022 Tobias Backes

Abstract

Entity resolution (*ER*) is the task of grouping entity mentions by the real-world object they refer to. It is central to ordering and aggregating knowledge in the growing amount of available structured, semi-structured and unstructured information. At its core, ER determines whether the similarity between two entity representations is sufficient to indicate equivalence. From a technical point of view, the most essential aspect is not how to compute this similarity, but how to discover the most similar pairs efficiently. As it is infeasible to compare all mention pairs, dedicated techniques must be used to structure or partition the search space. This is known as *similarity search*. A simple approach is to establish a prior grouping based on selected key features or combinations thereof. This is known as (hash-based) *blocking* and is limited in modelling intransitive matching relationships. Alternative heuristics like alphabetical order can be used to suggest mention pairs in the order of their approximated coreference likelihood. This is known as *progressive* resolution. Progressive methods have focused on alphabetically sorting string-based entity representations, which optimistically assumes that coreference likelihood can be approximated as a total order. In this thesis, we suggest to *partially order* entity representations instead, as the subset partial order is better suited to model the matching relationships between sets of features. A notion of neighborhood as has been previously defined for total orders (e.g. alphabetical sorting) or continuous spaces (e.g. space partitioning) can also be defined on partial orders and exploited for progressive resolution. In this thesis, we explore opportunities of partially ordering entity mentions to develop a generalized set-based framework that can be adapted to ER tasks such as progressive author disambiguation, hierarchical affiliation resolution and large-scale duplicate detection. In a series of works, we have explored the topics of clustering, blocking and progressive resolution in the context of author disambiguation. This was followed by experiments with hierarchical resolution of affiliation strings and billion-scale blocking for detecting duplicate publication records. In the process, a modular entity resolution framework was refined that consists of the steps (1) representation, (2) specification, (3) generalization, (4) separation, (5) collocation and (6) conflation. Entity mentions are (1) represented as sets of attribute-value pairs, which are in some cases (2) isolated by specification if they are not informative enough. Further, (3) hypothetical representations are added by removing features that are not required for blocking equivalence. The result corresponds to sufficient overlaps for blocking equivalence. Then, (4) the representations are separated into super-blocks consisting of representations that are somehow connected in the subset partial order, which is built explicitly in (5) collocation. Finally, (6) edge-weights based on observation counts can be used in the partial order's directed acyclic graph to progressively contract edges, thereby merging nodes (blocks) to increase the size of clustering tasks. Each development step in the series of publications related to this thesis has been evaluated on gold datasets for different application scenarios in the domain of bibliographic data. Thereby, we have proven the practicability of our approach, shown where it outperforms existing baselines and where current limitations call for further research. The description of our works is complemented by a brief discussion of each individual publication and embedded in the body of existing literature by an integrative introduction and preliminaries chapter as well as a dedicated related work chapter. In the final chapter, we conclude how we have been able to design a novel ER approach that is unique in how it combines a number of beneficial properties in a modular and easily adaptable framework. From a number of inherent limitations, we derive tasks for future work before summarizing our contributions and how they have addressed gaps in the existing ER literature.

Contents

List of Figures

List of Tables

Abbreviations

AD	Author Disambiguation
APA	American Psychological Association
ARL	Association Rule Learning
ASCII	American Standard Code for Information Interchange
1B	1 Billion
BSL	Blocking Scheme Learner
CEP	Cardinality Edge Pruning
CNF	Conjunctive Normal Form
CNP	Cardinality Node Pruning
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DCS	Duplicate Count Strategy
DNF	Disjunctive Normal Form
DOI	Digital Object Identifier
ER	Entity Resolution
FCA	Formal Context Analysis
GB	Gigabyte(s)
GERiT	German Research Institutions
HRP	Hierarchy of Record Partitions
ID	Identifier
ISO	International Organization for Standardization
1K	1 Thousand
kNN	k-Nearest Neighbor classifier
LBCE	Low Block Co-occurrence Excluder
LBCP	Low Block Co-occurrence Pruning
LBSP	Large Block Size Pruning

LECP	Low Entity Co-occurrence Pruning
LSH	Locality Sensitive Hashing
1M	1 Million
MFI	Maximum Frequent Itemset
MLA	Modern Language Association
PSN	Progressive Sorted Neighborhood
RAM	Random-Access Memory
RDF	Resource Description Framework
SA-LSH	Suffix Array Locality Sensitive Hashing
SA	Suffix-Array Blocking
SB	Standard Blocking
SN	Sorted Neighborhood Blocking
TB	Token Blocking / Terabyte(s)
WEP	Weighted Edge Pruning
WNP	Weighted Node Pruning
WoS	Web of Science

Symbols

x	some entity mention
f	some feature
X	some set of entity mentions
F	some set of features
$\mathcal{X}$	set of all mentions x
$\mathcal{F}$	set of all features f
$R(x)$	representation of x
	set $F \subseteq \mathcal{F}$ of features f describing x
$\text{Я}(f)$	set of all mentions with feature f
	$\text{Я}(f) = \{x \in \mathcal{X} \| f \in R(x)\}$
$\sigma(X)$	features common to all $x \in X$
	$\sigma(X) = \bigcap_{x \in \mathcal{X}} R(x)$
$\tau(F)$	mentions represented by F or a superset
	$\tau(F) = \bigcap_{f \in \mathcal{F}} \text{Я}(f)$
$M(F)$	mentions represented *exactly* by F
	$M(F) = \{x \in \mathcal{X} \| R(x) = F\}$
$\mathcal{R}$	set of all representations / intensions R
	$\mathcal{R} \subseteq \mathcal{P}(\mathcal{F}), \mathcal{R} = \{R(x) \| x \in \mathcal{X}\}$
$\mathcal{T}$	set of all extensions $\tau(F)$
	$\mathcal{T} \subseteq \mathcal{P}(\mathcal{X}), \mathcal{T} = \{\tau(R(x)) \| x \in \mathcal{X}\}$
$\mathcal{M}$	mention partitioning by representations
	$\mathcal{M} \subseteq \mathcal{P}(\mathcal{X}), \mathcal{M} = \{M(R(x)) \| x \in \mathcal{X}\}$
$(\mathcal{R}, \leq)$	subset partial order over intensions
$(\mathcal{T}, \leq)$	subset partial order over extensions

Chapter 1

Introduction

Entity resolution (*ER*) is the task of grouping references or mentions of entities in structured, semi-structured or unstructured data by the real-world object they refer to. Popular examples are author mentions referring to real-world persons, metadata records referring to publications or affiliation strings referring to institutions. ER allows to relate all the information regarding a specific entity over a large amount of data. Thereby, it is central to ordering and aggregating knowledge in the growing amount of information available today. At its core, ER is about finding indicators of identity. In practical terms, this is implemented as a similarity over the feature-based representations of entity mentions. While the fine-tuning of such similarity measures is a respectable research task, the most essential aspect of ER is certainly *scale*. In any larger collection, it is infeasible to compare all pairs of entity mentions by a similarity measure. Therefore, techniques have to be deployed that enable a prior structuring or partitioning of the search space. This allows confining oneself to the most likely coreferences by retrieving the most promising entity mention pairs for comparison without computing the entire ranking of all pairs. A simple approach is to establish a disjoint or overlapping grouping of all entity mentions based on selected key features or combinations thereof. This is known as *blocking*. A more elaborate family of solutions aims at suggesting mention pairs one-by-one (or in batches), following the order of their coreference likelihood. This is referred to as *progressive* resolution or blocking and aims at obtaining the best possible result for the invested resources. Due to a lack of alternatives, blocking is a necessity and due to its superior usefulness, the idea of progressiveness is considered a guiding principle in this thesis. In contrast to $O(n^2)$ pairwise comparison, the most basic blocking is implemented as an $O(n)$ labelling process that is based on one or more features and only considers one mention at a time. Progressive methods have focused on (alphabetically) sorting mentions in $O(n \log n)$ based on one or more features, which assumes a *total order* of representations. Instead, in this thesis, we make a case for *partially ordering* representations, as in many cases – in particular when there are missing values in the mentions' representations – the subset partial order over set-based entity representations is more appropriate than the alphabetical order over string-based representations. In the directed acyclic graph of the partial order, measures of neighborhood can be defined to find the most similar mention pairs without pairwise comparison. Generally speaking, the goal of this thesis is to explore the opportunities and limitations of partially ordering entity mentions in the ER context and to develop a generalized formalization to embed our proposal into the existing range of methods for entity resolution and progressive blocking. Specifically, our contribution is the introduction, integration, evaluation and discussion of a novel, conceptually coherent framework for (progressive) entity resolution that embraces a number of essential features and as such appears to be one of few openly published methods – or even the only one – to combine in itself many of the benefits attributed to these features.

1.1 Motivation

There is no doubt that the amount of digital data, information or knowledge in the world is growing at astonishing rates, given that there are more and more efficient methods for content creation and recombination used by a growing number of people with access to such means. This is complemented by machines that automatically copy, extract, interpret and summarize from exiting data. In this context, one can argue that the actual amount of essential facts is not growing at the same rate – and certainly, data growth is accompanied by considerable redundancy. Hence, when making sense of large datasets, for example by aggregations like ”Find me all the information concerning company XYZ”, it is essential to resolve the varying references to this entity. This is known as the entity resolution problem. Finding all references to a specific entity (a.k.a. *entity linking*) allows to aggregate information about it. Resolving all references to *any* entity in a collection allows to determine and distinguish which entities are mentioned where. Performing such entity resolution over all kinds of entity types can be assumed a prerequisite for determining general semantic overlap, in particular if one takes an entity-centric approach to *textual entailment*.

1.1.1 The Impact of Entity Resolution

Since entity resolution has numerous specific applications across different domains (see Table 1.1) and is essential to redundancy reduction and aggregation, it not far-fetched to claim that it is a prerequisite for most large-scale data processing tasks. As a result of its practical relevance, not all applications are harmless and for example must be expected to play a significant role in global mass surveillance as revealed by contemporary whistle-blowers like Edward Snowden:

> *And what's more you can tag individuals using ”XKeyscore”. [...] I can track your username on a website on a form somewhere, I can track your real name, I can track associations with your friends and I can build what's called a fingerprint which is network activity unique to you which means anywhere you go in the world anywhere you try to sort of hide your online presence hide your identity, the NSA can find you and anyone who's allowed to use this or who the NSA shares their software with can do the same thing.*
>
> – Edward Snowden on *XKeyscore*[1]

[1]https://web.archive.org/web/20140128224439/http://www.ndr.de/ratgeber/netzwelt/snowden277_page-3.html, retrieved 29.06.2021

While the application of ER can have positive and negative consequences, it is certain to have a considerable impact on the world today.

1.1.2 The Need for Scalability

A fundamental problem in ER is that even if the data is growing linearly over time, the number of potential coreferring pairs is growing quadratically. Neither the computational power of existing machines nor the amount of energy available to them can keep up with a task that becomes unsustainably inefficient in comparison to the source data size and the target knowledge gains. Given the increasing prevalence of ER in various domains like management, analysis and surveillance, this efficiency problem also has an impact on general sustainability and carbon emissions. While it acknowledges that the global computing energy consumption remains flat due to economies of scale in data centers, a recent International Energy Agency (IEA) report [3] also identifies machine learning as an emerging technology that is likely to have an increasing impact on computation energy demand.

1.1.3 A Cross-Domain Problem

Due to the number of different application domains and as a result of the many synonyms for both the term "entity" and the term "resolution", what we refer to as "entity resolution" is also known under many other names. This manifests itself in all kinds of compounds as indicated in Table 1.2. While not all noun-verb combinations in this table are conventional (e.g. "record disambiguation" is not), a large number of them are used in some contexts (e.g. "identity resolution" or "object identification"). Further terms like "merge/purge processing"/"list washing" or "duplicate detection"/"deduplication"

Table 1.1: Different example applications by domains.

management	
cleaning	e.g. *duplicate detection*
customer relations	e.g. *personalization*
analysis	
real-time analysis	e.g. *trend analysis*
general knowledge extraction	e.g. *knowledge graph learning*
specific knowledge extraction	e.g. *question answering*
verification	e.g. *fact checking*
surveillance	
intelligence	e.g. *counter-terrorism*
compliance	e.g. *auditing*
prosecution	e.g. *evidence compilation*
prevention	e.g. *fraud detection*

exist, as pointed out in the Wikipedia article on “Record Linkage”[2]. The number of different terms for the task is likely to have contributed to a lack of communication across domain boundaries, underlining the importance of a domain-independent approach that can be adapted to any concrete task. Here, general means for inspection and visualization are appreciated, as they are beneficial to the application of a general purpose solution to individual scenarios by facilitating the necessary fine-tuning.

1.1.4 The Intricate Relationship of Matching and Equivalence

This thesis focuses on handling diverse references to the same entity. While entity resolution generally embraces the assumption of resolving not only the most apparent equivalences, but also finding more inconspicuous ones, few approaches model the actual logical or world-knowledge-based relationships between individual references in terms of matching or contradiction. For example *John Doe* vs. *Jack Doe* might be equally similar as *John Doe* vs. *J. H. Doe*, but only the latter pair matches from a practical point of view, while the first can be considered contradictory. This can be uncovered when parsing the strings into sets of their essential components, in which case we see that the first name *John* contradicts the first name *Jack*, while the first initial *J* is shared by *John Doe* and *J. H. Doe* and the second initial *H.* can complement any first name such as *John*. These relations are important as they express what can in principle co-refer and what cannot (necessary conditions for equivalence). This constitutes a first, binary approximation of the coreference likelihood that we would like to account for when deciding actual coreference (sufficient conditions for equivalence). In their fundamentality, matching-based necessary conditions allow a structuring of the search space that provides means for avoiding the expenses of exhaustive pairwise comparison. The central part of the work on this thesis was initially motivated by addressing obvious inadequacies in the handling of matching author names by traditional blocking schemes as well as observing and understanding the inherent incompatibility of the matching relation and the equivalence relation.

[2]https://en.wikipedia.org/wiki/Record_linkage, retrieved 11.07.2021

TABLE 1.2: Analysis of various terms that are synonymous to ’entity resolution’.

INPUT-RELATED	
noun	**verb**
record	*linkage*
data	*matching*
name	*disambiguation*
reference	*reconciliation*
information	*integration*

OUTPUT-RELATED	
noun	**verb**
object	*resolution*
entity	*identification*
identity	
reference	

1.2 Problem Statement and Objectives

In ER, we are presented with a set of entity mentions, each of which is represented by a set of features. The task is to use those features to find groups of entity mentions that refer to the same entity. An entity can be virtually anything, but often is considered a real world person, institution or publication. Ultimately, equivalence of entity mentions is based on some explicit or implicit inter-mention similarity. With an appropriate definition of similarity, the most similar mentions can be considered coreferring. ER has two sides. The first deals with determining an effective definition of similarity (often with respect to a specific domain or even dataset). As the target output constitutes a grouping of mentions, transitivity is required (if a is equivalent to b and b is equivalent to c, then a must also be equivalent to c). This can be a blessing or a curse. The blessing is that one can use multiple surrounding mentions to determine which group a mention belongs to. Most clustering methods will exploit this information. The curse is that transitivity can propagate errors, especially if the belonging to a group is determined only based on one nearest neighbor (e.g. single-link clustering). The second side of ER addresses the technical infeasibility of comparing all mention pairs to obtain their similarity and decide coreference. While single-link clustering (or pairwise classification with transitive closure) constitutes the cheapest clustering approach, more effective clustering methods like agglomerative clustering require much more time and even those are only approximations towards optimizing intra-cluster similarity and inter-cluster dissimilarity. Therefore it is essential to avoid comparing mention pairs that are clearly not co-referring. In other words, the task is to compare only mention pairs that satisfy some necessary conditions for equivalence. This is usually referred to as *blocking*. As previously indicated, in this work we consider the second side of ER to be the essential problem since clustering methods are well studied and optimal similarity measures are likely to vary between different domains and datasets.

To identify gaps in the existing body of literature on entity resolution, a number of desirable properties of a useful ER solution have been summarized in Figure 1.1. To the best of our knowledge, no approach has been described in the scientific literature that incorporates all of them. The goal of this thesis is to develop such a method. The solution should be straight-forward but not trivial and offer a novel and convenient approach to ER that integrates and complements prior research in the field.

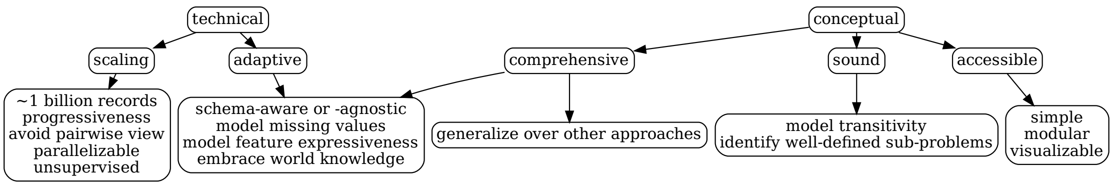


FIGURE 1.1: Properties of a good ER method.

1.2.1 Scaling

A good ER method should find duplicates in a collection of **up to 1 billion records** in a matter of days, given realistic hardware constraints. Generally speaking, modern consumer hardware can compute billions of simple operations in a few minutes if the information is retrieved instantly (from RAM). However, this requires very large primary memory, which is not available in PCs or laptops. Indexed databases stored on large solid-state drives can step in at this point, although it then becomes a matter of hours not minutes. Independent of these I/O challenges, it is essential that the cost of each of the above operations is decoupled from the input size. The bottom line is that the billion-scale is technically not infeasible, but will inevitably expose any design mistakes. Therefore, it is very practical to test an ER approach's maturity. While there are a few commercial applications that advertise resolution at such scale (e.g. *Quantexa*[3] or *Senzing*[4]), almost all scientific publications on the matter settle for a few million records as “large” cases.[5] In addition, a strong ER method should allow for **progressive resolution**, i.e. the continuous improvement of the output. It is common to assume the improvement to be an increase in recall as a result of comparing more and more pairs (ranked by coreference likelihood). While progressive ER is a growing field, to this date, most approaches still do not embrace this extremely convenient paradigm (cf. Papadakis et al. [1]). In our blocking approach, we **avoid perspectives that refer to pairwise comparisons**, for example the notion of specifically determining the similarity of two entity mentions or the notion of pruning pairs (e.g. Papadakis et al. [6]). We try to implement our methods such that the respective task **can be run in parallel**. This property has seen wide-spread adoption in the literature (e.g. Efthymiou et al. [7], Altowim and Mehrotra [8]) – but it is important to note that parallelization is no cure for quadratic complexity. By **not assuming a supervised classification task**, we are relieved from requiring training data (although of course we still need some annotated data for evaluation purposes). In addition to the problem of scaling annotation, a classification scenario implicitly assumes pairwise comparison (classifying pairs as co-referent or not), which can lead to problems in scaling an ER method's

[3]`https://www.quantexa.com/platform/entity-resolution/`, retrieved 29.06.2021
[4]`https://senzing.com`, retrieved 29.06.2021
[5]Notable exceptions are Manku et al. [4] and Zhang et al. [5].

application in terms of the processed collection size. Regardless of those problems, the classification view on ER is still found frequently in the literature (e.g. Draisbach et al. [9], Koumarelas et al. [10]).

1.2.2 Adaptivity

In order to adapt to all kinds of datasets from different domains, a versatile ER approach should **allow to include schema information** in feature-representations, which essentially constitutes the attribute-part of attribute-value pairs, but must also be able to work without it. Although some literature [1, 11] creates the impression of a deeper distinction between schema-aware and -unaware methods, the latter are in fact easily derived from the first by removing or generalizing attributes. Our method must be able to **properly model missing values** by distinguishing matching representations of different information granularity from those with contradictory values. For example, *J. Doe* lacks a value for the attribute ”first given name”, but is not contradictory to *John Doe*, which has both a value for ”first initial” and ”first given name” – in other words, a missing value does not contradict any values. Also, it is desirable that there are options to **weight features by their expressiveness**, e.g. if two representations share a rare feature, they are more likely to co-refer than if they share a frequent feature. This should be determined automatically so that it also helps the adaptability to unknown datasets. The discriminatory power of individual features is essential to determining a reliable similarity as it means distinguishing important from unimportant differences. A useful guide for distinguishing relevant and irrelevant differences is *world knowledge*. While it is not good if an approach *requires* world knowledge, the latter can still be very useful if available. Therefore, an adaptable method should **embrace world knowledge** without compromising general automatic adaptivity. Most existing approaches either rely on such knowledge (these are usually works on specialised ER like author disambiguation) or reject it altogether [12, 13].

1.2.3 Comprehensiveness

In addition to the features mentioned under “adaptivity”, a comprehensive approach to ER should **incorporate as many methods from prior work as possible**. To contextualize an approach and to enable better comparison, it is highly appreciated if it can be defined within a framework that is general enough to encompass as many other solutions as possible. Then, existing solutions are special cases invoked by certain hyperparameter choices rather than separate models. Often, seemingly high-level distinctions can be easily described as parameter-configurations of a single "master-method" (see also *soundness*). We elaborate on this in the related work chapter.

1.2.4 Soundness

In this thesis, we strictly require *transitivity* (together with the more obvious *reflexivity* and *symmetry*) of the extracted coreference information, that is a sound ER method should ultimately **return a partitioning (labelling) of the input records**. It should *not* return (partial) pairwise coreference information, where it is unknown whether $a \equiv b \land b \equiv c \Rightarrow a \equiv c$. The latter problem is likely to occur if one applies a classification view to ER. The classifier decisions (and their evaluation) requires the transitive closure to be applied afterwards [9, 14] (with consequences difficult to predict). Also, where possible, one should **describe sub-tasks by means of well-studied problems** in computer science (e.g. connected component search or nearest neighbor problem, etc.). Generally speaking, there are two types of papers on ER methods: Those accessible to the regular scientific software engineer (e.g. Papenbrock et al. [15]) are lacking in terms of conceptual decomposition. The alternative is mathematically derived approaches (e.g. Judson [16]) that focus on known problems and feature many theoretical proofs, but are often hard to understand in terms of concrete implementation guidelines. In our opinion, a middle course is desirable.

1.2.5 Accessibility

To foster reproducibility, reuse and advancement of a proposed method/prototype, it is important that it can be understood as a conceptional framework and implementation template rather than a blackbox of cryptic code. It is therefore important that the underlying principles remains **simple and modular**. Modularity allows to view the approach on different levels of detail and "zoom in" to a specific component without having to consider what is happening in the others. This includes the distinction between blocking and clustering. While modularity is sometimes achieved on a technical level (e.g. in Papadakis et al. [17]), this separation is then usually not along conceptional boundaries. As a result, different methods are implemented as black-box modules instead of being instantiated by different parameter configurations in an overarching conceptual framework. For error analysis and continuous improvement within the parameter and hyper-parameter space defined by our framework, it is important that our method **creates visualizations for specific ER scenarios**. Although sometimes theoretical examples are displayed graphically in the literature (e.g. [7]), these are hardly ever extracts from the real data, which suggests that no automatic visualization has been implemented.

1.2.6 Further aspects

The following aspects are desirable, but could not be incorporated into this thesis due to time constraints. Real-time ER does not assume a static collection of records in which duplicates are to be found, but a dynamic one, that is over time, mentions are added – and also removed. Ultimately, the update-based framework that addresses this problem is the only realistic setup, especially in very large-scale cases, where complete re-computation is very expensive. However it is also very complicated to properly model transitivity in this context, in particular if one makes the realistic assumption that removed items must be "forgotten" due to privacy legislation. Similarly, dynamically adapting the input (especially in progressive scenarios) by feeding back each resolution (merge) decision (e.g. by combining representations of mentions determined co-referent) offers great promise. However, this too involves great challenges in transitivity and increases the danger of error-propagation. For conceptual and mathematical simplicity, *order-invariance* (i.e. the order of operations is irrelevant within each step) is desirable as it allows dynamic modification without unexpected consequences, e.g. we can expect that first adding and later removing an entity mention has the same effect as never having added it in the first place.

1.3 Publications and their Contributions

The main contribution of this thesis lies in the proposal of a novel integrated, formalized and easily accessible solution with the properties discussed in the previous section. The novelty lies on the one hand in the method used, but also in the entirety of its beneficial properties. This thesis should describe one of the first openly published approaches that combines these advantages and can be implemented and comprehended in a relatively straightforward way due to its formal simplicity. The approach has been developed and refined over the course of multiple publications, which are all attached to and discussed in this thesis. In the following, we list and summarize the individual projects and afterwards point out their individual contribution to the thesis:

1. Tobias Backes. Effective unsupervised author disambiguation with relative frequencies. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries (JCDL'18)*, pages 203–212[6], Fort Worth, Texas, USA, 2018. ACM Press. ISBN 978-1-4503-5178-2. doi: 10.1145/3197026.3197036. [18]

2. Tobias Backes. The impact of name-matching and blocking on author disambiguation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*, page 803–812[7], New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360142. doi: 10.1145/3269206.3271699 [19]

3. Tobias Backes and Stefan Dietze. Lattice-based progressive author disambiguation. *Information Systems*[8], 109:102056, 2022. ISSN 0306-4379. doi: 10.1016/j.is.2022.102056. [20]

4. Tobias Backes, Daniel Hienert, and Stefan Dietze. Towards hierarchical affiliation resolution: framework, baselines, dataset. *International Journal on Digital Libraries*[9], May 2022. ISSN 1432-1300. doi: 10.1007/s00799-022-00326-1. [21]

5. Tobias Backes and Stefan Dietze. Subset partial order components for blocking billion entities. *Submitted to 49th International Conference on Very Large Databases (VLDB).*

The first two publications are single-author contributions where the research conceptualization and execution was conducted completely independently. Likewise, most of the

[6]Full research paper.
[7]Full research paper.
[8]The journal published by Elsevier, not to be confused with *Information Systems Journal* by Wiley.
[9]Published by Springer.

work for the other publications was done by the first author, incorporating suggestions by the other authors regarding experimental design as well as their contributions to the structure, writing and presentation of the manuscripts. In the fourth work, Daniel Hienert has also contributed means for accessing the data and presenting the results online. In addition, this work was partially motivated by a practical need for institutional disambiguation within GESIS and funded by the respective DFG project "Scalable Disambiguation of Institutions for Web of Science", the proposal for which was mostly sketched by the first author as well.

1.3.1 Contributions: "Effective Unsupervised Author Disambiguation with Relative Frequencies"

This work contributes an agglomerative clustering approach for author disambiguation in the large-scale Web of Science, emphasizing the topic of evaluation, in the sense that the clustering method is visualized as a process of precision and recall *development* and in relation to trivial baselines. This allows to verify the plausibility of the performance numbers and observe the true contribution of the proposed method. Doing so, it incorporates the following desirable properties:

Scaling: parallelizable in that surname+all-initials blocks can be considered separate jobs to be processed by a pool of workers ✓

Scaling: unsupervised with only one parameter (stopping criterion) to be tuned ✓

Adaptivity: model feature expressiveness through a probabilistic weighting ✓

Soundness: model transitivity by relying on clustering instead of pairwise classification ✓

Soundness: identify well-defined sub-problems by using the basic agglomerative clustering framework ✓

Accessibility: simple in that it can be described by a few probabilities, which are implemented as simple matrix operations – mostly matrix multiplication ✓

Accessibility: visualizable as the clustering development can be visualized for inspection ✓

This work was conducted entirely independently.

1.3.2 Contributions: "The Impact of Name-Matching and Blocking on Author Disambiguation"

The contribution of this work focuses on blocking and name-matching in the context of author disambiguation to address the shortcoming of the previous publication in that this one only considered the *surname, all-initials* blocking scheme. It orders author names of different completeness in a partial order to visualize their relationships. Some name pairs are contradictory while others can potentially refer to the same person. Existing blocking schemes as well as one novel method are implemented to operate on this acyclic directed graph. Their effectiveness in the task of disambiguation is evaluated and compared. This includes the general question of how much of a disambiguation method's performance can be attributed to name-based blocking alone. Doing so, it incorporates the following desirable properties:

Scaling: avoid pairwise view as author mentions are directly sorted into the right buckets depending on their name information ✓

Scaling: parallelizable in that mentions can be sorted into buckets based on surname, first-initial so that further name-distinction falls into separate jobs ✓

Scaling: unsupervised in that the blocking methods do not require training data and are mostly parameter-free ✓

Adaptivity: model missing values by the partial order of name representations.

Adaptivity: model feature expressiveness in a local context through the proposed entropy-based method that considers the relative frequency of name-representation compared to more general names ✓

Adaptivity: embrace world knowledge in the context of parsing name representations from the given strings ✓

Comprehensiveness: generalize over other approaches that can be implemented in the partial order of name representations ✓

Soundness: model transitivity in the partial order of name representations.

Accessibility: simple in that it only builds the subset partial order over name representations and isolates elements that satisfy certain criteria to implement blocking schemes ✓

Accessibility: visualizable by plotting the partial order's DAG for selected surname, first-initial combinations ✓

This work was conducted entirely independently.

1.3.3 Contributions: "Lattice-based Progressive Author Disambiguation"

This work formalizes the relationships between different name-representations further and advances the previously proposed entropy-based blocking scheme into a progressive blocking method. Multiple ways of dynamically modifying edge weights in the blocking graph are compared to determine their effectiveness in terms of found duplicates vs. compared author mentions. The method is completed by applying single-link and agglomerative clustering (the latter taken from the first publication) to obtain an end-to-end disambiguation system and evaluate its blocking and clustering performance. Doing so, it incorporates the following desirable properties:

Scaling: progressiveness is introduced through iterative merging of name representations adjacent in the subset partial order of name representations ✓

Scaling: avoid pairwise comparisons in the blocking method itself, which are postponed to within-block clustering ✓

Scaling: parallelizable by considering surname+first-initial blocks as separate jobs, i.e. "super-blocks". Simultaneous parallel iterative progression across all super-blocks was not implemented, although a deployed system must realize it ✓

Scaling: unsupervised blocking and unsupervised disambiguation ✓

Adaptivity: model missing values in the blocking method ✓

Adaptivity: model feature expressiveness locally in blocking and globally in disambiguation ✓

Adaptivity: embrace world knowledge in the name parsing ✓

Comprehensiveness: generalize over other approaches describing baselines in the blocking framework as before, which now also allows for the definition of some progressive blocking baselines ✓

Soundness: model transitivity at all times by block merging and clustering ✓

Soundness: identify well-defined sub-problems in the new formalism, e.g. (semi-) lattices, partial order production, edge contraction, graph minors, minimal spanning trees, transitive reduction, nearest neighbor search, discounting, association rule learning, hierarchy of record partitions, etc ✓

Accessibility: simple and modular as it mostly consists of representation parsing, partial order production, weight modification, edge-contraction, condensation and (re-)clustering ✓

Accessibility: visualizable through iteratively contracted directed acyclic graphs ✓

Conceptualisation and implementation of the approach and large parts of the writing have been conducted independently. Co-authors contributed to the experimental setup, writing, presentation and revision.

1.3.4 Contributions: "Towards Hierarchical Affiliation Resolution: Framework, Baselines, Dataset"

This work is the most complex in the collection of publications associated with the thesis as it introduces institutions as a new entity type and also considers the potential of the blocking graph to go beyond an intermediate structure for obtaining comparison order and towards an approximation of true institutional hierarchies. Due to its complexity, this work is considered mainly an entry point for developing improved methods that address the identified subtasks. A major aspect of general significance studied in this context is to determine the super-blocks as previously set manually (surname+first-initial), now automatically through minimal element or connected component search. Doing so, it incorporates the following desirable properties:

Scaling: avoid pairwise formulations in this approach with the exception of a limited number of adjacent representations regarding merging decisions ✓

Scaling: parallelizable in minimal element search and the processing of the resulting overlapping components through a pool of workers processing a queue of jobs ✓

Scaling: unsupervised without requiring training data ✓

Adaptivity: schema-aware or -unaware as the baseline implementation does actually not need to be schema-aware, but attributes seem to play an important role for better conflation methods ✓

Adaptivity: model missing values as before ✓

Adaptivity: Embrace world knowledge in improved conflation methods and also in representation parsing, which is much more difficult than for author names ✓

Comprehensiveness: generalize over other approaches as generic top-level resolution is a special case of this approach ✓

Soundness: models transitivity as by the anti-symmetry of the subset/superset partial order, an affiliation can refer to separate top-level institutions without merging them ✓

Soundness: identify well-defined sub-problems by using connected component- and minimal element search ✓

Accessibility: modular due to the increased complexity of the task, establishing the separate subtasks of representation, interpolation, collocation, conflation and separation ✓

Accessibility: visualizable through visualizing the extracted institutional hierarchies by plotting the underlying directed acyclic graph ✓

Conceptualisation and implementation of the approach and large parts of the writing have been conducted largely independently. Co-authors contributed to the experimental setup, writing, presentation and revisions. The second co-author also supported the experimental implementation by providing input data and the presentation of the result output.

1.3.5 Contributions: "Subset Partial Order Components for Blocking Billion Entities"

This unpublished work focuses on the technical challenge of creating super-blocks through parallellized minimal element- or connected component search with finite memory resources. It provides a method for extremal set discovery in the superset partial order without full partial order production (loading the graph into memory). Thereby, it constructs a bipartite graph that has the same connectivity as the full partial order but is considerably smaller and facilitates connected component search to identify separate super-blocks that are guaranteed to be unrelated. The method can also be used for partial order production as it computes all superset relations on the fly. Doing so, it incorporates the following desirable properties:

Scaling: 1 billion records can be processed on limited hardware by the implementation to find connected components. In fact the memory-runtime tradeoff can be controlled to some extend.

Scaling: avoid pairwise formulations as that would be clearly infeasible ✓

Scaling: parallelizable as this is at the heart of the proposed algorithm ✓

Scaling: unsupervised and completely parameter-free and unsupervised ✓

Adaptivity: schema-aware or -unaware in the sense that representations can be sets of both values and attribute-value pairs ✓

Adaptivity: model missing values as before through the superset partial order ✓

Soundness: model transitivity as connectedness is transitive by definition ✓

Soundness: identify well-defined sub-problems by constructing a bipartite graph with equal connectedness ✓

Accessibility: simple as the parallelization approach using 'batches and patches' can be visualized in two dimensions and deploys a standard job-based framework as opposed to an existing recursive baseline ✓

Accessibility: modular with two modules: Finding the minimal elements and finding connected components in it ✓

This work was conducted mostly independently, with the exception of some feedback regarding experiments and the revisions to early manuscript drafts.

1.4 Structure of this Thesis

As summarized in Table 1.3, the remainder of this thesis is structured as follows: In **Chapter 2**, we pursue an extended problem exploration trying to link the entity resolution task and in particular the basic concepts involved in avoiding exhaustive pairwise comparison to general fundamentals from mathematics and computer science. Then, we present the latest version of our end-to-end entity resolution approach, which has been developed over the years. In **Chapter 3**, we go into more detail describing individual fundamental and current literature related to the problem of entity resolution. This is meant to complement, not replace the related work sections of our separate publications. In **Chapter 4**, we explore clustering for entity resolution based on the example of grouping author mentions into real-world author entities with a focus on developing a useful similarity measure for mention-mention and cluster-cluster similarity in an agglomerative clustering framework. In **Chapter 5**, we model the relationship between matching and contradictory blocking representations based on the example of author mentions in the Web of Science to shed a light on the distribution of name representations, their matching relationships and how it relates to different blocking schemes, which are also compared performance-wise. In **Chapter 6**, we combine and refine the insights and methods from the clustering paper (Chapter 4) and the blocking paper (Chapter 5) to view the contribution of each task to the overall author disambiguation performance. In this context, we embrace the notion of progressive resolution for author disambiguation and present the first such method. In **Chapter 7**, we experiment with the use of the subset partial order of entity representations for describing hierarchical relationships in the real world. The focus was on the application of previous work around the subset partial order of blocking representations on institutions as a new type of entities to disambiguate international affiliations. Hierarchical aspects derive naturally from the subset partial order. In **Chapter 8**, we study the separation problem, the task of creating super-blocks inside which progressive resolution can be applied. The objective is to compute connected components in the subset partial order of entity representations by relating every representation to its minimal element and using this condensed representation to compute connected components. The focus of this work is to identify options for balancing speed and space requirements to enable fast search of connected components in the subset partial order over roughly one billion representations of publication records from the CORE collection to find duplicates. In **Chapter 9**, we conclude the work done in the context of this thesis by stating its main results, summarizing the research process, deriving remaining questions for future work and underlining our contributions to the field.

TABLE 1.3: Overview of the individual publications that make up the backbone of this thesis.

Chapter	Topic	Goals	Desirable features incorporated															Domain	Main methods
			billion-scale	progressive	no pairwise view	parallelizable	unsupervised	schema-ignorant	missing values	feature weighting	world knowledge	generalizes other	models transitivity	sub-problems	simple	modular	visualizable		
4	Clustering	▶ author disambiguation in the Web of Science ▶ visualizing clustering process ▶ view true effect of proposed method compared to trivial baselines				🟩	🟩			🟩			🟩	🟩	🟩		🟩	authors	▶ agglomerative clustering
5	Name Matching and Blocking	▶ investigate relationship between name matching and blocking ▶ evaluate different blocking schemes			🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩		🟩		🟩	authors	▶ edge removal in subset partial order DAG
6	Progressive Blocking	▶ introduce progressive author disambiguation ▶ derive coreference likelihood from degree of name-matching		🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	authors	▶ progressive edge contraction in subset partial order DAG
7	Hierarchy Extraction	▶ introduce task of hierarchical institution resolution ▶ model institutional hierarchy by subset partial order			🟩	🟩	🟩	🟩	🟩			🟩		🟩		🟩	🟩	institutions	▶ parsing ▶ minimal element search ▶ selective edge contraction in subset partial order DAG
8	Connected Components	▶ efficient and effective connected component search in subset partial order ▶ parallelized minimal element search ▶ large-scale duplicate detection	🟩		🟩	🟩	🟩	🟩	🟩				🟩	🟩	🟩	🟩		publications	▶ partial order production ▶ minimal element search ▶ connected component search

Chapter 2

Preliminaries

In this chapter, we pursue a bottom-up process of problem exploration, where we derive important characteristics by describing and addressing first the basic entity resolution task and then consecutively all follow-up challenges and opportunities. We relate these essential aspects to mathematical concepts and other fundamentals from different research areas, without referring in detail to (recent) individual works (not to preempt the literature review). As a result of this theoretical exercise, in the second part, we present our approach for end-to-end entity resolution that has been developed over the course of the five research projects which make up this thesis.

2.1 Problem Exploration

In this section, we go back to first principles in order to provide a formal groundwork for the methods proposed later. The basic problem setting is search for equivalent pairs in a large set of elements that are each represented as a set of features. We touch upon a number of different research fields and existing approaches where they are applicable. In some cases, a deep exploration is not feasible. References to existing work are further elaborated on in the related work chapter, where they are complemented by a description of individual publications that summarize or establish the respective approaches.

2.1.1 Equivalence and equality

Generally speaking, the task of entity resolution is to establish a *relation* over a set X, trivially by comparing all pairs $x, x' \in X \times X$. In our case, we are interested in *equivalence relations*, which means that the target relation is *transitive*, *symmetric* and *reflexive*. Let *equivalence* of two entity mentions be denoted as $x \equiv x'$ or $eq(x, x') = 1$. Equivalence is not the same as *equality* $(x = x')$. For example, we say two entity mentions are equivalent if they refer to the same (equal) real world entity. No two mentions are equal, but might have equal *representations*, where a representation is a set or multi-set of features used to describe the mention. The equivalence relation *partitions* X into *equivalence classes*. We must assume that any comparison technique can only return an approximation of true equivalence and must be evaluated against a gold standard of known equivalences to determine its degree of correctness.

2.1.2 Transitivity of partitioning

For better conceptualization, in this work, we assume that any resolution method must return an equivalence relation, the crucial part being that it must satisfy transitivity and thereby partition X. This cannot be taken for granted: an imperfect simple pairwise comparison over X might fail to return such partitions, unless the transitive closure of the so-discovered equivalences is taken. Taking the transitive closure over a set of pairs detected as equivalent is the same as finding connected components in the undirected graph over X that has an edge exactly for each of these pairs and can be done in $O(n)$ or even faster [22]. Instead of using a binary equivalence classifier followed by transitive closure, one can also deploy methods that are in themselves guaranteed to partition the search space, like most clustering algorithms.

2.1.3 Categorization

Due to practical restrictions described earlier, the focus of this thesis and of many other works is to avoid the quadratic complexity of going over all $x, x' \in X \times X$ to determine their similarity or equality. As described above, there are two possible approaches to entity resolution: (1) deploy an equivalence classifier or oracle and present it with a limited number of pairs such that each mention occurs in at least one pair. Then perform the transitive closure to translate the coreference information for these sampled pairs into an approximative partitioning; (2) use a method that goes over elements $x \in X$ instead of pairs and on the fly creates in sub-quadratic time a structure (e.g. taxonomy) from which the equivalence can be read of in sub-quadratic time. A simple instance of such type of method is categorization with a complexity of $O(n)$, where each $x \in X$ is put into a certain category (or disjoint block) using a hash function, blocking scheme or categorization method $b(x)$. A popular example is surname, first-initial blocks for name-based person representations, which takes the name of a person and discards all information except surname and first initial, then puts the mention into the respective category if the latter already exists or else creates a new one with just the mention in it. This approach is compelling as it is not required to compare all elements to find that they belong together when togetherness is defined by categories. The categorization baseline still leaves a number of problems, for example when the information present in a mention's representation is not sufficient to apply the distinction scheme implemented in the taxonomy (e.g. there is no surname available, but the author mentions are sorted by *surname,first-initial*) or the number of mentions sorted under a category grows beyond measure (e.g. for Y. Wang). These examples show that it might be difficult to clearly define $b(x)$ without considering the categorized mention's relationships. When only considering individual mentions, $b(x)$ is likely to suffer from a lack of adaption to the collection as a whole. In other words, it is difficult to define mutually exclusive and collectively exhaustive categories without comparing mentions to each other, especially if the set of features is not fixed (i.e. we are trying to arrive at a general blocking scheme that can be applied to unseen representations). In a flat taxonomy, it is difficult to address volatile category sizes and differing information granularity.

2.1.4 Concept hierarchy; formal concept analysis

If a concept is defined by a set of features, then subsets correspond to generalizations of the respective concept. This leads to a hierarchy, in which different levels of abstraction exist. Every concept is described by a set of properties (its *intensions*). A sub-concept or specification adds at least one additional property to its super-concept or generalization. Hence the description of a specification is a superset of the description of its generalization. As the properties required by a super-concept form a subset of those required by one of its sub-concepts, the set of items that fit into the sub-concept (its *extensions*) is a superset of the sub-concepts extensions. Formal concept analysis (FCA) builds a concept hierarchy over mentions $x \in \mathcal{X}$ represented as feature sets $F \subseteq \mathcal{F}$. Each hierarchy node (concept) is defined by a set of necessary features $f \in \mathcal{F}$ (its intension $F \in \mathcal{R} \subseteq \mathcal{P}(\mathcal{F})$) and subsumes all elements with these features (its extension $X \subseteq \mathcal{X}, X \in \mathcal{T} \subseteq \mathcal{P}(\mathcal{X})$). Here, $\mathcal{R}$ is the set of all intensions F and $\mathcal{T}$ is the set of all extensions X. Each sub-concept (specification) contains a subset $X' \subseteq X$ of the extension defined by a superset $F' \supseteq F$ of the intension. The relationship between intension and extension is formalized as an *antitone Galois connection.* Given a set of features $\mathcal{F}$, for an element x, we define its *representation*

$$R(x) = \{f \in \mathcal{F} \mid x \text{ has feature } f\}$$

Given a set of elements $\mathcal{X}$, for a feature f we define its *represented elements*

$$\text{Я}(f) = \{x \in \mathcal{X} \mid x \text{ has feature } f\}$$

as inverted index. If X is a subset of $\mathcal{X}$, then $\sigma(X)$ denotes the set of common features:

$$\sigma(X) = \bigcap_{x \in X} R(x)$$

Correspondingly, for a subset F of features from $\mathcal{F}$, we define the set

$$\tau(F) = \bigcap_{f \in F} \text{Я}(f)$$

of all elements x that have all features in F. $\sigma(X)$ and $\tau(F)$ are called the derivatives of X and F. The functions σ and τ are called *derivative operators* and define an antitone Galois connection between the *power set lattice* of the extensions and that of the intensions. An antitone Galois connection is a pair σ, τ of antitone mappings $\sigma \colon \mathcal{T} \to \mathcal{R}$, $\tau \colon \mathcal{R} \to \mathcal{T}$ with *extensive* compositions $\sigma\tau$, $\tau\sigma$ between two *partially ordered sets* $(\mathcal{T}, \leq)$, $(\mathcal{R}, \leq)$. Mappings $\sigma\tau$,$\tau\sigma$ are extensive if $X \leq \tau\sigma(X)$, $F \leq \sigma\tau(F)$ for all $X \in \mathcal{T}$, $F \in \mathcal{R}$ respectively.

2.1.5 Partial orders; lattices

A partially ordered set is a set ordered by a reflexive, antisymmetric and transitive binary relation $\leq$. In this work, we specifically use the *subset/superset partial order* over feature representations $R(x)$ of entity mentions x. Here, $\tau(R(x)) = \bigcap_{f \in R(x)} Я(f)$ is the set of all entity mentions x' that have representations $R(x')$ which are equal to or specifications of $R(x)$. The subset partial order over the power set $\mathcal{P}(\mathcal{F})$ defines a *lattice*. In a lattice, every two elements have a unique *supremum* or *join* and a unique *infimum* or *meet*. If $(\mathcal{R}, \leq)$ is a partially ordered set, and $\mathcal{R}' \subseteq \mathcal{R}$ is an arbitrary subset, then a representation $F \in \mathcal{R}$ is an upper bound of $\mathcal{R}'$ if $F' \leq F$ for each $F' \in \mathcal{R}'$. An upper bound F is a supremum if $F \leq F''$ for each upper bound F'' of $\mathcal{R}'$. A set can have at most one supremum. A representation $F \in \mathcal{R}$ is a lower bound of $\mathcal{R}'$ if $F' \geq F$ for each $F' \in \mathcal{R}'$. A lower bound F is an infimum if $F \geq F''$ for each lower bound F'' of $\mathcal{R}'$. A set can have at most one infimum. A partial order with only a supremum forms a *join-semilattice* and one with only an infimum forms a *meet-semilattice*.

2.1.6 Directed acyclic graphs; association rule learning

If we build the subset partial order $(\mathcal{R}, \subseteq)$ over all observed representations $R(x)$ of entity mentions x, it is likely that among them there is none that qualifies as supremum or infimum. A partial order with neither join nor meet is not a lattice nor semi-lattice. However, all partial orders correspond to a *directed acyclic graph (DAG)* and partially ordered sets are commonly visualized by a *Hasse diagram* which is a drawing of the transitive reduction of this graph. A transitive reduction of a directed graph D is a directed graph with the same nodes and a smallest possible subset of its edges such that there is still for each pair of nodes a path where there was a path in the original graph. A graph can have multiple transitive reductions. [23] Correspondences between the graph view and the relation view include the weakly connected components of a DAG corresponding to the equivalence classes of the partial order's symmetric closure[1]. An application that works on the above DAG is *association rule learning*, in which strong connections or correlations (*associations*) between elements of the subset partial order over so-called *itemsets* are discovered with the help of edge weights like *confidence*, where the latter is similar to the transition probabilities introduced later in our work. However the classic application scenario of ARL is not entity resolution, but *shopping basket analysis*, where the task is to predict missing items in a shopping basket.

[1]The symmetric closure is obtained by adding (F', F) for each pair (F, F')

2.1.7 Necessity and sufficiency for equivalence

In order to analyze methods that reduce the number of comparisons, it is useful to distinguish necessary and sufficient conditions for equality. Thereby, we distinguish opposite ends of the coreference likelihood scale: A pair that does not satisfy necessary conditions for equality is assumed not to be equivalent. A pair that satisfies sufficient conditions for equality is assumed to be equivalent. Some conditions are both necessary and sufficient. All pairs that satisfy sufficient conditions for equality also satisfy the necessary conditions. This does not mean that the sufficient conditions itself are also necessary. For example, one might consider two mentions having the same email address as sufficient evidence for them referring to the same person, but this is not necessary, as one person can also have different email addresses. Each of the two types of conditions can be visualized as a border around the space of pairs that satisfy them. Hence in this image, overlapping borders signify conditions that are both necessary and sufficient and overlapping areas signify pairs that satisfy both necessary and sufficient conditions (as stated above, there is no area that is surrounded by sufficient conditions but not by necessary conditions). The gray area in between where necessary conditions are satisfied but sufficient are not, is subject to further investigation. While necessary conditions can be used to define blocks, sufficient conditions are ultimately defined in the clustering or pairwise classification step. In blocking, if the satisfaction of necessary conditions for equivalence can be ruled out a priori (i.e. two mentions are *contradictory*), two mentions should be placed in separate blocks. During clustering, if sufficient evidence for equivalence is found, two mentions should be placed in the same cluster. We will see that the transitive closure that is required on these relations in order to obtain disjoint partitions is likely to include some contradictory pairs in the same block. The same "problem" can apply during clustering, where the transitive closure includes pairs that are not sufficiently similar in the same cluster. For example *J. Doe* needs to be in the same block with *Jack Doe* and *John Doe*, but that means contradictory *John* and *Jack* are in the same block. The reason is a lack of information regarding *J. Doe*. On the other hand some mention x might be sufficiently similar to another mention x', and the latter sufficiently similar to another x'', but x' and x'' might not be that similar. Essentially this is the same problem in both cases, only that in blocking, here we assume a Boolean similarity of "matching" and in clustering a continuous similarity like (inverted) Euclidean distance or cosine similarity.

2.1.8 Rough sets

The problem of necessary and sufficient conditions of mention equivalence can also be described in the context of *rough sets.* If we have our set $\mathcal{X}$ of entity mentions and our set of features $\mathcal{F}$, each $f \in \mathcal{F}$ belongs to some feature-type or attribute $a \in \mathcal{A}$, so that each feature is actually an attribute-value pair $f = (a, v)$, where $v \in \mathcal{V}$ is a value that attribute a can take. This corresponds to a database or *information system* table giving zero or one value v for each mention x (row) and attribute a (column). If one mention can have multiple values for the same attribute, theoretically this can be achieved by a Boolean table where each column is a possible pair of attribute and value that is either present ($v = 1$) or not ($v = 0$). For any subset $A \subseteq \mathcal{A}$ of attributes, there is a possibly empty subset $IND(A) = \{(x, x') \in \mathcal{X} \times \mathcal{X} \mid \forall a \in A : a(x) = a(x')\}$ with all mention pairs (x, x') that have the same value $a(x) = a(x')$ for all attributes $a \in A$ and can therefore be considered equivalent based on the selection of attributes A. For example A could be $\{surname, firstinit\}$ of a person x and then we consider all persons equivalent if they have the same surname and first initial. As $IND(A)$ is an equivalence relation, which by definition creates equivalence classes over $\mathcal{X}$, this also partitions $\mathcal{X}$ into subsets X assigning each $x \in \mathcal{X}$ to its equivalence class $[x]_A$ by simply combining its values $v \in \{a(x) \mid a \in A\}$. If we assume that we know the true equivalence classes $E \in \mathcal{E}$, i.e. which mentions refer to the same real-world entities, we may study to which degree we can approximate these true equivalences by means of the attributes $\mathcal{A}$. As the "resolution" of these attributes is usually not perfect (e.g if we have only name-information available, there might be different persons with the same name or the same person goes by different names), our target sets most E cannot be described exactly. For this reason, each E is approximated by a rough set, which uses the notion of *upper-* and *lower approximation.* The lower approximation

$$L(E) = \{x \in E \mid [x]_A \subseteq E\}$$

is the set of all mentions x that can be grouped by the attributes A such that they are all in our target E. Over all $E \in \mathcal{E}$, the lower approximations correspond to all mentions that satisfy sufficient conditions for equality defined by our "vocabulary" A. The upper approximation

$$U(E) = \{x \in E \mid [x]_A \cap E \neq \{\}\}$$

is the set of all mentions that $IND(A)$ groups with mentions in E, which can also include mentions outside of E that are indiscernible to those inside. Over all $E \in \mathcal{E}$, the upper approximations correspond to all mentions that satisfy necessary conditions for equality defined by our vocabulary A. In the first step of representing mentions, we chose a set of attributes and so on this level equivalence classes defined by these attributes correspond to representations, each of which can represent multiple indiscernible mentions if we consider the empty value (e.g. NULL) to be a value like any other. Accounting for missing information (i.e. using NULL as a placeholder), all supersets are indiscernible from their subsets, which creates much larger equivalence classes, of which more later. In the application domain of rough sets, usually the target sets E are classes in a classification scenario and a classifier consists of a set of rules made up from the optimal set A of attributes and a mapping saying which shared values correspond to which classes. The target sets are annotated data and to allow for generalization towards unseen data, the set of rules is reduced by various techniques. This however is not directly applicable to entity resolution.

2.1.9 Bounds in continuous space

In a continuous space, two elements x, x' can be assumed equivalent if they are sufficiently close, i.e. $dist(x, x') \leq t_{lower}$, where t_{lower} is a lower bound on $\{dist(x, x') : x, x' \in X \times X \wedge x \equiv x'\}$. For x and x' to be sufficiently close, we define it necessary that $dist(x, x') \leq t_{upper}$ with $t_{upper} \geq t_{lower}$, where t_{upper} is an upper bound on $\{dist(x, x') : x, x' \in X \times X \wedge dist(x, x') \leq t_{lower}\}$. So all distances considered sufficient ($\leq t_{lower}$) to indicate equivalence of pairs (x, x') are lower than an upper bound t_{upper}. There are probably pairs with distances greater than t_{lower} and lower than t_{upper} that are also equivalent. Only if we find a *tight upper bound* or supremum t'_{upper}, can we say that all equivalent pairs have a distance below t'_{upper} (perfect precision) and all non-equivalent pairs have a distance of more than t'_{upper} (perfect recall). As in most binary classification scenarios, it can be assumed that such a bound does either not exist or constitutes an unpractical and over-fitted decision boundary. Essentially, the goal of blocking is to find an upper bound that is *as tight as possible* (i.e. it is *not* tight, but close to this state) while being reasonably cheap to find. In other words, we are balancing tightness of a bound with the cost of finding it. Informally, we can assume that the more relaxed the bound (or necessary condition for equivalence) is, the easier it is to obtain. For example with a distance function that is known to always return values above what is considered sufficient for equivalence if surname is different, categorizing by surname is a cheap way to obtain an upper bound. Here having the same surname is a necessary condition for equivalence.

It is important to remember that we cannot obtain the bound from looking at all pairs of mentions as there are n^2 pairs for n mentions. Therefore, in a continuous space, it seems desirable to introduce a blocking by partitioning the space such that no pair closer than t_{upper} is in different partitions. However, there might be just one partition if there is no proper separation with respect to our bound t_{upper}. This is the same as connecting all points that are closer than or equal t_{upper}. Then, a *k-nearest-neighbor* classifier with $k = 1$ can be used to map new points to the partition that they are closest to. The space partitioning would then be the *Voronoi-cells* of the “training”-data. However such a new point y could actually be close enough to two points x, x' from different partitions ($dist(y, x) \leq t_{upper} \wedge dist(y, x') \leq t_{upper}$) and thereby connect these or it could be far enough from any point ($dist(y, x) \geq t_{upper} \forall_x \in X$) so that it would make for a new partition. In any case, some data is required to obtain the partitioning and the *kNN*-classifier is not cheap either. If it is possible to define the partitions in terms of limits on the space's dimensions, we can order each new point into the partitioning without any comparison to other points by simply looking at its feature vector. This amounts to a *range-tree* or *k-d-tree* data structure for the space. Still, the ranges would have to be determined based on some existing data. We can use Boolean forms like

$$(d_1 = 2 \vee d_1 = 3) \wedge (d_2 = 1 \vee d_2 = 2)$$

to define areas in this space that contain possibly equivalent mentions. How exactly these areas are found is not specified at this point (consider *DNF/CNF-learning* [24–28]), but all mentions can be sorted into their area or category in linear time by using the Boolean forms as a pointer to the respective category. Many works in clustering define bounds in continuous spaces by nearest neighbor search [29–31], meaning that for each mention we want to find its nearest neighbors in the clustering space (obviously without enumerating all pairs). Means for efficient nearest neighbor search usually involve some kind of *space partitioning* algorithm (see above) where the length of the longest line within a partition is an upper bound on the distance between two equivalent mentions. We might search for bounds on all dimensions (or blocking keys) separately - or combinations thereof. In a continuous space of mention pairs, sufficient conditions define a hypersphere inside which all pairs are equivalent so that its diameter is a lower bound on the distance between non-equivalent mentions. This set is fully contained in a larger hypersphere defined by the necessary conditions of equality so that its diameter is an upper bound on the distance between equivalent mentions. An example for a space-partitioning method used in *k-nearest neighbor search* is the *ball tree* or its generalization, the *M-tree*. These algorithms require a distance metric between mentions and usually assume continuous data, but in this work, we assume categorical data.

2.1.10 Bounds in partial vs. total orders

One way to include information about other related mentions into the similarity measure is to prohibit measuring the linear distance between one mention and another (as in the usual distance measures) and only allow taking a path of observed closely related mentions. For example if we have three mentions represented as $\{a\}$, $\{a,b\}$ and $\{a,c\}$, L1 distance between $\{a,b\}$ and $\{a,c\}$ is 1 as they are differing by one feature. However, using the subset partial order, one would have to go via $\{a\}$ to get from $\{a,b\}$ to $\{a,c\}$. If there were no $\{a\}$ then the distance would be infinite (the similarity being zero). Alternatively, we insist that the subset relation holds in only one direction, hence even with $\{a\}$ present, there is no path from $\{a,b\}$ to $\{a,c\}$. Finally, even if $\{a\}$ is present and we allow both directions, we could use edit distance to model that deletion costs extra, so the path length becomes 2. Upper bounds on this distance measure correspond to lower bounds in the subset partial order. This structuring implicitly considers density since in a densely populated area, it is more likely that there exists a (short) path in the subset relation. It is related to a *nearest neighbor graph* (*NNG*) and the *Euclidean minimal spanning tree* (*EMST*) in the case of a Euclidean space. This also structures the search space in a way that can be used to reduce the time complexity of (approximate) nearest neighbor search, where it is important to consider not the lattice of all *possible* subsets but the partial order (that is probably not a (semi-)lattice) over *observed* representations (i.e. the difference between FCA and ARL). As will be shown later, paths can be additionally weighted by a function of the involved edges.

In this work, we use edge weights in the subset/superset partial order as similarities. In the simplest case, these denote the cardinality difference between any two sets that are in the subset relation. In a more advanced setting, we use the observation frequencies of the representations in a measure similar to *confidence* in association rule learning. For the first case, we can define the dissimilarity between two sets a, b as

$$d(a,b) = |(a \cup b) \setminus (a \cap b)|$$

which satisfies the requirements for a distance metric for pairs of representations that are in the subset relation:

1. symmetry: $d(a,b) = d(b,a)$
2. non-negativeness: $d(a,b) \geq 0 \wedge d(a,a) = 0$
3. positiveness: $d(a,b) = 0 \rightarrow a = b$
4. triangle inequality: $d(a,b) \leq d(a,c) + d(c,b)$

Specifically if a, b are not in the subset relation, then there is a supremum $c = a \cup b$ s.t. $d(a,b) = d(a,c) + d(c,b)$. Like in any distance metric, the triangle inequality amounts to $d(a,b) = d(a,c) + d(c,b)$ if c lies on the shortest path between a and b. In a Euclidean space, this amounts to the line between a and b. In a partially ordered space, this is the case if $a \subseteq c \subseteq b \vee a \supseteq c \supseteq b$. In the Euclidean space $|\{c \in U : d(a, c \leq d(a,b) \geq d(c,b))\}| = \infty$, but in the subset partial order there are only a finite number of c, in practical examples often very few. For example between $a = \{1,2,3\}$ and $b = \{1,2,3,4,5\}$ is only $c = \{1,2,3,4\}$. Therefore, the likelihood of observing representations that are exactly between two other representations is quite high and allows to structure the search space in a way that we can describe the distances between the representations by a subset of $\mathcal{R} \times \mathcal{R}$ – in this example a, b, c by only two numbers $d(a,c), d(c,b)$. Since we are only interested in the most similar representations and we know that by the triangle inequality the indirect paths are longer than the ones for which we have explicit values, we do not need to consider the first ones.

For the second case, we define the *observation count* $\#(a)$ as the number of times a representation a is observed and the *carry count* $\breve{\#}(a)$ as the number of times the representation *a or a subset of it* is observed, i.e. $\breve{\#}(a) = \sum_{b \supseteq a} \#(b)$. Then we define the factor

$$d(a,b) = \frac{\breve{\#}(b)}{\breve{\#}(a)}$$

Like the subset relation, these distances are not symmetric and only defined in one direction. However, as they are ultimately used to merge neighboring nodes, the effect is symmetric, and so we can create the symmetric closure by simply using the same similarity in the opposite direction so that here $d(b,a) = \frac{\breve{\#}(b)}{\breve{\#}(a)}$. Using the common practice to replace $\prod d(a,b)$ by $-\sum \log d(a,b)$, these are also non-negative ($\frac{\breve{\#}(a)}{\breve{\#}(a)} = 1$), positive and satisfy the triangle inequality.

In addition, we define

$$s(a) = \frac{\#(a)}{\breve{\#}(a)}$$

to multiply with our distance metric factor $d(a,b)$ and obtain a probability:

$$p(b|a) = d(a,b) \cdot s(b) = \frac{\#(b)}{\breve{\#}(a)} \Rightarrow s(a) = p(a|a)$$

so that $1 \geq p(a|a) = 1 \cdot s(a)$ and $\sum_b p(b|a) = 1$. There may be b such that $p(b|a) = 0$ (not reachable). This can be read as the probability that an observed representation a actually has the true form b, only that some information is missing. The other way around is not possible ($p(a|b) > 0 \Rightarrow p(a|b) = 0$) as we do not assume that information has been erroneously added.

In addition, we define a different conditional probability distribution that uses the probability mass attributed to “indirect” paths in $p(a|b)$ to only “direct” paths and allows to factorize the indirect paths:

$$\hat{p}(b|a) = \begin{Bmatrix} s(a), & \text{if } a = b \\ d(a,b), & \text{if } b \subset a \land \neg\exists c : b \subset c \subset a \\ 0, & \text{otherwise} \end{Bmatrix}$$

Again, $1 \geq \hat{p}(a|a)$ and $\sum_b \hat{p}(b|a) = 1$. If $b \subset c \subset a$, then $p(b|a) = \hat{p}(c|a) \cdot \hat{p}(b|c)$:

$$p(b|a) = \frac{\cancel{\breve{\#}(R_2)}}{\breve{\#}(R_1)} \cdot \frac{}{\cancel{\breve{\#}(R_2)}} \cdot \ldots \cdot \frac{\cancel{\breve{\#}(R_{n-1})}}{} \cdot \frac{\breve{\#}(R_n)}{\cancel{\breve{\#}(R_{n-1})}}$$

Finally, we say the probability that two representations a, b are equivalent is

$$p(a \equiv b) = \sum_{sup \in sups(a) \cap sups(b)} p(sup|a) \cdot p(sup|b) = p(\equiv|a,b) \Rightarrow p(\not\equiv|a,b) = 1 - p(\equiv|a,b)$$

where $sups(a)$ denotes the (possibly empty) set of suprema $sup \subseteq \mathcal{R}$ of a and the conditionals $p(sup|a), p(sup|b)$ are computed as defined above. Regardless of these exact probabilities, the distances $-\log d(a,b)$ or similarities $d(a,b)$ always give the closest pairs of representations.

$$d(a,b) \geq d(a,c) \Rightarrow p(b \equiv a) \geq p(c \equiv a)$$

$$\max(p(a|b), p(b|a)) \geq \max(p(a|c), p(c|a)) \Rightarrow p(b \equiv a) \geq p(c \equiv a)$$

$$\max(\hat{p}(a|b), \hat{p}(b|a)) \geq \max(\hat{p}(a|c), \hat{p}(c|a)) \Rightarrow p(b \equiv a) \geq p(c \equiv a)$$

Therefore we can use the transitive reduction over the subset partial order to obtain a minimal set of values required for deriving all other distances or probabilities. This trick is at the core of our proposed approach.

2.1.11 More correspondences

It seems obvious that both the concept of nearest neighbor search with space partitioning techniques like the ball-tree and the concept of rough sets with its concept of upper- and lower approximations can be formalized in the context of mathematical topology by finding correspondences to *neighborhood*, *open-* and *closed sets*, etc. In particular, as there are also *partially ordered spaces* in topology, it should be possible to embed the subset partial order of mention representations into a partially ordered space with the appropriate edit distance and/or the edge weights described later being used to define neighborhood and/or distance. Describing the task of entity resolution as a topological problem might allow to use insights from topology and its application domains specifically for the advancement of ER. Unfortunately, due to the required background in topology, we cannot follow this path further in this work.

In our work, we are concerned with at least two levels of equivalence: (1) representation equivalence, (2) mention equivalence and possibly also (3) feature equivalence. In addition, in the context of progressive blocking and agglomerative clustering, the extend of equivalence classes on each of these levels is iteratively increased. This makes it possible to view the problem – or rather its solution – as an instance of what is known as *granular computing*, i.e. any means that looks at data on multiple levels of resolution to select only the information granularity that is necessary and sufficient to represent or extract the knowledge in focus.

In *constraint clustering*, one extends normal clustering by *must-* and *cannot-link constraints*, where must-links correspond to mention pairs that meet sufficient conditions for equivalence and cannot-links correspond to mention pairs that do *not* meet necessary conditions. This means that must-links are expected to link only mentions in the lower approximation of one cluster when using the rough set terminology. Likewise, cannot-links would be useful if they refer to pairs in the upper approximation of a cluster, telling us that we might want to consider further attributes to distinguish them as unrelated. However, constraint clustering is very expensive [32] and therefore does not provide a solution for efficient entity resolution.

In the rough set framework, we can define our target set E by Boolean combination of attributes in *disjunctive normal form (DNF)*, e.g. of $a, b, c, d, e \in \mathcal{A}$ like so:

$$(a \wedge b \wedge c) \vee (d \wedge e)$$

Then E is the union of the *upper set* of the attribute set $A_1 = \{a, b, c\}$ and the upper set of $A_2 = \{d, e\}$. However E is not definable by a single set of attributes, so that the upper approximation is $A_1 \cap A_2 = \{\}$ and the lower approximation is $A_1 \cup A_2 = \{a, b, c, d, e\}$, that is all the mentions that share the values in the respective attributes. If the Boolean formula is given in *conjunctive normal form (CNF)*, it can be transformed into DNF based on the rules of logical equivalences. Even though target sets (e.g. blocks) described by such Boolean formulas cannot be defined exactly by a single set of attributes (e.g. an entity representation), methods have been proposed to learn CNF or DNF formulas to describe target classes or clusters [24–28]).

Rough sets, FCA and ARL are strongly connected. The set of "definable" attribute subsets is a lattice over the power set of attributes that is closed under union, intersection and complement. Then the lower approximation of a target set is the greatest lower bound and the upper approximation is the least upper bound.

2.2 Proposed formalized solution

In the following we summarize and preview the findings of our individual studies in a formalized framework for integrated blocking and (progressive) block-processing. We have identified the following main components that are arranged in a pipeline and all work on the same overarching structure defined by the subset partial order over mention representations:

1. **Representation**: create sets of features to represent the different entity mentions
2. **Specification**: add additional features to representations that are determined overly general
3. **Generalization / Interpolation**: add unobserved generalizations of observed representations to increase connectivity where desired.
4. **Separation**: discover independent components of the representation space to reduce the complexity of downstream tasks.
5. **Collocation**: order the representations of each independent component by the subset partial order.
6. **Progressive Merging / Conflation**: merge blocks represented by adjacent blocks and create a graph minor through edge contraction.
7. **(Within-Block) Clustering / Verification**: cluster any blocks that have not been clustered yet, including new ones created by merges.

This framework constitutes the main contribution of this thesis as it addresses the requirements from Chapter 1.2 and thereby answers the research question of how a single conceptually sound model can satisfy them.

Figure 2.1 abstracts from the different steps listed above. The weakly connected components in the subset partial order create a partitioning equivalence relation (super-blocks) that is used as a necessary condition for equivalence. Within each super-block, we build the subset relation, which is antisymmetric. When contracting edges, we add the reverse to the relation ("selective symmetry"). The result is considered sufficient to determine which representations are in principle reachable from each other (can be merged). Reachability is necessary for two representations ending up in the same strongly connected component, which is a result of the additional reversed edges added. The strongly connected components define another equivalence relation. Condensation is performed to merge the respective groups of representations and the corresponding nodes in the graph.

Being in the same representation (after merging) is considered necessary and sufficient for representation equivalence, which in turn is necessary for a high mention similarity. In the clustering step, highly similar mention pairs within each block define a reflexive, symmetric but intransitive relation. The transitive closure implicitly or explicitly adds additional edges so that mention reachability is enhanced to be an equivalence relation. This is in considered necessary and sufficient for mention equivalence, which is what we were looking for in the first place. On the right, the same concepts are visualized with a focus on what is *not* connected in the different steps: (1) two separate weakly connected components; (2) two representations that are not in the subset relation; (3) two representations that are in the same weakly connected component, but not in the same strongly connected component. (4) two mentions that are connected by mention reachability because of the transitive closure, but not similar; (5) two mentions that are in the same strongly connected component (block) but not in the same cluster as they are not reachable via mention-reachability (neither similar nor connected through transitive closure over similarity).

2.2.1 Representation

Each entity mention is represented by a set of features. It is assumed that each feature is an attribute-value pair, however it can be any kind of feature, e.g. character-ngrams. We distinguish blocking features and clustering features. When we refer to a mention's *representation*, we mean its blocking features. The clustering features can be a superset of the blocking features or completely different. An entity mention's representation is obtained from whichever information is available for the mention, for example by parsing an author name or affiliation string into labelled parts or by extracting features such as character n-grams from the title field of a publication record. Multiple mentions can have the same representation. Representation similarity is assumed to correlate with coreference likelihood, which is why mentions should be compared based on the similarity of their representations. Mention-pairs with more similar representations should be compared before mention-pairs with less similar representations. Hence, mentions with equal representations will be compared first.

2.2.2 Specification

Sometimes the information available for a mention is insufficient for distinguishing it from other mentions. For example, an author mention might only be specified by a surname without any first name information or an affiliation string is incorrectly parsed as $\{(Univ, Clin)\}$ or a publication record is almost empty. The respective representations

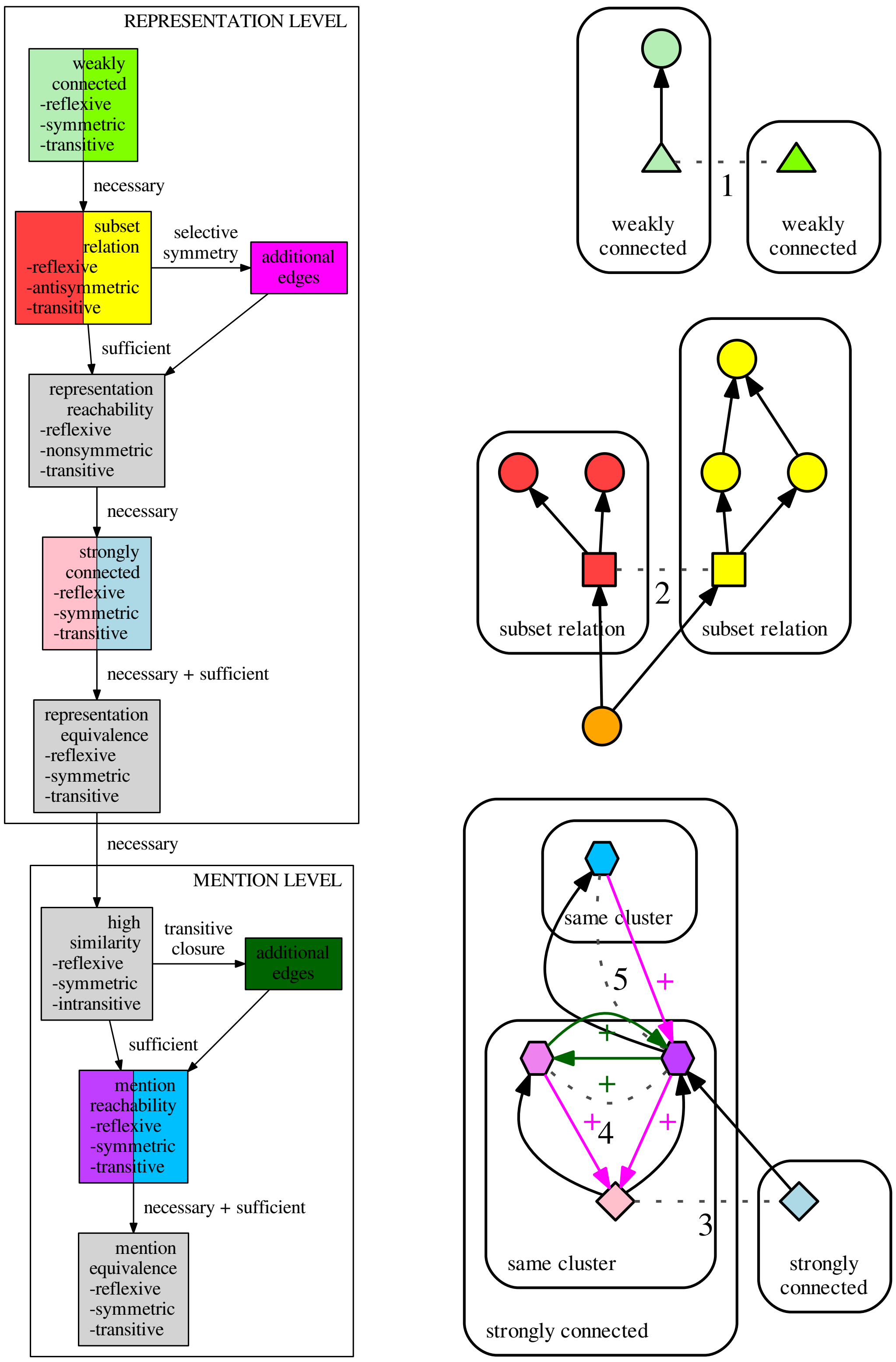

REPRESENTATION LEVEL
weakly connected -reflexive -symmetric -transitive
necessary
subset relation -reflexive -antisymmetric -transitive
selective symmetry
additional edges
sufficient
representation reachability -reflexive -nonsymmetric -transitive
necessary
strongly connected -reflexive -symmetric -transitive
necessary + sufficient
representation equivalence -reflexive -symmetric -transitive
necessary
MENTION LEVEL
high similarity -reflexive -symmetric -intransitive
transitive closure
additional edges
sufficient
mention reachability -reflexive -symmetric -transitive
necessary + sufficient
mention equivalence -reflexive -symmetric -transitive
weakly connected
1
weakly connected
subset relation
2
subset relation
same cluster
5
+
+
+
+
4
+
3
same cluster
strongly connected
strongly connected

FIGURE 2.1: Left: formal pipeline of operations for end-to-end entity resolution including super-blocking, progressive blocking and clustering; Right: Different steps visualized: 1: different weakly connected components, 2: not subset-reachable, 3: different strongly connected components, 4: not similar but connected, 5: different clusters.

are very likely to be subsets of many other different representations and are thereby almost guaranteed to cause large blocks with completely unrelated mentions. We therefore suggest to scan all representations and check if they satisfy some defined minimal requirements of specificity. If they do not, one can simply add the mention's or the representation's identifier as an additional feature to isolate it in the set of all representations and prevent it from connecting a large number of unrelated representations and mentions.

2.2.3 Generalization

In order to add additional connectivity, we can hypothesize unobserved representations by generalizing observed ones. This can also be referred to as *interpolation*, especially if the resulting representations are targeted to be "in between" observed representations. Generalizing a representation simply means selectively creating subsets of a representation. These can be added to the observed representations. For example if we see *John Doe*, we can hypothesize an unobserved representation *J. Doe*. Note that *J. Doe* is represented by a subset of *John Doe* as the *J* in *John* is part of the latter's representation. If we also see *J. Doe* explicitly mentioned in the data, nothing will change as the hypothesized observation frequency was 0 and – after actual observation – is the same as if not hypothesized ($n + 0 = n$). However, if we do not observe *J. Doe*, then the hypothesized generalization can connect matching representations like *John Doe* and *Jhon Doe* or *Johnathan Doe* – but also contradictory ones like *Jack Doe*. If a hypothesized generalization does not add any connectivity, for example because *John Doe* is the only *J. Doe*, then the edge between them will have a weight of 1 and will be contracted right at the beginning (see below).

2.2.4 Separation

In very large datasets, we aim at completely separating representations that almost certainly refer to unrelated mentions. Generally speaking, mentions with contradictory representations need not be compared. However they might still be inseparable due to another representation being not contradictory to both. For example *J. Doe* is related in the subset partial order to *John Doe* and *Jack Doe*. Since we do not know if a mention referenced as *J.* refers to *John* or *Jack* we have to compare it to both and the three cannot be separated. No two representations related by the subset partial order are assumed contradictory, i.e. any legal representation that adds features to another representation is not contradictory to it. It is however possible that two representations are not in the subset partial order, but are also not contradictory (e.g. *John H. Doe* and *J. Herbert*

Doe). For this reason, generalizations like *J. H. Doe* are important to indirectly connect them. This notion of contradiction imposes some degree of freedom and responsibility to the representation generation. If we say that a person can have only one birth date, then we must not create a representation with an additional date as that could be a superset to any representation with only one date. Any legal representation with a different birth date is unrelated by the subset partial order – although not necessarily by its symmetric closure. The symmetric closure relates all representations that are somehow "connected" by the subset partial order or its inverse, the superset partial order. If $a = \{(1st, John), (sur, Doe), (born, 1972)\}$ and $b = \{(1st, John), (sur, Doe), (born, 1972)\}$ are supersets of $c = \{(1st, John), (sur, Doe), (born, 1972)\}$, then they are not related by the subset nor by the superset partial order, but by their union. The symmetric closure of the subset partial order returns connected components that include all pairs that are not contradictory as well as pairs that are contradictory but somehow related by non-contradictory representations. It is futile to compare pairs across connected components. Therefore, we obtain what we call super-blocks as weakly connected components on the DAG imposed by the subset partial order or its transitive reduction. We show later how these components can be discovered efficiently and in parallel for up to one billion representations. All further processing only happens within connected components.

2.2.5 Collocation

For each connected component, we build the subset partial order over the respective representations. In fact, it is sufficient to build only its transitive reduction. This gives a DAG with one node per representation and edges implementing the subset partial order over these representations – or its reduction. At this point, we already know that any node b reachable from node a via some path p is more or equally similar than a node reachable by an extension of p. In addition, we generate weighted edges by assigning each representations/node two counts: (a) the observation-count giving the number of times the representation has been observed exactly and (b) the carry-count giving the number of times this representation or a superset has been observed. As described in Chapter 2.1.10 these are used to compute edge-weights between adjacent representations to find the representations that are most likely to be equivalent.

2.2.6 Conflation

We assume that all mentions with the same representation / under the same node are always compared. To progressively compare more mentions, nodes are merged by edge

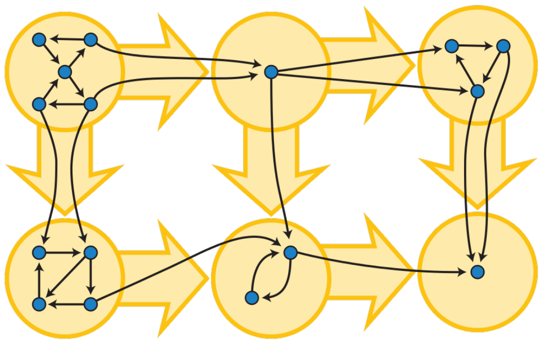


FIGURE 2.2: Visualization by David Eppstein: "The yellow directed acyclic graph is the condensation of the blue directed graph. It is formed by contracting each strongly connected component of the blue graph into a single yellow vertex.", https://commons.wikimedia.org/wiki/File:Graph_Condensation.svg, retrieved 6th March 2023.

contraction. As a consequence, mentions of directly connected nodes are combined under the same node and hence compared. Edge contraction creates a graph minor and is guided by edge weights. The higher the edge weight, the higher the chance of being contracted. For iterative merging, many options are possible, as is shown later. However, the one most true to the probabilistic interpretation of the edge weights is the following. During each iteration, lower the merge-threshold t, starting at a threshold of 1. Compare all mentions in the same node. For each edge in the DAG draw a number r from a uniform distribution over [0,1]. If $r > t$, note this edge as "activated". A simpler way is to activate all edges that are above the threshold, but this has the problem that for example the probability of a path $a \overset{0.5}{\to} b \overset{0.5}{\to} c$ is contracted at $t = 0.5$ and not at $t = 0.25$ as we essentially take the minimum over the edge weights and not the product as intended by the probabilistic interpretation. For each activated edge $a \to b$ we can add a reversed edge $b \to a$, compute the *strongly connected components* of the new graph and merge all nodes inside the strongly connected components, creating its *condensation* for further iterations (see Figure 2.2). The observation- and carry-counts must be updated after each iteration so that they reflect the new numbers of mentions in each node. For visualization purposes, each node can be described by the union of the merged representations. However the superset partial order is *not* recomputed based on those unions.

2.2.7 Clustering

Pairwise comparison within the growing representation nodes can be implemented by any clustering method. One might want to keep the similarities or even the togetherness information previously computed for mention pairs that were already in the same node during the previous iteration, however the additional memory requirements should be considered. This describes essentially the entire framework with leaving open a number of aspects for how to exactly obtain the order of conflation of the graph.

Chapter 3

Related Work

This chapter is meant to complement the related work sections of our individual publications. Therefore, we focus here on higher level problems of entity resolution and how they are addressed in the literature. In that sense, Sections 3.1–3.6 are referring back to the problem exploration in Chapter 2.1. In addition, in Section 3.7, we include a comprehensive interpretation of the most common blocking approaches by thoroughly analysing a recent survey article by Papadakis et al. [1]. In this context, we derive a generalized matrix-based view on blocking in which we can describe most of the reviewed blocking methods as well as our own approach. Finally, in Section 3.8, we touch upon the issue of clustering by deriving a generalized model for different clustering methods, in which blocking is merely a special case as well as summarizing in a schematic way the different options from the literature for integration and communication between the blocking and the clustering step.

3.1 Transitivity in ER

The notion of transitivity is very central to the task of entity resolution for a number of reasons. There are at least four major aspects related to it:

1. Transitivity as a target assumption
2. The incompatibility of intransitive matching and transitive equivalence
3. Transitivity as a threat (error propagation)
4. Transitivity as an opportunity (avoiding redundant comparisons)

3.1.1 Transitivity as a target assumption

Ultimately, if we aim at disambiguating entity references, we assume a target equivalence relation (reflexive, symmetric, transitive) where each equivalence class corresponds to a separate real world entity and all mentions in that equivalence class are references thereof. As mentioned earlier, most clustering methods incorporate this target assumption to the extend that they are unable to return non-equivalence output relations. Clustering or blocking methods that output overlapping clusters could be modified by duplicating all elements in the overlaps to obtain an equivalence relation. Otherwise all overlapping clusters/blocks would be connected in this case. Other methods that only output pairwise equivalence decisions must be followed by the application of the transitive closure to obtain the final equivalence relation, a straight-forward rationale pointed out by Papenbrock et al. [15] and many others.

3.1.2 Intransitive matching vs. transitive equivalence

Independent of how it is implemented, *matching* of two mention representations can be considered a necessary condition for equivalence (any two equivalent mentions must have matching representations / any two mentions that have contradictory representations cannot be equivalent). Matching is however not sufficient for equivalence because of potential homonymy. Therefore, the equivalence relation is a subset of the matching relation. As the equivalence relation partitions the set of all mentions into disjoint equivalence classes, it can be described not only as a set of pairs, but as a set of mention sets, which is theoretically and computationally convenient, e.g. in connected component labelling, because otherwise the time complexity is quadratic just because the output is a square matrix. It is tempting to assume that the matching relation can also be described as a set of matching-based equivalence classes that are supersets of / can be further partitioned into the equivalence-relation's equivalence classes. However, despite contrary statements in Monge and Elkan [33], matching is *not* transitive [34]. Consider *J. Doe*, which matches with *John Doe* and *Jack Doe* although the latter two are contradictory. The consequence is that mentions that are represented by *J. Doe* could refer to the same entity as mentions in *John Doe* or in *Jack Doe*. But if we connect all mentions in *J. Doe* with all those in *John Doe* and *Jack Doe* because they are matching, then the transitive closure will also connect mentions that have been clearly specified to refer to different entities (i.e. those that have been described as *John* and those described *Jack*). In other words, creating disjoint blocks for further comparison, we are forced to combine mentions that we know to be unrelated – the reason being missing information, e.g. regarding the first name in *J. Doe*. Basically, some mentions in *J. Doe* could perhaps be moved into *John Doe*, some into *Jack Doe*, and some might remain in *J. Doe* as they do not co-refer with any of the mentions described by the two more specific representations. But this information is not available at the matching level. Obviously this makes it challenging to use all the information available on the matching level as a precursor (blocking) to more detailed inspection (clustering). As the same holds for similarity measures [34], simple bounds on similarity have the same deficiencies. This unfortunate incompatibility has been directly and indirectly addressed in a number of works. In blocking for author disambiguation, the problem is often circumvented by choosing *surname, first initial* as blocking keys, i.e. the minimal information that can be expected to present in any author reference. It is usually not mentions what happens if for example the first initial is not present. It can be assumed that those mentions are silently ignored. In other cases, *surname, all initials* is used, suggesting that for example *J. Doe* ends up in a different block than *J. H. Doe*, which is not very intuitive. This has been addressed to some extend in [35] using a basic rule. [34] present a more formal approach, although they claim for their matching functions that "merging two records r_1 and r_2 cannot create evidence (in the merged record r_3) that would prevent r_3 from matching any other record that would have matched r_1 or r_2.". What if r_1 is *J. Doe*, r_2 is *Jack Doe* and "any other record" is *John Doe* which matches r_1 but not r_3 (*J. Doe*+*Jack Doe*=*Jack Doe*)? The solution is to indirectly apply the transitive closure on the intransitive matching relation by representing the result of two merged records by the union of their representations, e.g. $\{\text{J. Doe}\} + \{\text{Jack Doe}\} = \{\text{J. Doe}, \text{Jack Doe}\}$. They require only one of the elements in the representations to match, so that $\{\text{J. Doe}, \text{Jack Doe}\}$ matches $\{\text{John Doe}\}$. This means that the values are not AND-, but OR-connected (Disjunctive normal form), amounting to single-link clustering or basic blocking (anything that shares or matches in one value is connected). Their iterative merging process is actually the same as our progressive merging, although we separate the intension (initial node representation) from the extension (mentions in a node). The same approach is also taken in [36]. Regardless, it remains the problem that mentions with clearly contradictory representations are connected in order to connect matching ones (sacrificing blocking precision for recall against better judgement).

3.1.3 Transitivity as a threat (error propagation)

Many entity resolution methods are described in terms of a pairwise classifier that does not guarantee by itself the transitive property of the equivalence relation [37]. Although it is sometimes neglected, the pairwise classification must always be combined with the transitive closure to obtain the true equivalence assumptions inherent in its local decisions [38]. One can assume in principle that a pairwise classifier that is agnostic regarding the global consequences of its decisions performs worse than a clustering method that acknowledges this at least to some extend by considering multiple data points in order to connect those – or other points. For example in agglomerative clustering [39], two clusters are merged considering in one way or the other the proximity of all of their points, or in density-based clustering [40] surrounding points are summarized into a density measure that acts as a prerequisite for connecting two points. In other words, on the clustering level, a pairwise classifier might return a good precision and recall on detecting pairwise equivalence, but only a few false positives can lead to enormous drops in precision when the transitive closure is applied because suddenly all kinds of unrelated mentions are connected via these “narrow bridges” [14]. Therefore, transitivity is the source of a great threat of error propagation.

3.1.4 Transitivity as an opportunity (avoiding redundant comparisons)

Being optimistic regarding the threat of error propagation, one can also use the transitivity of the equivalence relation to one’s advantage – i.e. to save unnecessary comparisons. If we know that $a \equiv b$ and $b \equiv c$, it is not necessary to compare a and c (note however that from a quality-management point of view it does not hurt to verify $a \equiv c$, detect potential prior errors and avoid their propagation). Although some address this opportunity more directly (e.g. Hernández and Stolfo [41]), all blocking methods exploit it. For example, they might create an artificial data point x for each blocking key and link mentions a, b, c to it, thereby determining $a \equiv b$, $b \equiv c$ and $a \equiv c$ without direct comparison. Monge and Elkan [33] interpret entity resolution as “determining the connected components of an undirected graph”, computing the transitive closure of matching decisions to avoid comparing what is known to belong together rather than what is known not to. Note that this does not address the major ER complexity problem as most potential for omitting comparisons comes from ignoring pairs that are very unlikely to co-refer rather than ignoring those that are known to co-refer – simply because most pairs do *not* co-refer. In the following we will summarize the work done by Firmani et al. [36] as it precisely addresses the saving of redundant comparisons in due consideration of the transitive closure and constitutes a powerful approach to entity resolution in general. For any relation that satisfies the assumed necessary conditions for equality (the default being all pairs), one can compute a (minimum) spanning tree or transitive reduction giving a minimal set of coreference claims to be verified. For example if we assume all n pairs in a supposed equivalence relation $X \times X$ and assume that all of them can be verified if they are correct, then a simple total order can be created where each mention x is connected to at most two other mentions. We can verify the entire equivalence relation by verifying $n - 1$ supposed co-reference pairs. In reality, of course we do not know the true equivalence relation but we want to find it. Still, assuming a perfect verification oracle, two mentions cannot be connected indirectly (via transitive closure) if they are not connected directly. A process that amounts to single-link agglomerative clustering can therefore be applied, starting with singleton clusters and connecting clusters if any two of the respective members are connected. A single pair suffices to determine whether two clusters belong together – or not. In the following Section 3.8, we present a theoretical clustering and blocking framework that also relies on this notion and shows that most such methods can be described in terms of connections in a bipartite graph and subsequent transitive closure computation or connected component search. It is even more important to avoid comparing pairs that are already known to co-refer by transitive closure if the oracle is implemented as manual labor. Therefore, the issue is addressed in the context of crowd-sourced entity resolution, like in Wang et al. [42].

3.2 Partial Orders, Lattices and DAGs for ER

The partial order (of entity mention representations) plays an important role in our work. A number of other approaches to ER have also made use of partial orders to formalize important relationships. Benjelloun et al. [34] assume a partial order of records and allow to compute "record domination" by the subset/superset relation. "Instance domination" is built upon record domination by adding additional symmetric matching relationships. It therefore instantiates a preorder. This has the same effect as merging neighboring nodes in a blocking graph based on the record domination partial order, e.g. by computing strongly connected components in the directed (but not acyclic) graph that results from adding backwards edges for adjacent node pairs that we want to merge. This is exactly what we propose in our framework, however here it is applied on entity mentions, not on blocks of them. Kenig and Gal [43] do not directly describe a partial order of representations, but represent mentions by sets of features and deploy frequent itemset mining on them. Frequent itemset mining assumes a lattice over observed and unobserved feature combinations, which amounts to a general partial order if unobserved itemsets are dropped. In their work, the support of discovered maximum frequent itemsets is taken for a block if its size is not too large and some cluster-coherence measure is satisfied. Otherwise, more specific representations might be chosen as blocks in later iterations with a reduced minimum support size. It seems that similar to sparse representations in author disambiguation (e.g. only surname present), mentions with very general representations will thus be ignored. Finally, Chai et al. [44] build a partial order not on representations or mentions, but on mention pairs to avoid asking crowd-sourced oracle questions for mention pairs that can be determined coreferent by applying the transitive closure one previous decisions (see also Firmani et al. [36]). A pair of mentions is represented by a vector with one dimension for each attribute and the respective value being the similarity of the two involved mentions' feature for this particular attribute. Domination on pairs is defined as having a similarity value higher or equal in every single attribute. It is not trivial to imagine what such a partial order looks like in practice and unfortunately no instantiated examples are given. So despite this work being very comprehensive and potentially relevant to the ER task in general, it is not presented in an accessible way, making it hard to leverage the insights.

3.3 Formal Concept Analysis, Association Rule Learning and Rough Set Theory

As stated in the introduction, the research fields of Formal Concept Analysis (FCA), Association Rule Learning (ARL) and Rough Sets are tightly connected both mathematically and by overlapping literature (or rather FCA is tied to both ARL [45–48] and Rough Sets [49–52]). As described earlier, they are also strongly related to our blocking approach and to general entity resolution problems. Despite this, we can find only few works on entity resolution that explicitly reference FCA, ARL or Rough Sets. A reason for this might be that each of the three research fields traditionally has their own, focused application scenario. FCA is most used for ontology induction or concept learning, ARL for shopping basket analysis (i.e. recommending products) and Rough Sets in a classification rather than clustering context.

FCA is related to ER in that the subset/superset relation is used to build a lattice of feature sets representing entities. Traditionally, each node in the lattice corresponds to a (sub-)concept of an entity, while in our interpretation it corresponds to an entity's representation. There are two major reasons that make it problematic to use FCA for ER. The first is that mentions of the same entity are usually expected to be variations with similar granularity that are therefore expected to be arranged rather next to each other than in a hierarchical relation. The second is that FCA usually assumes a lattice that also contains unobserved representations obtained from instantiating all possible feature combinations (i.e. the power set over the feature vocabulary) as nodes, which is computationally infeasible in ER. However, we find that in many cases there exist at least one entity mention whose set of blocking keys constitutes a generalization or subset of multiple related mentions with similar granularity. For example you are likely to find a *J. Doe* reference when you have *John Doe* or *Jonathan Doe*. Therefore it is reasonable to build a partial order not over all possible combinations but only on the observed ones. Furthermore, as long as the number is independent of the collection size, we can generate any number of unobserved generalizations or subsets of observed representations to tie together different variations of entity references. For example if we see *Jon Doe*, we can hypothesize that this can also be referenced as *John Doe* by representing *Jon Doe* as $\{\text{Jon}, \text{John}, \text{Doe}\}$ and hypothesizing a normalized representation $\{\text{John}, \text{Doe}\}$ that ties it with another potential reference $\{\text{John}, \text{Jonathan}, \text{Doe}\}$. The great advantage of the superset relation that underlies this approach is that it allows a fast nearest neighbor search, because a superset can be found by taking the intersection $\bigcap_{f \in R(x)} Я(f)$ of all mentions $Я(f)$ with a feature f present in a representation $R(x)$, for which there are efficient solutions from Boolean retrieval.

ARL adds useful frequency information to the concept lattice or its observed partial order subset. The *support* of a representation is the sum of the observation count of all representations that are the same or a specification thereof. It can be used to obtain edge weights indicating the association strength (or coreference likelihood) between two representations, for example by the *confidence* measure. This lends itself well to progressive resolution methods where the most promising associations are exploited first to merge tightly connected representations and resolve their mentions in a verification step. The reasons why ARL is not commonly used for ER are probably the same as those for FCA. However, Kenig and Gal [43] use the concept of *maximum frequent itemsets* that comes from ARL to obtain a complicated blocking algorithm.

Regarding the relationship between rough sets and entity resolution, some works by K.A.Vidhya et al. [53–55] deploy “rough set attribute-based unsupervised hierarchical blocking” to resolve web-based entities in a schema-agnostic way. This approach is comprehensible because rough set theory is concerned with the search for appropriate equivalence relations based on a selection of attributes and essentially formalizes necessary and sufficient conditions for equality. We have explained this in some detail in Chapter 2.

3.4 Similarity Search

If equivalence is defined as high-enough similarity, then the task of determining and exploiting bounds on equivalence likelihood can be interpreted as *similarity search*, *nearest neighbor search* or *similarity joins*. These methods are not always exact and are described for the ER context in Papadakis et al. [1] under the term *Filtering*. The general relationships are as follows. Similarity search [56–59] denotes all tasks that can be related to finding similar objects in a large collection, retrieving the similarity of two objects or sorting pairs of objects by similarity. Nearest neighbor search is a special case of similarity search, where the task is either to find the k nearest neighbors for a given object or to find all objects that are at most ϵ away from a given object. Similarity join [42, 60–63] refers to a database operation that generalizes the join operator to a similarity threshold operator returning all pairs of rows that differ at most ϵ in a certain column. *MinHash* [64] is an efficient way of approximating Jaccard similarity of two sets as the probability of collision over a number of simple hash functions. *Locality Sensitive Hashing* (*LSH*) [57, 65, 66] often uses MinHash to perform a dimensionality reduction from all features of a collection to a fixed number of individual hash values. Identical objects should have identical hash values and somewhat similar objects should have at least one hash value in common. LSH is not a nearest neighbor method but can be used to implement it. Essentially, LSH is a blocking method if we interpret the hash values of an objects as its blocking keys. Therefore it is popular as an end-to-end ER method [67–71]. In LSH, blocks are overlapping. The sparse dot product of the objects$\times$keys matrix with its own transpose returns all the pairs that share at least one blocking key, so only somewhat similar pairs are returned. If these are too many, one might adapt the hash function to produce fewer collisions. Other approaches to nearest neighbor search are for example based on space partitioning (e.g. *k-d-trees* [72], *R-trees* [73], *R*-trees* [74] or *metric trees* [75]). The subset partial order corresponds to a neighborhood in the Hamming space over all possible representations (binary strings of length $|F|$) and thereby lends itself to nearest neighbor search for the Hamming distance of representations as these can be implemented as edge hops in the corresponding lattice's graph. As there is an infeasible number of $2^{|F|}$ possible representations, in our work, we only consider a subset of them, i.e. only observed representations and a limited number of hypothesized representations derived from them. This comes at the risk of missing connections, but the larger the number of data points in comparison to the space's dimensionality, the less likely are missed connections. In the subset of observed (and derived) representations ϵ-hops also do not necessarily translate to a Hamming distance of ϵ but to one that is at least ϵ, but while this is a problem for finding ϵ-nearest neighbors, it is not relevant to finding the k nearest neighbors of a given representation. Several exact and approximate solutions for nearest neighbor search in Hamming space have been proposed (e.g. [76–80]), but it is a beyond the scope of this work to determine whether and to which extend they are applicable to ER blocking in practice. In a survey by Papadakis et al. [1] discussed later, a number of similar-purpose methods are presented under the term “Filtering”, in particular “partition-based Filtering”.

3.5 Constrained Clustering

Constrained clustering [81] is a generalization of standard clustering that allows to specify *must-link* or *cannot-link constraints* to be satisfied by the output clusters. Constraints are simply sets of point-pairs that need to be together in the same cluster or must be in different ones. Then standard clustering is a special case where the sets of must-link and cannot-link constraints are both empty. Constrained Clustering is interesting for ER because it formalizes a different view on the matching relation than blocking does. In blocking, we are trying to exploit the matching relation in order to save computational complexity by indirectly excluding contradictory pairs from direct comparison, e.g. by a pre-partitioning the collection. We have already seen that the incompatibility of the matching- and the equivalence relation forces us to accept knowingly contradictory representations within the same block. Constrained clustering enforces clusters that separate contradictory records and group equivalent ones. Cannot-links are the result of unsatisfied necessary conditions for equivalence and must-links are the result of satisfied sufficient conditions (given the limited view defined by the blocking features). Although constrained clustering is opposed to blocking in that it is not less but more expensive than normal similarity-based clustering, it might be relevant to consider its relationship to blocking and clustering. For example, we could consider the clustering step followed by blocking to be constrained by the contradictions that we have had to accept during blocking. In a recent survey, Gançarski et al. [82] summarize that there are additional types of constraints. Besides cannot-links and must-links that are instance-level constraints, cluster-level constraints are also possible. Examples are (a) the number of clusters, (b) an acceptable range for the number of points per cluster, (c) a maximum difference for points in the same cluster, (d) a minimum difference between two clusters (e.g. measured by the closest elements) or (e) a minimum cluster density (e.g. each element must be in a dense region). However, (c) can also be achieved by establishing cannot-links for all sufficiently dissimilar points and (d) through must-links for all sufficiently similar points. Constraint (a) is an unfortunate parameter of many cluster methods anyway and (e) is automatically constraint in density-based clustering methods. Finally (b) is interesting from a blocking point of view, but not so much in a clustering setting as it should be taken care of by (c) so that large clustered that are actually justified are not split up. From our point of view, these additional constraint-types are rather parameters of the clustering methods and add unnecessary conceptional overhead in the context of entity resolution where we are mostly interested in using cannot-link constraints for known contradictions and perhaps some must-links for semi-supervised learning. Without going into great detail, probably the most important aspect regarding constraint clustering in our context is the expected computational cost associated with it in comparison to standard clustering methods (see also [83]). In Davidson and Ravi [32], it is summarized that contraint clustering is intractable if both must-links and cannot-links are considered and if only cannot-links are considered. Obviously, only must-links can be considered easily, for example in agglomerative clustering by using them as initial clusters. A general consideration is that with a good similarity metric, the constraints should be useless, as points that must be linked would be very similar and points that cannot be linked would be very dissimilar. If a constraint violates the similarity notion, a reasonable approach would be to investigate the reasons and adapt the similarity measure so that it better captures the true coreference likelihood. Alternatively, the known constraints could also be used to learn or tune such a similarity measure automatically. This is also acknowledged by Basu and Davidson [84]. In fact, in Davidson and Ravi [32], two works [85, 86] are cited that have observed better results applying a standard clustering with a similarity measure optimized w.r.t. constraints than enforcing the constraints during clustering. However, this only works if the number of clusters specified in the clustering method does not contradict the underlying assumptions in the similarity measure and in the constraints [32]. In other words, if one asks for a clustering that makes no sense, one will get it. If the (modified) similarity measure distinguishes roughly k dense regions, and these also correspond to the constraints, then the clustering method should not be asked to cluster the data into k' clusters, where k' is much different from k. The features used to obtain the constraints (e.g. the blocking features that result in a non-matching) can be included with the clustering features, and even weighted heavily if necessary. Especially if only cannot-links or only must-links are used, one can simply combine the Boolean constraint with the gradual similarity of the remaining features, e.g. by taking the minimum or product with cannot-links (*can-be-linked:False*) or maximum with must links *must-be-linked:True*. If a differentiable function is desired, we are back at the problem of adopting the similarity measure. Although in this work we do not pursue the option of deploying constraint clustering to recognize known within-block matching contradictions, this presents an interesting opportunity for future work.

3.6 CNF and DNF Learning

If some curated coreference data is available, it might be desirable to learn a blocking scheme rather than using only common sense in its design or improving an initial scheme iteratively. The essential properties against which to evaluate and optimize schemes is that it should group coreferent mentions and separate non-coreferent mentions. This can be measured in terms of false-positives and false-negatives and aggregated in various performance functions like precision and recall. Concerning the learned output, blocking schemes are expected to be described in a logical formula, either in disjunctive normal form (DNF) or in conjunctive normal form (CNF). General purpose propositional logic rule learning algorithms can then be applied. A blocking predicate $d(x, y)$ is defined to be a binary Boolean function d that takes two representations x, y and returns True if they satisfy the constraint declared by it, or False if not. In practice, these functions are defined based on a feature-selector (e.g. a certain column or attribute) and a feature-comparator (e.g. equality or similarity). For example *same surname* returns True for two names with the same surname component. Likewise, the predicate can also be deployed in a function $g(x, d)$ that returns for a given x all other representations y such that $d(x, y) = 1$ [28]. Obviously, the pairwise formulation in $d(x, y)$ is useless in a blocking context where the main point is to avoid pairwise comparisons. Therefore, it is advisable to use only scalable feature-comparators that can be determined without exhaustive pairwise comparison [26]. Then, we can use the feature-selector to define a sparse $|X| \times |F|$ matrix ${}^{x}\overset{f}{\boxplus}$ that assigns each representation $x \in X$ its features $f \in F$. In addition, we can define a sparse $|F| \times |F|$ matrix ${}^{f}\overset{f}{\boxplus}$ that encodes the feature relationships, e.g. a diagonal matrix for equality or a matrix with a wider diagonal on sorted features if alphabetical similarity is deployed with a threshold. The sparse dot-product ${}^{x}\overset{f}{\boxplus} \cdot {}^{f}\overset{f}{\boxplus}$ returns a modified sparse $|X| \times |F|$ matrix that could for example be used to read off all representations that have a selected key exactly or a sufficiently similar one as defined by the comparator. Regardless of whether we have generalized ${}^{x}\overset{f}{\boxplus}$ in this way or not, we can then deploy a special operation $\star$ with an $|F| \times |K|$ matrix ${}^{f}\overset{k}{\boxplus}$, where K is the feature conjunctions that are called blocking keys: ${}^{x}\overset{f}{\boxplus} \star {}^{f}\overset{k}{\boxplus} = {}^{x}\overset{k}{\boxplus}$, where ${}^{x}\overset{k}{\boxplus}_{ij} = \prod_{l} {}^{x}\overset{f}{\boxplus}_{il} \geq {}^{f}\overset{k}{\boxplus}_{lj}$, meaning every feature that some x has must appear in the features of the conjunction k for x to be mapped to k. The sum of products (dot product) implements disjunction and our special operation implements conjunction. Each blocking key is considered a conjunction of features. By assigning each representation not only its selected features but also the features that are considered sufficiently similar to them, we can implement the feature-comparator. The dot-product ${}^{x}\overset{k}{\boxplus} \cdot {}^{k}\overset{x}{\boxplus} = {}^{x}\overset{x}{\boxplus}$, where ${}^{k}\overset{x}{\boxplus} = {}^{x}\overset{k}{\boxplus}^{T}$, returns a sparse $|X| \times |X|$ matrix ${}^{x}\overset{x}{\boxplus}$ with a value for all representation pairs that share at least one block. The blocking scheme can be considered disjoint if each representation is only assigned one key in ${}^{x}\overset{k}{\boxplus}$ (and each key processed as a separate block) or if we take the transitive closure and connect all overlapping blocks (which is what the sum in the dot-product does). Note that in this setup, the feature-selector is implemented in ${}^{x}\overset{f}{\boxplus}$ and the feature-comparator in ${}^{f}\overset{f}{\boxplus}$. It is also possible to redefine F and use ${}^{x}\overset{f}{\boxplus}$ strictly for mapping all kinds of associated information to a representation and then encode both the selector and the comparator in ${}^{f}\overset{f}{\boxplus}$, where a non-selected feature corresponds to an empty row (and column). Note also that if the number of distinct selected features is very small, then pairwise feature-comparison could actually be feasible. For this, iterate over all pairs of distinct selected features, check whether their similarity satisfies the comparator threshold and if so, add a value to ${}^{f}\overset{f}{\boxplus}$. If the threshold results in a feature matrix that is sufficiently sparse, the above dot-product should still be feasible. In the literature, there is a distinction made between DNF- [24–27] and CNF [28, 87, 88] blocking. From a theoretical perspective, this distinction is not necessary as every DNF can be converted into a CNF and vise-versa. In some cases, the CNF might be more compact than the DNF. For example, in Kim et al. [28], the following CNF is learned:

$$\binom{\text{same}}{\text{surname}} \wedge \binom{\text{same}}{\text{firstinit}} \wedge \binom{\text{compatible}}{\text{middlename}} \wedge \left(\binom{\text{compatible}}{\text{firstname}} \vee \binom{\text{cosim} > 0.8}{\text{coauthors}}\right)$$

which translates to the following DNF:

$$\left(\binom{\text{same}}{\text{surname}} \wedge \binom{\text{same}}{\text{firstinit}} \wedge \binom{\text{compatible}}{\text{middlename}} \wedge \binom{\text{compatible}}{\text{firstname}}\right)$$

$$\vee \left(\binom{\text{same}}{\text{surname}} \wedge \binom{\text{same}}{\text{firstinit}} \wedge \binom{\text{compatible}}{\text{middlename}} \wedge \binom{\text{cosim} > 0.8}{\text{coauthors}}\right)$$

We prefer the DNF as it corresponds directly to the matrix notation presented above and the individual conjunctions can be interpreted as lower bounds of the respective set of representations in the subset partial order of key-sets. Each conjunction within the disjunction corresponds to a potential key assignment in ${}^{x}\overset{k}{\boxplus}$. In the subset partial order, we can observe non-disjointness as overlaps. For the above example, Kim et al. [28] claim that the formula is disjoint, but we can see that this can only hold if $\binom{\text{compatible}}{\text{firstname}}$ and $\binom{\text{cosim}>0.8}{\text{coauthors}}$ are either mutually exclusive or equivalent/redundant, which might be the case in the training data, but is not guaranteed per se. Let $a = \binom{\text{same}}{\text{surname}} \wedge \binom{\text{same}}{\text{firstinit}} \wedge \binom{\text{compatible}}{\text{middlename}}$ and $b = \binom{\text{compatible}}{\text{firstname}}$ and $c = \binom{\text{cosim}>0.8}{\text{coauthors}}$. Then $a \rightarrow (a \wedge b)$ and $a \rightarrow (a \wedge c)$. Why should there not be a d, s.t. $c \rightarrow (c \wedge d)$ with overlapping DNFs $(a \wedge b) \vee (a \wedge c)$ and $(a \wedge c) \vee (d \wedge c)$? As said above, the easiest way to ensure a disjoint blocking seems to assign each representation only a single key, i.e. to learn only a single conjunction of individual terms. Alternatively, one can also accept the transitive closure to connect overlapping blocks and learn relatively specific DNFs so that blocks do not get too large. After the output format and its (feasible) application to blocking is established, a remaining question is how to learn the DNF or CNF from annotated data. In the literature, a number of general purpose methods are discussed and/or deployed, in particular the *Sequential Covering Algorithm* in [26–28], *Peleg's Greedy Algorithm* in [25] and *ApproxRBSetCover* in [25]. In addition, Whang and Garcia-Molina [87, 88] and Kejriwal and Miranker [24, 26] present custom algorithms for learning blocking schemes.

3.7 Existing Blocking and Filtering Methods

This section is based on a comprehensive recent survey by Papadakis et al. [1] that lists all major blocking and filtering methods until 2020. We build on this survey by referencing and interpreting its taxonomy as well as introducing a straight-forward framework to generalize most of the presented methods in terms of sparse matrix multiplication.

In the recent survey [1], the following high-level distinction of blocking methods (or rather methods used in the context of blocking) is made[1]:

1. Block Building
   (a) Hash-based Blocking
   (b) Sort-Based Blocking
   (c) Hybrid (Hash+Sort) Blocking
   (d) Learned Blocking
   (e) Schema-Agnostic Blocking
2. Block Processing
   (a) Static Block Cleaning
   (b) Dynamic Block Cleaning
   (c) Comparison Cleaning
3. Filtering
   (a) Basic Filtering
   (b) Prefix-Filtering
   (c) Partition-based Filtering
   (d) Tree-Based Filtering
   (e) Approximate Filtering
   (f) Fuzzy Matching
4. Join-based Blocking
   (a) Lossless Join-based Blocking
   (b) Lossy Join-based Blocking
   (c) Spacial Join-based Blocking
5. Progressive Blocking
   (a) Schema-based Progressive Bl.
   (b) Schema-agnostic Progressive Bl.

This taxonomy is presented in more detail in Figures 3.1 (Block Building), 3.6 (Block Processing), 3.9 (Filtering), 3.11 (Join-base Blocking) and 3.12 (Progressive Blocking). In the survey paper [1], not all taxonomies are displayed as Figures – probably due to space constraints. We have done our best to identify all methods referenced in the survey and have listed them together with a shortened description from the survey as well as our own comments (the latter only for Block Building, Block Processing and Progressive Blocking). This list is displayed in Tables 3.1 (Block Building), 3.2 (Block Processing), 3.3 (Filtering), 3.4 (Join-based Blocking), 3.5 (Progressive Blocking). Our focus lies on Block Building, Block Processing and Progressive Blocking as these are the tasks most closely related to our work. In our opinion, the taxonomy presented in the survey [1] is not always optimal in the sense of exposing generalizations, commonalities and peculiarities of the individual methods. We have noticed that a great number of Block Building and Block Cleaning methods can be described in a simple matrix-based

[1]Following the original publication, we capitalize the respective terms in this section.

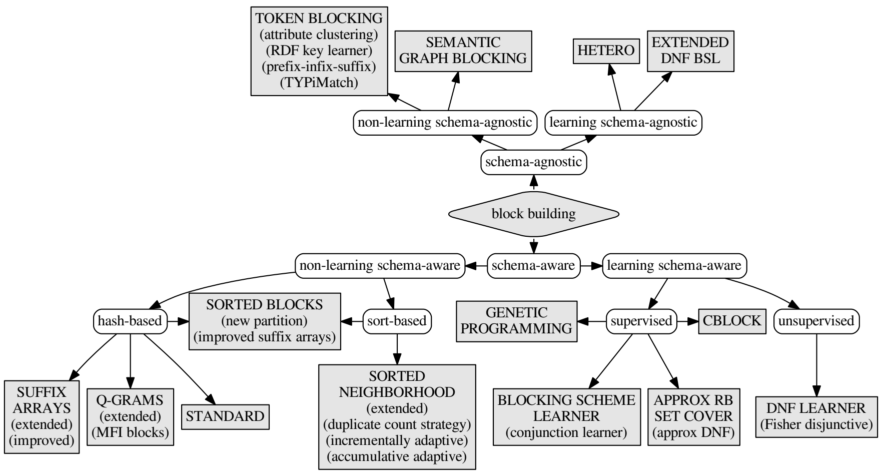
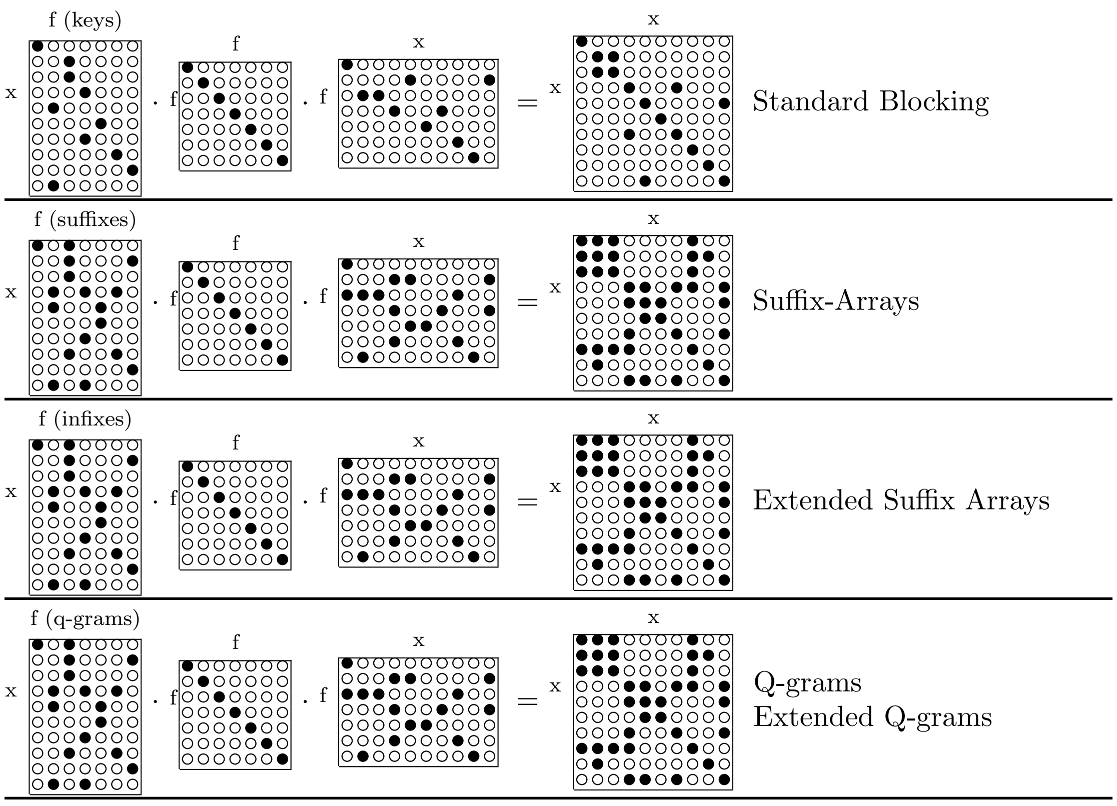


Figure 3.1: Block building taxonomy as defined in Papadakis et al. [1]

formalism, which is visually accessible and useful to discover the above mentioned taxonomic relationships. The visualizations corresponding to the matrix-based formalism are displayed in Figures 3.2 (Hash-based Block Building), 3.3 (Sorted-Neighborhood Block Building), 3.4 (Hybrid Block Building), 3.5 (Schema-agnostic Block Building), 3.7 (Static Block Cleaning). Finally, we have created our own flat taxonomy for Block Building methods that we present in Table 3.6. Here, we distinguish between the kind of blocking features, the feature-similarity and the threshold applied on it and whether a representation is usually assigned one or multiple features. Furthermore, we try to determine whether the methods feature the desired properties formulated in the introduction, which constitutes a kind of application-based or user-side taxonomy in contrast to the technical taxonomy inherent in the other columns of this table. In contrast to the survey [1], we do not distinguish between *deduplication* and *record linkage*, as they are essentially the same, only that in the latter some side-information is available that tells us we do not need to compare record pairs within each collection.

3.7.1 Block Building

In the survey [1], Block Building describes the process of creating a representation to block(s) mapping from scratch (i.e. from the representations). Any refinement of this block-mapping is described under Block Processing in the following subsection. We have summarized these methods in Table 3.1 and reproduce the taxonomy by Papadakis et al. [1] in Figure 3.1. Our sparse matrix view that works with most block building methods consists of two matrices (cf. Figure 3.2). The first is the $|X|\times|F|$ matrix ${}^{x}\overset{f}{\boxplus}$ that assigns each representation $x \in X$ its features $f \in F$. In addition, we assume an $|F|\times|F|$ matrix ${}^{f}\overset{f}{\boxplus}$ that maps each feature $f \in F$ to all sufficiently similar features. We can then obtain



● filled cell (true) ○ empty cell (false)

FIGURE 3.2: Describing hash-based block-building as matrix multiplication.

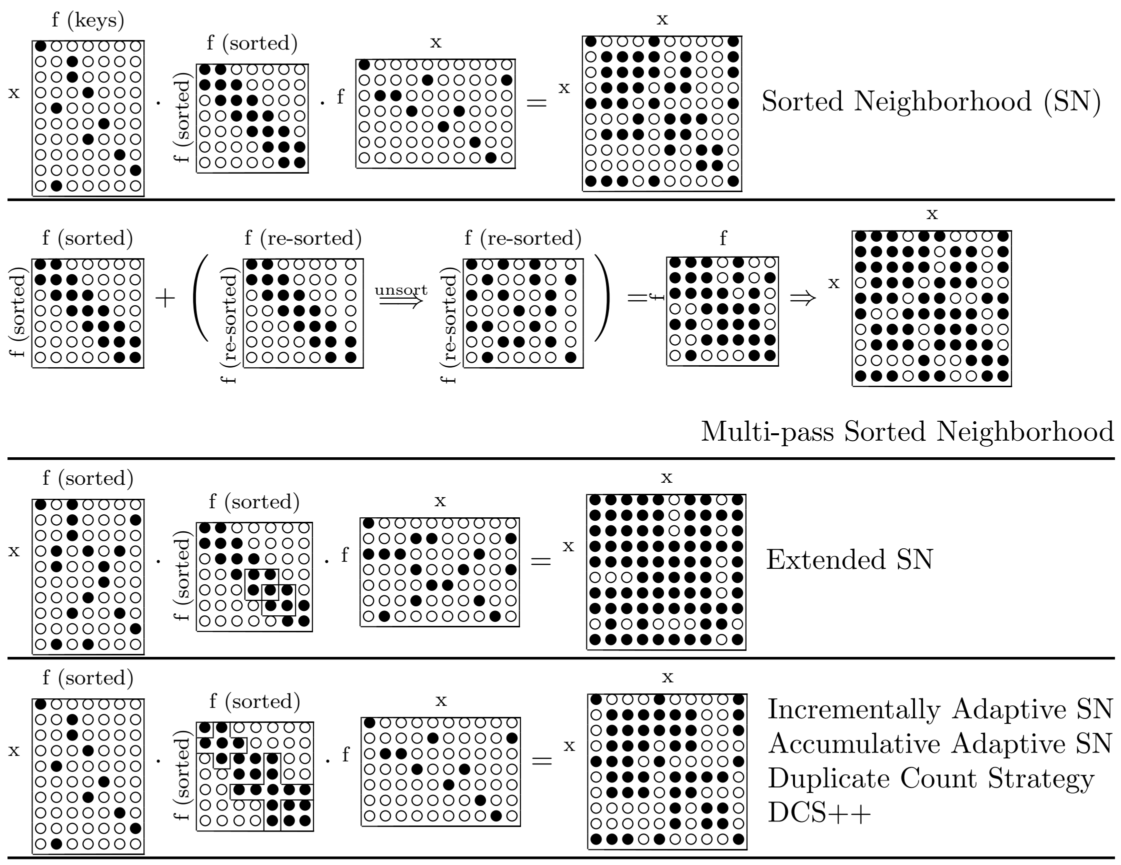
the sparse $|X| \times |X|$ matrix ${}^{x}\overset{x}{\boxplus} = {}^{x}\overset{f}{\boxplus} \cdot {}^{f}\overset{f}{\boxplus} \cdot {}^{f}\overset{x}{\boxplus}$, where ${}^{f}\overset{x}{\boxplus} = {}^{x}\overset{f}{\boxplus}^{T}$. The taxonomy from the survey [1] is in Figure 3.1.

3.7.1.1 Hash-based Block Building

In Hash-based Block Building, ${}^{f}\overset{f}{\boxplus}$ is diagonal, that is each feature is only assigned to itself and there are no feature-feature relations. The differences between the hash-based methods can be summarized quickly in reference to Figure 3.2. **Standard Blocking**: map only one feature (i.e. blocking key, i.e. feature-combination) to each representation. This returns a disjoint blocking. **Suffix-Arrays**: use suffixes of the blocking key as features. This is not disjoint. **Extended Suffix Arrays**: Use infixes instead. **Q-grams** and **Extended Q-grams**: Use character q-grams instead.

3.7.1.2 Sort-based Block Building

The specialty of sort-based block building methods is to exploit the fact that alphabetical sorting can be done in subquadratic time by sorting features and using neighborhood



● filled cell (true) ○ empty cell (false)

FIGURE 3.3: Describing various sort-based block-building methods as matrix multiplication.

in the resulting total order as an approximation of string similarity. The matrix views are shown in Figure 3.3. In **Sorted Neighborhood**, only one key is assigned to each representation and these keys are sorted alphabetically. Obviously this does not work with hashed keys but requires them to be such that their alphabetical ordering resembles a meaningful similarity. In **Multi-pass Sorted Neighborhood**, we use different types of sortable features and get the final $\overset{x}{{}^{x}\boxplus}$ matrix for each of them to then finally add up all the different $\overset{x}{{}^{x}\boxplus}$. In **Extended Sorted Neighborhood**, multiple features can be assigned to each representation. In **Incrementally Adaptive Sorted Neighborhood**, **Accumulative Adaptive Sorted Neighborhood**, **Duplicate Count Strategy** and **DCS++**, the window size applied on the sorted features in order to define feature-neighborhood varies depending on the current feature in focus and its concrete neighbors in the total alphabetical order.

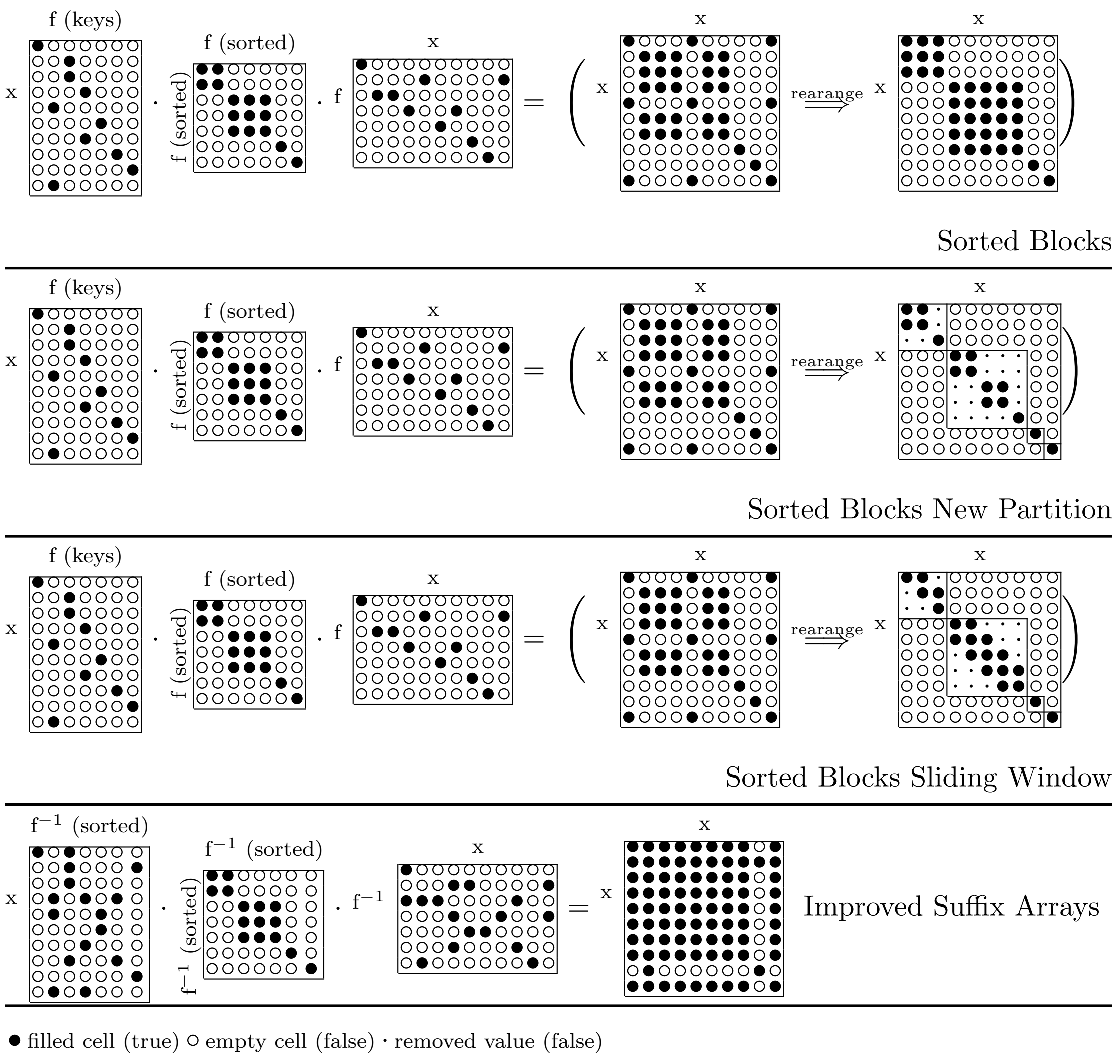
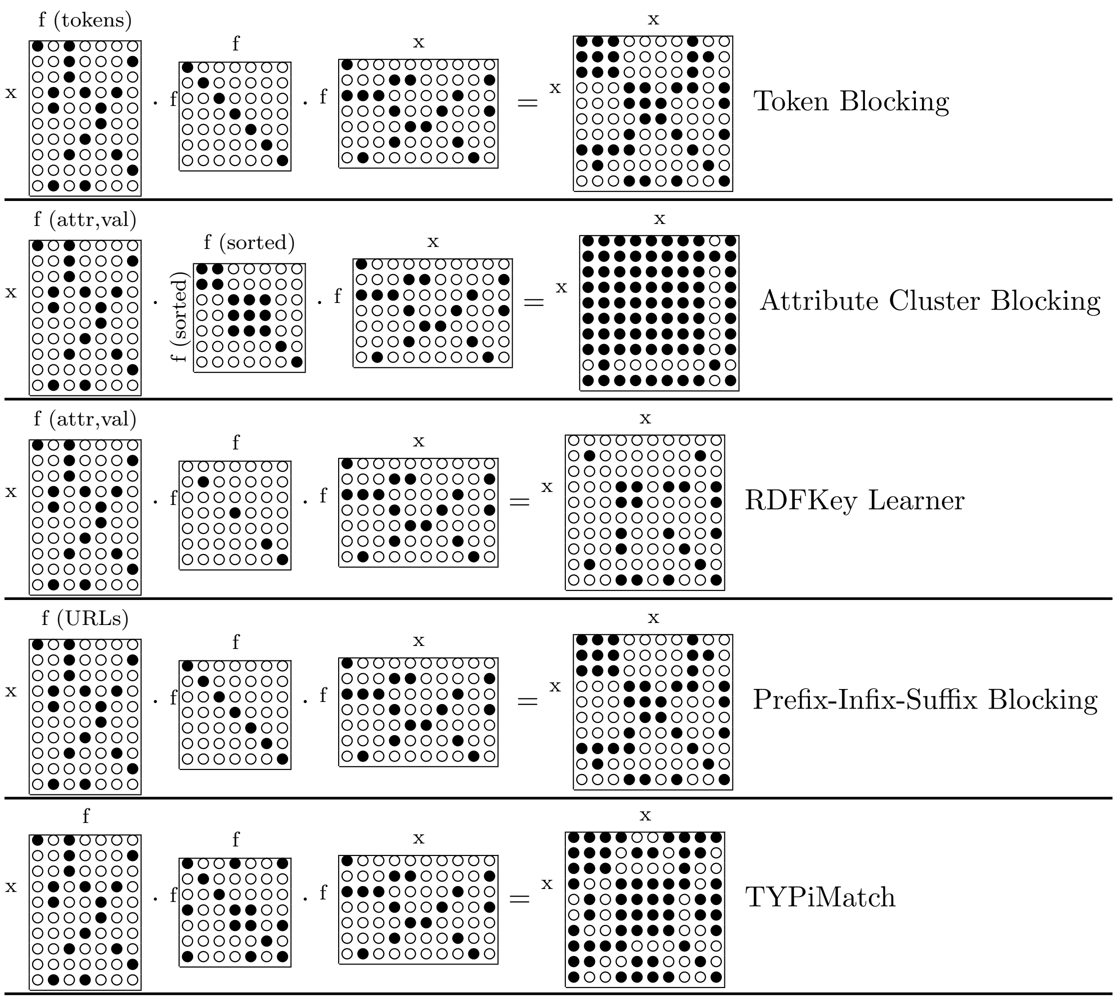

f (keys)
f (sorted)
x
f (sorted)
f
rearange
Sorted Blocks
Sorted Blocks New Partition
Sorted Blocks Sliding Window
f⁻¹ (sorted)
Improved Suffix Arrays
● filled cell (true) ○ empty cell (false) · removed value (false)

Figure 3.4: Describing hybrid block-building as matrix multiplication.

3.7.1.3 Hybrid Block Building Methods

Hybrid Block Building methods can be described as performing a blocking on features (see Figure 3.4). In **Sorted Blocks**, each representation is assigned a single blocking key. These are then grouped into disjoint blocks rather than continuously overlapping windows as in the Sorted-Neighborhood approaches. Because of the 1:1 mapping from representations to features, this corresponds to a disjoint blocking on the representation level. In **Sorted Blocks New Parition**, the same method is applied, only that the resulting representation blocks are split up if they are too large. Likewise, in **Sorted Blocks Sliding Window**, a sliding window is applied on the result matrix $^{x}\overset{x}{\boxplus}$. However, if we are not mistaken, this last step should actually be considered a means of *Comparison Cleaning* as discussed later. In **Improved Suffix Arrays**, each representation is assigned multiple features, i.e. suffixes of the blocking key, and the suffixes



● filled cell (true) ○ empty cell (false)

FIGURE 3.5: Describing schema-agnostic block-building as matrix multiplication.

are sorted and partitioned based on this order. This can be implemented by reversing the suffixes and applying the Sorted Blocks method. Thereby the length of the suffix-overlaps will be rewarded by proximity in the alphabetical order.

3.7.1.4 Schema-agnostic Block Building Methods

In our opinion, schema-agnostic methods do not really define a separate category of block building methods, but constitute additional methods that are similar to methods previously described in various categories. The difference is simply in the feature-extraction step. The methods are visualized in Figure 3.5. In **Token Blocking**, a representation is assigned multiple tokens that are used as before to determine overlapping blocks. The definition of a token is essentially free. **Attribute Cluster Blocking** is similar to Improved Suffix Arrays in that it allows to assign multiple features to one representation and groups the features. The difference is that here, the feature-blocking it is

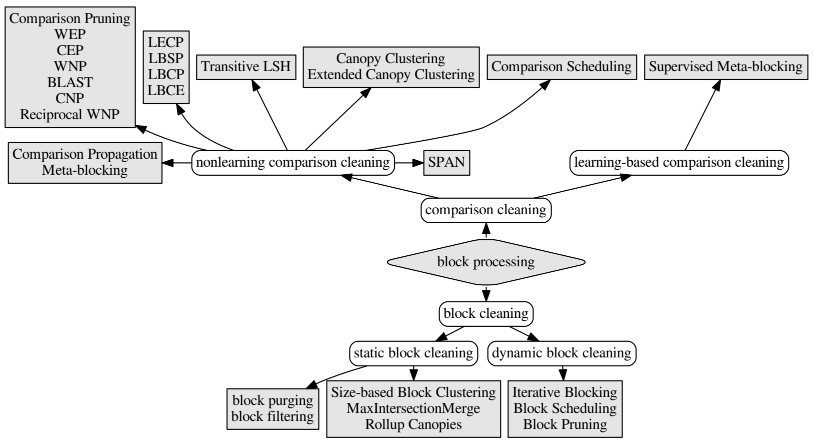


FIGURE 3.6: Block processing taxonomy as defined in Papadakis et al. [1]

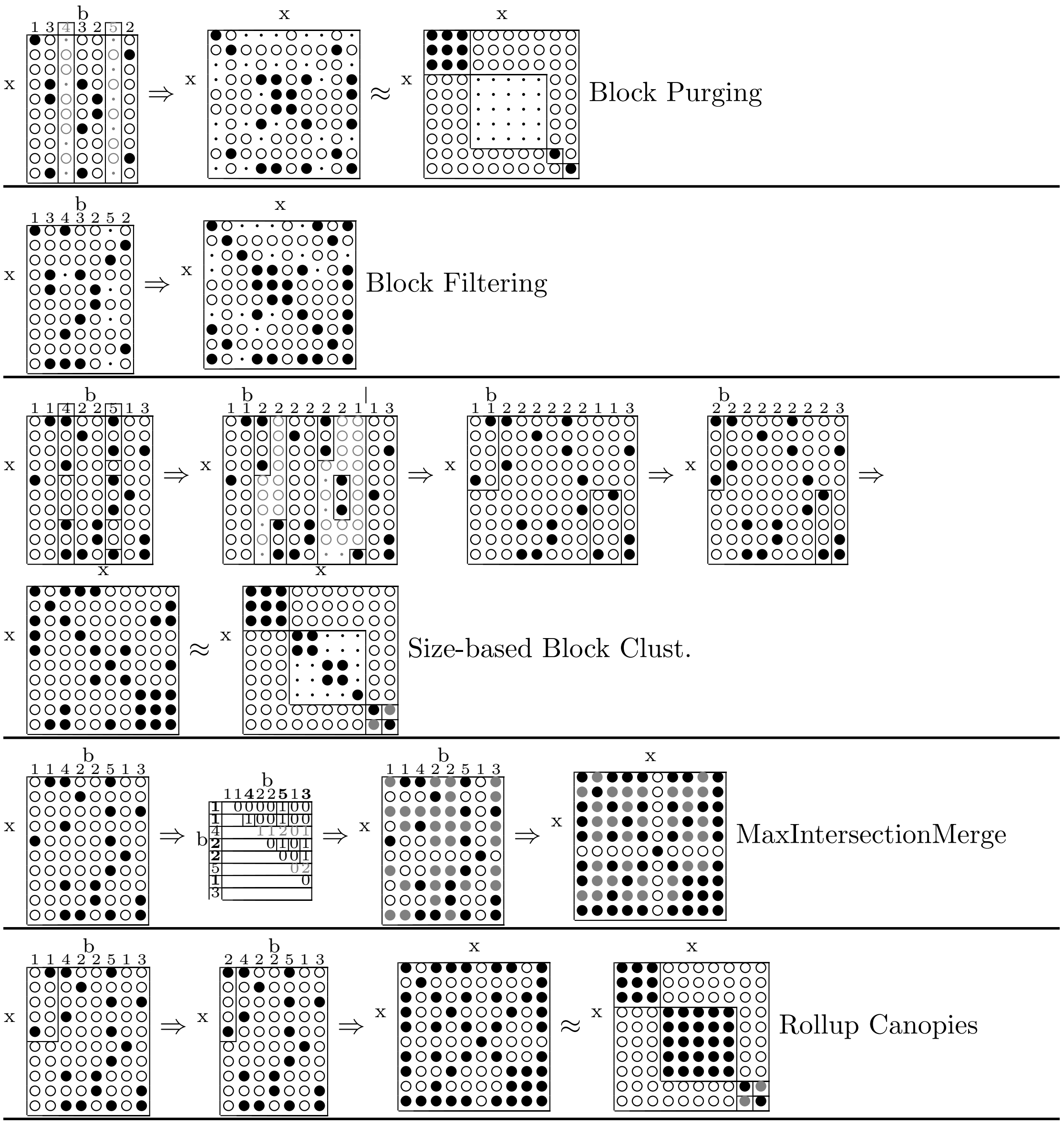
not based on the alphabetical order but on some attribute relationships as all features are attribute-value pairs. We have noted before that if $|F|$ is sufficiently small, then an exhaustive comparison of features might be feasible. **RDFKey Learner** allows to de-select some features, which can be implemented by empty rows (and columns) in ${}^{f}\overset{f}{\boxplus}$. In **Prefix-Infix-Suffix Blocking**, the features are selected to be components of the URLs representing the entities to be disambiguated. This obviously requires a very specific input data. Finally, **TYPiMatch** tokens are considered entity-references. These are grouped into "entity-types" in ${}^{f}\overset{f}{\boxplus}$ based on distributional similarity in a different dataset. As a result, all representations are compared that share at least one feature with the same entity-type.

3.7.1.5 Block Building Methods outside our Scheme

A number of Block Building methods do not fit as easily into the sparse matrix framework introduced above. One of them is **MFIBlocks**, which we have already discussed in Section 3.1. **Semantic Graph Blocking** is defined on a semantic knowledge graph that could of course be encoded in a matrix, however the exploited notion of path length is rather hidden in this view. The others are all block learning methods, i.e. *Supervised Block Learning*, *Unsupervised Block Learning* and *Schema-agnostic Block Learning*. The problem of learning blocking schemes has been discussed to some extend in Section 3.2. There, we have also shown how to extend the above matrix view by a sparse matrix ${}^{x}\overset{k}{\boxplus}$ where ${}^{x}\overset{k}{\boxplus}_{ij} = \prod_l {}^{x}\overset{f}{\boxplus}_{il} \times {}^{f}\overset{k}{\boxplus}_{lj}$. While this does not answer the question *how* such schemes are learned (which is what mainly distinguished the Block Learning methods), still we have seen that the result can be easily integrated in the matrix formalism.

Hash-Based	**Standard Blocking (SB)** Concatenate features to form blocking keys. One key per mention. One block per key. All mentions with this key are in the block. → *A hash function that maps mentions into non-overlapping blocks trying to minimze cross-block coreferences.*
	Suffix Arrays Blocking (SA) Split keys into all suffixes longer than a threshold. Remove frequent suffixes. One block per suffix. → *Essentially just a key-generalizing preprocessing step to Standard Blocking. Should generate overlapping blocks.*
	Extended Suffix Arrays Blocking Split keys into all infixes longer than a threshold. → *Just a straightforward modification to Suffix Arrays Blocking.*
	Q-Grams Blocking Split keys into all character q-grams. → *Essentially just a key-generalizing preprocessing step to Standard Blocking.*
	Extended Q-Grams Blocking Split keys into all character q-grams. Remove frequent q-grams. → *Just a straightforward modification to Q-Grams Blocking.*
	MFIBlocks Maximal frequent itemset algorithm to define as blocking keys those q-grams exceeding a support threshold. → *Q-grams are not the essential part here. This is quite a unique approach.*
Sort-based	**Sorted Neighborhood (SN)** Sort all blocking keys alphabetically. Compare mention at position i with those at i-w,i-w+1,...,i-1. → *Relies fully on the total alphabetical order, which however often does catch meaningful similarities in meaning.*
	Multi-pass SN Use different blocking key functions and repeat for all of them. → *Straight-forward extension of Sorted Neighborhood.*
	Extended Sorted Neighborhood Compare all mentions that have the blocking key at position i with those mentions that have the key at i-w,i-w+1,...,i-1. → *Add an intermediate level, which I guess means that one mention can have multiple keys.*
	Incrementally Adaptive SN Increase window size w depending on similarity of mention i and i-w. → *Makes sense because you need to calculate the similarity of mention i and i-w anyway.*
	Accumulative Adaptive SN Increase window size w depending on likelihood of finding an equivalence in the window. → *How is this different from Incrementally Adaptive SN? Maximum similarity should correlate with average similarity.*
	Duplicate Count Strategy (DCS) Increase window size w depending on likelihood of finding an equivalence in the window in proportion to window size. → *Straight-forward modification of Accumulative Adaptive SN.*
	DCS++ Some technical improvements to DCS. → *Conceptionally the same as DCS.*
Hybrid	**Sorted Blocks** Sort blocking keys lexicographically. Partition entities using blocking key prefixes. Extend partitions by fixed length overlaps. → *Seems to be just a recombination of previous principles. If full keys are sorted then same prefixes are next to each other.*
	Sorted Blocks New Partition Start another block when the current block's size exceeds a threshold for the current blocking key prefix. → *The most important aspect is the dynamic consideration of block size. However, the breaking point is arbitrary and not optimized.*
	Sorted Blocks Sliding Window Use a sliding window over each block separately. → *Straightforward recombination of previous approaches in a two-layered framework. But how are the blocks sorted?*
	Improved Suffix Arrays Blocking Sort blocking key suffixes alphabetically. If consecutive suffixes are sufficiently similar, the corresponding blocks are merged. → *Basically determining block boundaries by blocking key similarity, conflating similar keys. How is this not incrementally adaptive?*
Supervised	**ApproxRBSetCover** Learn disjunctive blocking schemes by solving a weighted set cover problem. → *This seems to be quite a remarkable approach that is quite different from the standards.*
	ApproxDNF Greedily learns a conjunction of up to k predicates that maximize the ratio of positive and negative covered instances. → *Probably a different approach to the same task.*
	Blocking Scheme Learner (BSL) Adapt the sequential covering algorithm to learn blocking schemes that maximize RR and while maintaining PC above a threshold. → *Sequential Covering learns a disjunctive set of rules. Learn a rule, remove the data that it covers, then repeat.*
	Conjunction Learner Minimizes candidate matches not only in the labeled but also in the unlabeled data. →
	BGP A set of genetic programming operators, such as copy, mutation, and crossover, are iteratively applied to the initial set of blocking schemes. → *Basically the whole scheme learning simply picks the atribute combinations that best discriminate coreference. This is one more way to do it.*
	CBlock Every edge is a transformation function and every node holds the entities that result after applying all functions from the root down. → *Similar to our approach if the representation features are chosen respectively and generalizations created for the nodes used here.*
	DNF Learner Apply a matching algorithm to some entity pairs to create a labeled dataset and learn DNF blocking schemes. → *Basically semi-supervised blocking scheme learning.*
Unsup-ervised	**FisherDisjunctive** Use extreme TF-IDF to get gold pairs. Learn DNFs by Fisher feature selecting relevant shared keys in the pairs. → *Another semi-supervised blocking scheme learning approach.*
Schema-Agnostic Nonlearning	**Token Blocking (TB)** Create a block for each token over all features. → *Essentially just a another key-generalizing preprocessing step to Standard Blocking.*
	Attribute Clustering Blocking Group attributes by similar values and use (group,token) pairs to determine blocks. → *This is a key-generalization that focuses on generalizing the attribute rather than the value of attribute-value pairs.*
	RDFKeyLearner Derive attribute discriminability from #distinct values and attribute coverage from #entities having it. Use the mean to select blocking keys. → *The mean is a rather arbitrary connection for this tradeoff. Very similar to tf-idf. Why was the latter not used instead?*
	Prefix-Infix(-Suffix) Blocking Use URL domain, local identifier and format or anchor along with all tokens in the literal values as blocking keys. → *Quite specific to URL-identified mentions.*
	TYPiMatch Detect entity types in a co-occurrence graph connecting tokens with high co-occurrence. Apply token blocking over mentions sharing a type. → *To call it a type seems unnecessary. It is a two-step blocking process.*
	Semantic Graph Blocking Use semantic relations between mentions. Group mentions if they share a short enough path and this block is not too large. → *This requires there to be relations annotated to actual mentions. In other words all mentions must occur in a relational context.*
Schema-Agn. Learning	**Hetero** Represent mentions as normalized TF vectors and apply an adapted Hungarian algorithm for optimal assignment, followed by FisherDisjunctive. → *What means assignment here?*
	Extended DNF BSL Combine an established instance-based schema matcher with weighted set covering to learn DNF blocking schemes with at most k predicates. → *Not sure what is learned in the DNF if there are no attributes? If all the values are learned then there must be many conjunctions.*

Table 3.1: Block building methods as presented in the survey by Papadakis et al. [1]



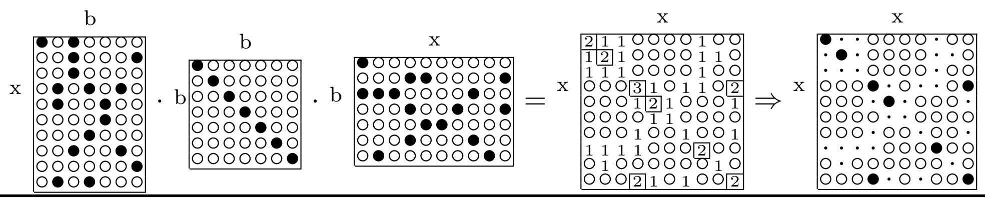
● filled cell (true) ○/○ empty cell (false) ·/· removed value (false) ● added value (true)

FIGURE 3.7: Describing static block cleaning methods in matrix terms. The final matrices after ≈ are only schematic.

3.7.2 Block Processing

In the survey [1], Block Processing describes the refinement of the block-mapping created under Block Building in the previous subsection. We have summarized these methods in Table 3.2 and reproduce the taxonomy by Papadakis et al. [1] in Figure 3.6. Our sparse matrix view can also be used to visualized some of these methods.

Block Processing methods that can be easily described and distinguished in the matrix view are *Static Block Cleaning* methods. These concern modifications of $^{x}\overset{f}{\boxplus}$ that result in a sparsification of the output $^{x}\overset{x}{\boxplus}$ (see Figure 3.7). The blocks b can be identical to the



● filled cell (true) ○ empty cell (false) · removed value (false) 2 integer value

FIGURE 3.8: Example of how comparisons can be cleaned based on number of shared blocks as returned by the (sparse) dot product. Here at least two shared blocks are required. Of course a mention x could also be attributed to a block b in a weighted way, returning a (sparse) degree-of-overlap matrix. Also, the block-block matrix could be weighted, e.g. inversely proportional to block-size. Almost all non-learning comparison cleaning approaches can be defined in this framework. The survey by Papadakis et al. [1] refers to this approach as *redundancy-positive*.

features f or the keys k used previously. In Figure 3.7, the size of each block is shown in the top row of $^{x}\overset{b}{\boxplus}$. **Block Purging** removes the largest blocks – that is columns from $^{x}\overset{b}{\boxplus}$. Note that this avoids the creation of all pairs of mentions in these blocks. **Block Filtering** removes each mention from the largest block it is part of. This eliminates considerably more pairs than cells dropped in $^{x}\overset{b}{\boxplus}$. In **Size-based Block Clustering**, large blocks are split up and small blocks are merged. This boths removes and adds pairs to the output matrix. **MaxIntersectionMerge** does not remove pairs but only adds some by merging blocks that share the most mentions. Neither does **Rollup Canopies**, which merges the smallest blocks but leaves the largest ones intact. As the focus of *Dynamic Block Cleaning* is on a certain sequential order of operations, it cannot be displayed as a single state of a matrix.

Nonlearning Comparison Cleaning methods are not displayed individually in the matrix view, but most of them can be thought of as applying some threshold on a weighted output matrix $^{x}\overset{x}{\boxplus}$ (see Figure 3.8). As they are directly operating on the result pairs, they do not improve the computational complexity – unless one assumes the application of the final similarity measure (and perhaps an expensive clustering algorithm) to be much more time-consuming than the relevance test performed at this point. Most of these methods exploit what is referred to in the survey [1] as *redundancy positive*, i.e. some weight for a pair of mentions based on how many blocks they share followed by some specific local or global threshold. If do not apply the Boolean dot-product as before, but use integers or floating point numbers, nonzero cells in $^{x}\overset{x}{\boxplus}$ contain the number of blocks shared. Finally, two Learning-based Comparison Cleaning methods are mentioned, whereof the second is only a trick to reduce the amount of annotated data required by the first.

Static Block Cleaning	**Block Purging** Discard blocks that exceed an upper limit on block cardinality. → *Extremely primitive, but what about the mentions that are only in the purged blocks?*
	Block Filtering Remove every mention from the largest blocks that contain it. → *Here, no mentions are dropped, but sometimes the largest block might be the correct one? Are block sizes updated along the way?*
	Size-based Block Clustering Hierarchically cluster small blocks and split large ones according to the tradeoff between intra-block similarity and block size. → *The special aspect is that small blocks are merged.*
	MaxIntersectionMerge Merge each small-enough block with the block that has the most mentions in common and is large-enough. → *This should have a tendency to merge small blocks into large ones.*
	Rollup Canopies Use positive examples to train a greedy algorithm that merges pairs of small-enough blocks to increase recall. → *Here small blocks are merged and a different method is used for it.*
Dynamic Block Cleaning	**Iterative Blocking** Iteratively merge any new pair of detected duplicates in all blocks until convergence. Re-compare the result with all co-occurring entities. → *I would rather call this a basic property that any ER method could have or not. Not sure how this related to blocking.*
	Block Scheduling Process blocks in the order of their size. → *Basically as primitive as Block Purging. Not sure why this is not under progressive resolution.*
	Block Pruning Terminate block scheduling if the average number of executed comparisons per found duplicate falls below a threshold. → *Makes sense, but it is simply a stopping criterion for block scheduling.*
Nonlearning Comparison Cleaning	**Comparison Propagation** Use the mentions-block mapping to precompute all comparisons instead of processing all pairs in all blocks with redundant comparisons. → *This is a straightforward approach, but it should require more space as you need to compute all comparison pairs first.*
	Comparison Pruning Use the Jaccard coefficient to skip comparing mentions with different-enough sets of blocks. → *This calculates a weight for each pair of mentions and applies a global threshold on the resulting graph.*
	Weighted Edge Pruning (WEP) Create a graph that connects mentions if they cooccur in the same block. Weight edges by number of shared blocks and prune light-enough edges. → *Not sure how this is different from Comparison Pruning except that the weight is calculated differently.*
	Cardinality Edge Pruning (CEP) Retain global top k edges of blocking graph. → *Just a specific way for a threshold.*
	Weighted Node Pruning (WNP) Retain locally heavy-enough edges, e.g. by average edge weight of node neighborhood. → *Just a specific way for a threshold.*
	BLAST Define heavy-enough as the average of the maximum weights in two adjacent node neighborhoods. → *Just a specific way for a threshold.*
	Cardinality Node Pruning (CNP) Retain local top k weighted edges for each node neighborhood. → *Just a specific way for a threshold.*
	Reciprocal WNPfiltering Apply aggressive pruning that retains edges satisfying the pruning criteria in both adjacent node neighborhoods. → *Just a specific way for a threshold.*
	Low Entity Co-occurrence Pruning (LECP) Remove all those mentions from a block that have the lowest average edge weights within this block. → *Make a mention-block matrix with the average weights and pick top rows for each column.*
	Large Block Size Pruning (LBSP) Apply LECP only to blocks whose size exceed average block size. → *This is a very minor modification of LECP.*
	Low Block Co-occurrence Pruning (LBCP) Remove a mention from all those blocks where it has the lowest average edge weights among all blocks. → *This makes sure that no mention is fully dropped from all blocks. Make a mention-block matrix with the average weights and pick top columns.*
	CooSlicer In large-enough blocks, sort all mentions by average edge weight and place the least weighted mentions into a new block. → *Basically instead of ignoring, making a new block. But I do not see why multiple low-weighted mentions should be in the same block.*
	Low Block Co-occurrence Excluder (LBCE) Discard those blocks with the lowest average edge weight over their mentions. → *Nothing complicated, drop entire blocks. However that would also remove strong associations within low-quality blocks.*
	Transitive LSH Convert LSH blocks into a blocking graph and apply community detection to partition it. Stop when the largest partition is small enough. → *A two-step approach that cannot really be generalized to other methods as it basically two black-boxes.*
	Canopy Clustering Iteratively move a random mention as well as the most similar remaining mentions into a new block. Reasonably similar ones are copied, not moved. → *Should be quite expensive.*
	Extended Canopy Clustering ??? → *???*
	SPAN Apply spectral clustering on the tf.idf-based similarity of mentions sharing keys to obtain a binary tree from which blocks are derived. → *Feed sparse mention-mention matrix with tf.idf-similarity of key-sharing mentions to spectral clustering and obtain a dimensionality reduction.*
	Comparison Scheduling Order comparisons by the respective edge weight and execute only those comparisons where both mentions have not been matched to any other. → *Not sure why this is not progressive. This is probably only for clean-clean ER as clusters can be only of size two.*
Learn.-based Comp. Clean.	**Supervised Meta-blocking** Treat edge pruning as a binary classification problem representing edges to classify by ARCS, ECBS, JS, and the node degrees of adjacent mentions. → *It seems not 100% instuitive why the pruning classification is cheaper than the coreference classification.*
	BLOSS Restrict labeling cost by carefully selecting a training subset that is much smaller, but retains original performance. → *This is merely a modification to reduce the requirements on training data.*

Table 3.2: Block processing methods as surveyed in Papadakis et al. [1]

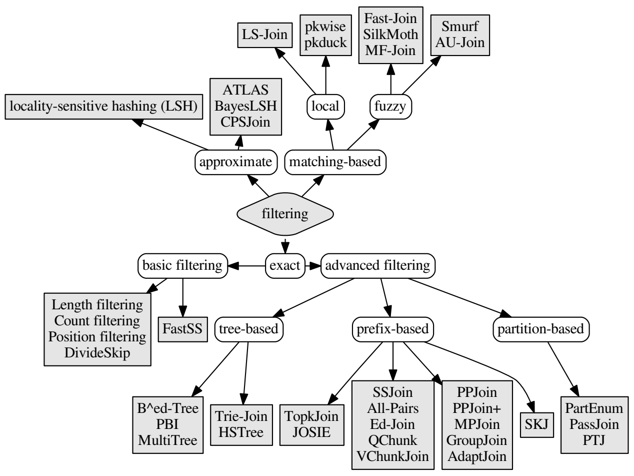


FIGURE 3.9: Filtering taxonomy as defined in Papadakis et al. [1]

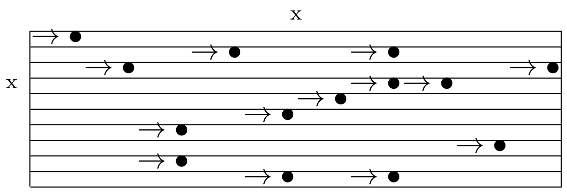


FIGURE 3.10: Visualization of the task of filtering according to Papadakis et al. [1] The essence is to jump from a given mention straight to its similar-enough partners, i.e. by neighbors in total alphabetical order.

3.7.3 Filtering

In the survey [1], Filtering describes a family of methods that help to find similar mentions given some reference mention (see Figure 3.10). As the task is to achieve this without going through all mentions in the collection or in the current block, this is an application of similarity search as discussed in Section 3.4. We have summarized these methods in Table 3.3 and reproduce the taxonomy by Papadakis et al. [1] in Figure 3.9. Filtering can be applied on the entire collection or on individual blocks. Some of the methods presented in the survey [1] use the popular Locality Sensitive Hashing (e.g. **ATLAS** or **BayesLSH**, here called “approximate”). Other “basic” methods use bounds like length difference (**Length Filtering**), q-gram overlap (**Count Filtering**) or slightly more advanced bounds (**Position Filtering**, **DivideSkip**, **FastSS**). An entire group of methods exploits such bounds based on string-prefixes (e.g. **Ed-join** or **Qchunk**). Others perform a kind of approximate nearest neighbor search in edit distance by exploiting redundancies and overlaps in the strings representing mentions (e.g. **PartEnum** or **B*ed Tree**, here called “partition-based” and “tree-based”). The most advanced Filtering methods strive to implement “Fuzzy Matching”, i.e. a gradual (an approximate) interpretation of the matching relation, e.g. via signatures that overlap when matching (**Fast-Join**) or via blocking predicates that should be satisfied for a

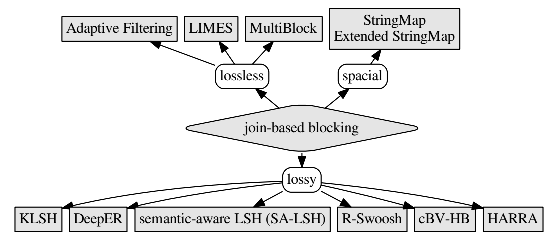


FIGURE 3.11: Join-based blocking taxonomy as defined in Papadakis et al. [1]

match (**Smurf**). Essentially, the task of Filtering is to structure the search space to efficiently exclude dissimilar pairs from further comparison. The main idea behind Filtering is that some indexable properties are assumed to be a minimum requirement or necessary condition for equality. Then, given a reference mention, one can pick the minimum overlap requirements for this mention and use these properties to look up related mentions in an inverted index. For example in **Prefix Filtering**, it is defined that it is necessary for two strings to share the most specific character q-grams. Then given some string-based representations, an index structure can be used to retrieve the intersection of all mentions with the reference mention's most specific q-grams (Boolean retrieval). This is similar to a limited number of allowed hops in a subset partial order graph of representations where generalizations like reference strings in **PBI** are added for some/all strings in the original representations and subsets with only such generalized features are hypothesized, tying together representations by greatest lower bounds.

3.7.4 Join-based Blocking

What is referred to as "Join-based Blocking" methods in the survey [1] is related to Filtering in that methods of the latter family are deployed not on the input or output of Block Building methods, but during its creation. This means that Filtering is applied to get ${}^{x}\overset{b}{\boxplus}$, not on ${}^{x}\overset{x}{\boxplus}$. We have summarized these methods in Table 3.4 and reproduce the taxonomy by Papadakis et al. [1] in Figure 3.11. In **Adaptive Filtering**, **LIMES** and **MultiBlock**, bounds are applied in the blocking context, e.g. by applying length and count filtering to the largest ones. The essence of *Lossy Join-based methods* like **KLSH** or **DeepER** lies in mapping the original input representations into a different space (of lower dimensionality), where string similarities are regarded and correspond to closeness. An approach that lends itself well to this task is of course locality sensitive hashing as discussed in Section 3.4. It is used in three out of five of the surveyed Lossy Join-based Blocking methods and is prototypical for what is understood as Join-based Blocking since it builds ${}^{x}\overset{b}{\boxplus}$, so that ${}^{x}\overset{b}{\boxplus} \cdot {}^{x}\overset{b}{\boxplus}^{T} = {}^{x}\overset{x}{\boxplus}$. Note how in contrast to filtering it does not operate on the result ${}^{x}\overset{x}{\boxplus}$ but instead contributes to its creation. In *Spacial Join-based*

Basic	**Length filtering**	Use length difference as an upper bound on edit distance.
	Count filtering	Use common character n-grams as an upper bound on edit distance.
	Position filtering	Use the difference between two positional n-grams as an upper bound on edit distance.
	DivideSkip	Use prefix-, length- and position filtering, and efficiently merge the inverted lists of signatures.
	FastSS	Obtain a string's deletion neighborhood by removing a certain number of characters. Use it as a filtering criterion for edit distance.
Prefix-Based	**Prefix filtering**	Represent strings as n-grams ordered by specificity. For two sets to be compared, their k most specific n-grams must overlap.
	Ed-Join	Analyze the locations and contents of mismatching n-grams to further removing unnecessary elements from the most specific n-grams.
	QChunk	Use the size of the intersection between the n-grams of one string and the n-substrings of another as upper bound for their edit distance.
	VChunkJoin	Use non-overlapping substrings whereof each edit operation destroys at most two to determine whether two mentions can match.
	PPJoin	Extend prefix filtering by the position of common tokens in the prefix to derive a tighter upper bound for set overlap.
	PPJoin+	Also use suffix filtering to calculate the maximum number of tokens in each pair of corresponding partitions between the two sets that can match.
	MPJoin	Add a further optimization over PPJoin to dynamically prune the inverted lists.
	GroupJoin	Extend PPJoin with group filtering to treat all sets with identical prefixes as one. Multiple candidates may thus be pruned in batches.
	AdaptJoin	Select longer prefixes for each set where appropriate. Prune a pair of sets if there are less than n common tokens in their extended prefixes.
	SKJ	Use index-level skipping to group related sets into blocks. Use answer-level skipping for dynamic answer set programming.
	TopkJoin	Retrieve the most similar set pairs using prefix filtering and the monotonicity of maximum possible scores of unseen pairs.
	JOSIE	Realizes top-k set similarity search for large sets with prefix- and position filtering, minimizing set read cost and inverted index probes.
Partition-Based	**PartEnum**	Generate signatures based on the principles of partitioning and enumeration.
	PassJoin	Create inverted indices for consecutive substrings to retrieve candidates. Minimize the number of substrings required to find candidate pairs.
	PTJ	Increase the pruning power of partition-based filtering using a mixture of subsets and their one-deletion neighborhoods.
	pigeonring principle	Arrange pidgeonholes in a ring and constrain the number of items in multiple boxes rather than a single one, which gives tighter bounds.
Tree-Based	**Bˆed -Tree**	Use edit distance to create an index based on a B+ trees for range-queries, top-k similarity queries and similarity joins.
	PBI	Select prototype strings around which other strings are grouped. Index strings based on distance to their corresponding prototypes.
	MultiTree	Obtain a global ordering to map each element to an integer, which is inserted into a B+ tree. Search for similar elements via range query.
	Trie-Join	If a trie node is not sufficiently similar to each of a string's prefixes, then its descendants cannot be similar either.
	HSTree	Using edit distance, build a hierarchical index of consecutive substrings to support threshold-based and top-k string similarity search.
Approximate	**locality-sensitive hashing (LSH)**	Map similar mentions to the same hash code. Hash each mention several times and compare mentions that share at least one hash code.
	ATLAS	Search for similar pairs within detected clusters. Use random permutations to compute candidates and their similarity.
	BayesLSH	Combine Bayesian inference with LSH to estimate similarities with probabilistic guarantees on the resulting accuracy and recall.
	CPSJoin	Use a recursive filtering technique for set similarity search to obtain perfect precision and a probabilistic guarantee on recall.
	LS-Join	Use character n-gram sharing as a necessary condition for local similarity.
	pkwise	Detect substrings with overlapping contexts. Use token combinations in prefix filtering.
	pkduck	Account for abbreviations. Extend prefix filtering and generate signatures without iterating over all strings derived from an abbreviation.
Fuzzy Matching	**Fast-Join**	Define similarity as maximum matching score in a bipartite graph. Use appropriate token subsets as signatures that overlap when matching.
	SilkMoth	Use heuristics to select signatures. Compare each set with its candidates to reject those condidates for which certain bounds do not hold.
	MF-Join	Use a frequency-aware partition-based signature scheme as well as count filtering and an upper bound on record-level similarity.
	Smurf	Perform multiple-predicate matching through a random forest classifier learned via active learning. Efficiently reuse computations across trees.
	AU-Join	Use character n-grams for synonym-detection and a taxonomy for taxonomy-based matching.

Table 3.3: Filtering methods as presented in the survey by Papadakis et al. [1]

methods **StringMap** and **Extended StringMap**, a similar goal is pursued, but here the target space that representations are mapped into is specifically the Euclidean space.

3.7.5 Progressive Blocking

In the survey [1], Progressive Blocking refers to a process of repeatedly increasing the number of compared pairs. The advantage is that in a large scale resolution scenario,

Lossless Join-based	**Adaptive Filtering**	Apply length and count filtering to the largest blocks while using an edit distance threshold on a dedicated non-blocking attribute.
	LIMES	Use the triangle inequality to compute bounds and approximations of mention distances based on previous comparisons.
	MultiBlock	Aggregate block collections for different similarity functions into a multidimensional index that respects the co-occurrence of similar entities.
Lossy Join-based	**KLSH**	Apply k-Means to compressed feature representations to obtain disjoint blocks.
	DeepER	Use word embeddings for mention representations and hash them into buckets with LSH. Create a block for each mention's most likely matches.
	semantic-aware LSH (SA-LSH)	Use the length of taxonomy paths to obtain semantic similarity of features and low-dimensional representations to combine with textual keys.
	cBV-HB	Embed features into a compact binary Hamming space where misspellings correspond to specific distance bounds and logical operators apply.
	HARRA	Use LSH to hash similar entities into buckets within which duplicates are merged and re-hashed. Repeat until convergence or abortion.
Spacial Join-based	**StringMap**	Convert keys to a similarity-preserving low-dimensional Euclidean space and cluster them into overlapping blocks.
	Extended StringMap	Map mentions into an intermediate Euclidean space before mapping keys into another Euclidean space of lower dimensionality.

TABLE 3.4: Join-based blocking methods as surveyed in Papadakis et al. [1]

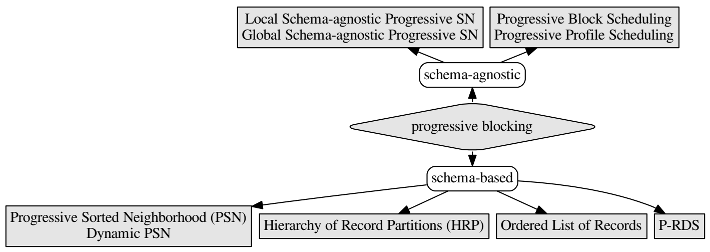
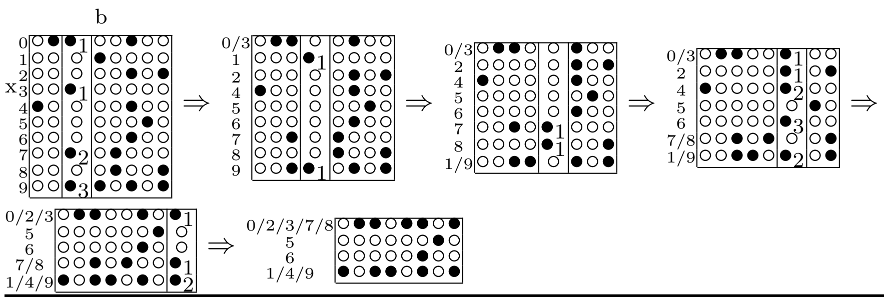

Local Schema-agnostic Progressive SN
Global Schema-agnostic Progressive SN
Progressive Block Scheduling
Progressive Profile Scheduling
schema-agnostic
progressive blocking
schema-based
Progressive Sorted Neighborhood (PSN)
Dynamic PSN
Hierarchy of Record Partitions (HRP)
Ordered List of Records
P-RDS

FIGURE 3.12: Progressive blocking taxonomy as defined in Papadakis et al. [1]



● filled cell (true) ○ empty cell (false) $●_1$ cell assigned to cluster 1

FIGURE 3.13: Visualization of clustering one block and as a result iteratively merging mentions x.

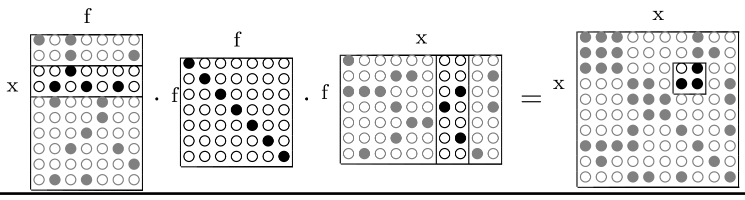
the resolution step can be triggered at some point and the resolution continuously improves until the process is terminated due to time constraints. Continuous improvement is not the only requirement. In addition, the goal is that in the sequential ordering of comparisons, the ones that are most likely to lead to detected duplicates come first. Even more, the expectation is that for any pair of mentions compared, all pairs with a higher coreference likelihood have already been compared. As this is a very strict requirement, the optimal solution can only be approximated. If we consider a mix of precision and recall as our definition of quality, then the continuous greedy increase in quality is usually achieved by an increase in recall. This is natural as the process involves checking more and more pairs, which automatically leads to an improvement in recall. It would however not be unthinkable to start with a rough partitioning that is continuously refined and would correspond to an increase in quality through growing precision. We have summarized these methods in Table 3.5 and reproduce the taxonomy by Papadakis et al. [1] in Figure 3.12. Because of its usefulness, progressiveness has been embraced as a central component in this thesis and is discussed in Chapter 6. Aspects that favor progressiveness are already inherent in some of the methods previously mentioned. For example in **Incrementally/Accumulative Adaptive Sorted Neighborhood**, The window size could be increased not just based on local conditions, but also on a globally increasing factor. This is indeed what is done in **Progressive Sorted Neighborhood** and even more so in **Dynamic Progressive Sorted Neighborhood**. In **Size-based Block Clustering**, the splitting and/or merging of blocks could be continued over time to achieve smaller or larger blocks (cf. Figure 3.13). The straight-forward result of this would be a **Hierarchy of Record Partitions** or the more memory-friendly **Ordered List of Records**. For *Dynamic Block Cleaning* methods like **Iterative Blocking** or **Block Scheduling**, it even seems that they already satisfy the requirements of Progressive Blocking. Finally, **P-RDS** adopts the blocking implemented by locality sensitive hashing to be progressive. Furthermore, the survey [1] distinguishes *Schema-agnostic Progressive Blocking* methods. **Local Schema-agnostic Progressive Sorted Neighborhood** substitutes attributes in attribute-value pairs by closeness in the alphabetical ordering of values. For all windows over the alphabetical order, sort the representations therein by co-occurrence frequency as an indicator of coreference-likelihood. **Global Schema-agnostic Progressive Sorted Neighborhood** improves this by skipping pairs that have already been compared. **Progressive Block Scheduling** clusters smaller blocks first, assuming that they are more concise. Within each block, higher weighted pairs are compared first. Weights might be computed as number of other blocks shared. It is not clear how **Progressive Profile Scheduling** avoids visiting all pairs during the blocking process when it computes the average comparison weight for each mention for prioritization. Also, if all pairs mentions are compared to a given reference mention before another reference mention is chosen, this does not seem to achieve a good global progression.

3.7.6 Parallel Blocking

In the survey [1], each family of methods (like *Block Building*) is accompanied by a description of how individual methods have been parallellized. Although this is often realized in the MapReduce framework, the details usually apply only to an individual method and are therefore not as interesting on a general basis. With respect to our matrix-based framework, Figure 3.14 shows how a sub-matrix of the target ${}^{x}\overset{x}{\boxplus}$ can be computed by taking some rows from ${}^{x}\overset{f}{\boxplus}$ and some columns from ${}^{x}\overset{f}{\boxplus}^{T}$. Using efficient matrix-multiplication algorithms, there should be some computational overhead involved

Schema-based	**Progressive Sorted Neighborhood (PSN)**	Apply an increasing window size w to a sorted list of mentions until termination. $\rightarrow$ *Straightforward, but should introduce a lot of redundancy. Only total alphabetical ordering.*
	Dynamic PSN	Adjust the processing order on the fly, based on where most duplicates have been found so far. $\rightarrow$ *Makes sense. Why use the arbitrary order from left to right when inter-window (in contrast to intra-window) order is free.*
	Hierarchy of Record Partitions (HRP)	Create a hierarchy of blocks that is resolved level by level, from the leaves, which contain the most likely matches, to the root. $\rightarrow$ *The straightforward result of consecutive merging of smaller blocks into larger ones.*
	Ordered List of Records	Turn the HRP into a list of records sorted by their likelihood to produce matches with neighbors, which gives a lower memory consumption. $\rightarrow$ *This probably means that we skip the redundancy of re-clustering blocks after merging. Not sure how this saves memory rather.*
	P-RDS	Make LSH-based blocking progressive by processing hash tables in the order of the number of found duplicates in previous buckets. $\rightarrow$ *LSH has multiple hash functions to capture similarity. This determines their order of application to find most collisions early.*
Schema-agnostic	**Local Schema-agnostic Progressive SN**	For each window size, order comparisons within each window over the sorted mentions according to the mentions' co-occurrence frequency. $\rightarrow$ *Alphabetical closeness for comparison decision, cooccurrence for order of comparison.*
	Global Schema-agnostic Progressive SN	Given a predetermined range of windows, eliminate all redundant comparisons within them. $\rightarrow$ *Should address the above mentioned redundancy problem of PSN, but not sure how.*
	Progressive Block Scheduling	Order blocks in ascending number of comparisons and prioritize all comparisons per block in decreasing weight. $\rightarrow$ *Process small blocks first and within them high-weighted comparisons. Not sure where the weights come from.*
	Progressive Profile Scheduling	Order mentions by decreasing average comparison weight and prioritize all comparisons per mention in decreasing weight. $\rightarrow$ *Need to go over all comparison weights first to get average. Determines the order of nonzero cells in a sparse comparison matrix.*

TABLE 3.5: Progressive blocking methods as presented in Papadakis et al. [1]



● selected filled cell (true) ○ selected empty cell (false) ○ ignored empty cell ● ignored filled cell

FIGURE 3.14: Basic batching for blocking matrix multiplication. Here, batch-size is 2.

with this computation of sub-matrices as the potential for time-saving shortcuts is more limited on a smaller matrix and ultimately all target cells are still computed. However, almost arbitrarily many cores can work on the product at the same time. Furthermore, the space consumption can be controlled – in particular if the target matrix is never computed fully at once, but verification is performed once a batch is complete and all non-verified duplicates are discarded.

3.7.7 Contextualizing our Approach

We have presented the outline of our approach in Chapter 2.2. In this section, we compare it to the other blocking methods described above and see how and to which extend it can be related to them by smaller modifications or by explaining it in the same framework. In the sparse matrix-view, we can describe our method as ${}^{x}\overset{k}{\boxplus}\cdot{}^{k}\overset{k}{\boxplus}\cdot{}^{k}\overset{x}{\boxplus}$, where ${}^{x}\overset{k}{\boxplus}$ maps mentions x to representation keys k, ${}^{k}\overset{k}{\boxplus}$ implements the subset/superset partial order over representations and ${}^{k}\overset{x}{\boxplus}$ is the transpose of ${}^{x}\overset{k}{\boxplus}$. In contrast to many Block Building methods, our approach does not focus on a certain type of features, although it is of course to be expected that it works better for some than for others. On the representations k, the subset/superset partial order constitutes a kind of **Sorted Neighborhood** by the cover relation. However this neighborhood is defined

Table 3.6: Characterization of non-learning block building methods as discussed in Papadakis et al. [1]. Distinguishing whether and how features are grouped by similarity and with which threshold, what features are extracted and whether a mention is represented by one or multiple features. If 1 billion mentions are feasible can only be guessed. We only check-mark those methods that control block size. The ability to be schema-agnostic comes from Papadakis' definition.

method	properties				scaling					adaptive					sound		accessible		
	feature similarity	feature extraction	feature mapping	feature similarity threshold	1 billion records	progressive	not pairwise	parallelizable	unsupervised	schema-un/aware	missing values	feature weighting	world knowledge	generalizes other	transitivity	basic tasks	simple	modular	displayable
Standard Blocking	equality	feature-combinations (keys)	1:1	boolean		✗	✓	✓	✓	✗	✗	✗	✓		✓		✓		✓
Suffix-Arrays	equality	suffixes of key	1:n	boolean		✗	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Extended SA	equality	infixes of key	1:n	boolean		✗	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Q-grams	equality	character q-grams of key	1:n	boolean		✗	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Extended Q-grams	equality	rare character q-grams	1:n	boolean	✓	✗	✓	✓	✓	✗	✗	✓	✗		✓		✓		
Sorted Neighborhood	lexicograph.	feature-combinations (keys)	1:1	constant window size	✓	✗	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Multi-pass SN	lexicograph.	different feature-types	1:1	constant window size		✗	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Extended SN	lexicograph.	different features	1:n	constant window size		✗	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Incrementally Adaptive SN	lexicograph.	different features	1:1	similarity-based window		✗	✓	✓	✓	✗	✗	✗	✗		✓				
Accumulative Adaptive SN	lexicograph.	different features	1:1	likelihood-based window		✗	✓	✓	✓	✗	✗	✗	✗		✓				
Duplicate Count Strategy	lexicograph.	different features	1:1	normalized likelihood window		✗	✓	✓	✓	✗	✗	✗	✗		✓				
DCS++	lexicograph.	different features	1:1	normalized likelihood window		✗	✓	✓	✓	✗	✗	✗	✗		✓				
Sorted Blocks	lexicograph.	feature-combinations (keys)	1:1	prefix-based		✗	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Sorted Blocks New Partition	lexicograph.	feature-combinations (keys)	1:1	prefix- and size-based	✓	✗	✓	✓	✓	✗	✗	✗	✗		✓				
Sorted Blocks Sliding Window	lexicograph.	feature-combinations (keys)	1:1	constant window size	✓	✗	✓	✓	✓	✗	✗	✗	✗		✓				
Improved Suffix-Arrays	lexicograph.	suffixes of key	1:n	similarity-based window		✗	✓	✓	✓	✗	✗	✗	✗		✓				
Token Blocking	equality	tokens of key	1:n	boolean		✗	✓	✓	✓	✓	✓	✗	✗		✓		✓		
Attribute Clustering Blocking	semantic	attribute-value pairs	1:n	constant		✗	✓	✓	✓	✓	✓	✗	✓		✓		✓		
RDFKeyLearner	distributional	attribute-value pairs	1:n	global	✓	✗	✓	✓	✓	✓	✓	✓	✗		✓				
Prefix-Infix(-Suffix)	equality	URL components + tokens	1:n	boolean		✗	✓	✓	✓	✓	✓	✗	✓		✓				
TYPiMatch	distributional	type + tokens	1:n	constant		✗	✓	✓	✓	✓	✓	✓	✗		✓				
Semantic Graph Blocking	taxonomic	mention node	1:1	max. path length		✗	✓	✓	✓	✓	✓	✗	✓		✓				✓
Progressive Sorted Neighborhood	lexicograph.	feature-combinations (keys)	1:1	growing window size	✓	✓	✓	✓	✓	✗	✗	✗	✗		✓				
Dynamic PSN	lexicograph.	feature-combinations (keys)	1:1	growing window size	✓	✓	✓	✓	✓	✗	✗	✗	✗		✓		✓		
Hierarchy of Record Partitions	open	open	open	none	✓	✓	✓	✓	✓	✗	✓	✓	✓		✓		✓		✓
Ordered List of Records	open	open	open	none	✓	✓	✓	✓	✓	✗	✓	✓	✓		✓				
P-RDS	collision	hashed	1:n	collision probability	✓	✓	✓	✓	✓	✗	✓	✓	✗		✓		✓		
Local Schema-agnostic PSN	lexicograph.	feature-combinations (keys)	1:1	none	✓	✓	✓	✓	✓	✓	✗	✗	✗		✓		✓		
Global Schema-agnostic PSN	lexicograph.	feature-combinations (keys)	1:1	none	✓	✓	✓	✓	✓	✓	✗	✗	✗		✓				
Progressive Block Scheduling	open	open	open	none	✓	✓	✓	✓	✓	✓	✓	✓	✗		✓		✓		
Progressive Profile Scheduling	open	open	open	none	✓	✓	✓	✓	✓	✓	✓	✓	✗		✓				

on a partial, not a total order. Progression is basically an incrementally increasing window (as in **Progressive Sorted Neighborhood**) over this partial order. In addition, our method can use some aspects of **Size-based Block Clustering**, when we use a redundancy factor on the edge weights between representations/blocks (see Chapter 6). As it is recommended to use some threshold (e.g. 5k or 10k) on the maximum block size to be clustered, in a way, **Block Filtering** is used in the progression when blocks get too large. In contrast to many other blocking methods, our approach is guaranteed to be disjoint, as we only merge blocks into larger ones (cf. Figure 3.13). The order of these merges is captured in an implicit **Hierarchy of Record Partitions**. We do not apply any *Filtering* or *Join-based Blocking* techniques. Our parallelization techniques studied for connected component search in the subset/superset partial order build on the basic idea displayed in Figure 3.14. In *Supervised Blocking*, one can learn a feature to key mapping ${}^{f}\overset{k}{\boxplus}$ that defines the blocking keys k in terms of included features f. We assume the source relation ${}^{x}\overset{f}{\boxplus}$ is fixed as it contains all the information available for any x. Then as described in Section 3.6, we assume with ${}^{x}\overset{k}{\boxplus}_{ij} = \prod_l {}^{x}\overset{f}{\boxplus}_{il} \geq {}^{f}\overset{k}{\boxplus}_{lj}$ that ${}^{x}\overset{f}{\boxplus} \star {}^{f}\overset{k}{\boxplus} = {}^{x}\overset{k}{\boxplus}$ tells us the representations that each mention x is a superset of, i.e. all the potential blocks that dominate it. We define ${}^{f}\overset{k}{\boxplus}$ mostly by the feature combinations that we observe in the data (and some of there generalizations). Note that ${}^{k}\overset{f}{\boxplus} \star {}^{f}\overset{k}{\boxplus} = {}^{k}\overset{k}{\boxplus}$ gives the superset partial order to be matrix-multiplied (dot-product) with ${}^{x}\overset{k}{\boxplus}$ in ${}^{x}\overset{k}{\boxplus} \cdot {}^{k}\overset{k}{\boxplus} \cdot {}^{k}\overset{x}{\boxplus} = {}^{x}\overset{x}{\boxplus}$.

3.8 Integrating Blocking- and Clustering Methods

This section is dedicated to the integration of blocking and clustering both in terms of technical interference and theoretical formalization. On the most general level, clustering follows blocking. The purpose of blocking is to make the clustering task cheaper. Many blocking methods return (either disjoint or overlapping) subsets of the collection. Each of these subsets can be considered a separate clustering task. If the subsets are disjoint, the result of blocking followed by clustering is a *modular decomposition* of the collection. If the subsets are overlapping, the transitive closure over all coreference-pairs determined by clustering must be taken in order to achieve an equivalence relation over the entire collection. Other blocking methods return not subsets of the collection, but pairs to be compared, for example a minimum spanning tree of a subset. Here, clustering is not applicable. Instead, a verification step applies a binary pairwise classifier that determines whether a given pair is coreferent – or not. Again, the transitive closure must be taken on the result to obtain an equivalence relation. Of course, any subset X of the collection also corresponds to a set of pairs $X \times X$, so the verification step could also be applied after a blocking methods that returns subsets. This would amount to single-link clustering, merging clusters depending on whether the distance of their closest elements is below the coreference threshold. Usually, other clustering methods produce better results as in one way or another they take into account multiple surrounding elements, not just two – that is they consider to some extend the *result* of their decisions.

3.8.1 Integration by Generalization

Ultimately, if it returns subsets, blocking is just a cheap clustering method. Then, clustering can be applied sequentially where each cluster on one level constitutes a new clustering task on the next level. So in theory is it also possible to have not just the usual two-step process, but multiple cascading rounds of clustering (i.e. with each new level deploying a more expensive clustering method on the results of the previous one). In Figures 3.15 we show a theoretical framework for clustering (and blocking) in general that is intuitively accessible. In this framework, a number of points from the input collection can be chosen as “connectors” by one property of the clustering method and are then copied to a second level. The second property of the method determines how to connect points to connectors. The result is a bipartite graph in which the transitive closure returns the equivalence relation over the entire collection. There is currently no obvious advantage of the graph being bipartite (i.e. very structured), except that the transitive closure should be particularly easy to compute. However, this does not exclude the possibility that the simplicity of the model may lead to practical insights

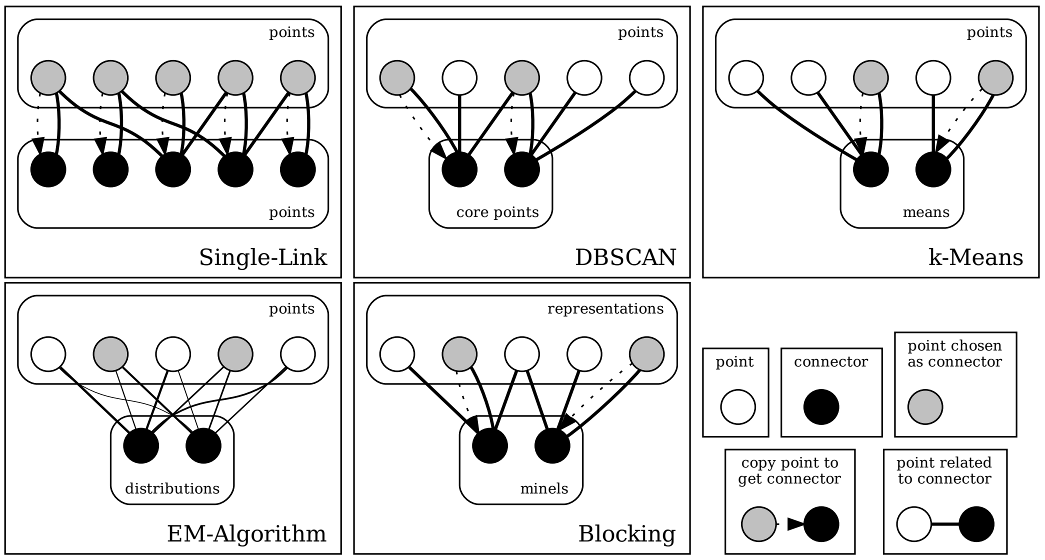


FIGURE 3.15: Clustering methods in the bipartite graph view. In **single-link** clustering, each point is copied to be a connector and there are relations from points to all close enough connectors. In **DBSCAN** all points that are central are copied as connectors (core-points) and there are relations from all points to close enough connectors. In **k-Means** random points are copied as initial connectors (cluster means). Relations are between points and the closest connector. Means are iteratively updated for all points assigned to the same mean (see also Figure 3.17). In **EM clustering** random distributions are initialized as connectors. Relations are between all points and all distributions and weighted by the likelihood of the point in the respective distribution. The distributions are iteratively updated according to these likelihoods. Ultimately, one may filter the relations to be those connecting each point to the distribution in which is has the highest likelihood. In **blocking** blocking keys are considered to be artificial points added to the representation space (e.g. singletons for each feature or larger sets AND-combining multiple features). Minimum elements are copied as connectors. Relations are built by the superset relation between points and connectors.

in the future that significantly reduce the complexity of relevant operations and related concepts or proofs. In **single-link clustering** each point is a connector and there are connections from each point to all close-enough connectors. In **DBSCAN** [40], all points with at least *minPts* points in their ϵ-neighborhood are connectors and there are connections from each point to all points in their ϵ-neighborhood. In **k-Means** [89], random points are initial connectors. There are connections from each point to its single closest connector. Here, the connectors are changed after each iteration to be the mean of their connected points (see also Figure 3.17). In **EM clustering** [90], connectors are not copied from the collection but are initialized as random probability distributions. However, usually the mean of these is taken to be a random point from the collection and the variance is chosen w.r.t. the distances in the collection. There are connections between all points and all distributions and these connections are weighted by the conditional probability of the point being produced by the distribution. The distributions are iteratively updated by according to the Expectation Maximization principle. After convergence, one would probably keep only the connections from each points to

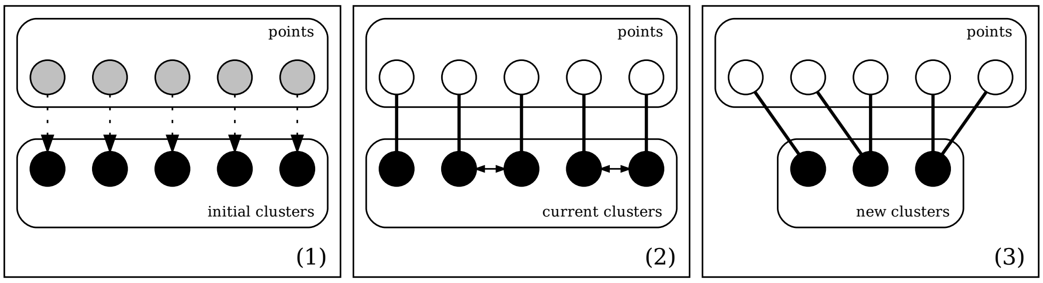


FIGURE 3.16: Average-link agglomerative clustering in the bipartite graph view.

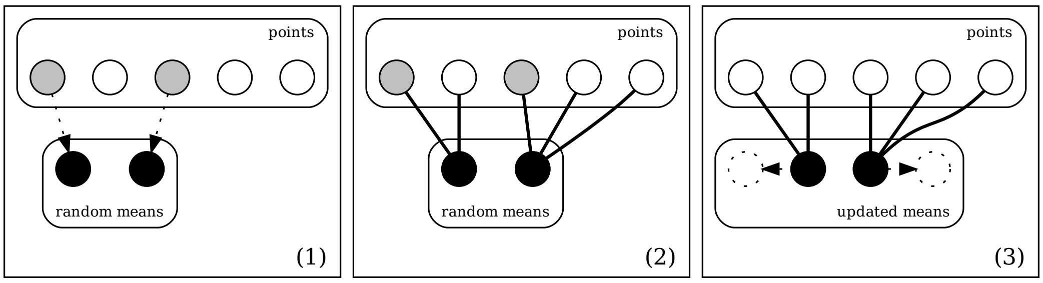


FIGURE 3.17: K-means clustering process in the bipartite graph view.

FIGURE 3.18: Iterative connector updates in hierarchical and K-means clustering.

the distribution under which it is the most likely. As we have stated above, blocking can be viewed simply a cheap clustering method. In this case, the following procedure allows to integrate both traditional blocking techniques as well as our proposed method: Blocking keys are added as artificial points to the collection. If the blocking key is a single feature, this amounts to a singleton representation. If the key is a conjunction, then this is a larger set. For example in author disambiguation, a popular key-scheme is *surname, first-initial*, which leads to keys like *J Smith*. The connectors are chosen to be minimal elements in the collection expanded by the artificial representations. These minimal elements would usually correspond to the keys. The superset relation then connects all points to the minimal elements that they satisfy in the sense that the features required by the connector are all present in the point's representation. In two of the works underlying this thesis, the clustering process has been implemented successfully by a probabilistic average-link agglomerative clustering. Average-link clustering is attributed to Sokal [91]. Figure 3.16 shows how the initial connectors correspond to all points in the collection (singleton clusters) and then clusters (i.e. connectors) are iteratively merged depending on the average link between them. When two connectors are merged, then the connections going from points to the individual connectors now go to the new connector. Our progressive blocking method performs the same kind of iterative operation on the blocking level.

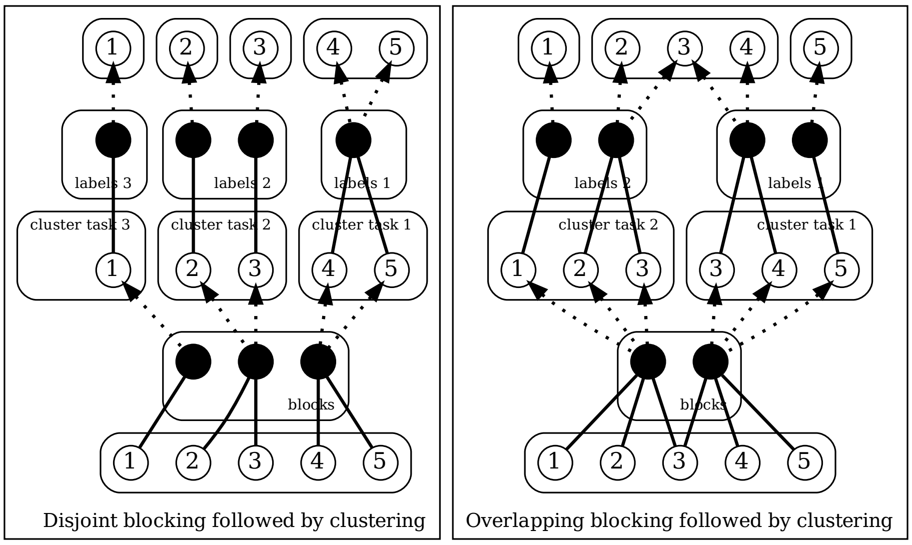


Figure 3.19: Relationship between blocking and clustering. Although blocking is simply a cheap clustering method, different separation steps can be concatenated to repeatedly break down large collections. On the left this is done by partitioning, leading to a *modular decomposition* while on the right overlapping clusters are allowed. On the left, connected components are computed (i.e. the transitive closure i.e. repeated matrix power M^k) while on the right, the matrix power is applied only once (M^2).

3.8.2 Integration by Combination

In Figure 3.19, we show how blocking and clustering are integrated. In disjoint blocking, each block corresponds to a completely separate clustering task. The clusters found in each of them correspond to the equivalence classes of the final equivalence relation. In non-disjoint (overlapping) blocking, each block is separately clustered, but the resulting clusters must be merged by the transitive closure if they share one or more points. If the blocking process returns pairs to be verified, then the block-connectors correspond to these pairs, i.e. each blocking connector connects exactly two points (see Figure 3.20). In the verification step, these are either confirmed or split up into two connectors. Then, we continue as with non-disjoint blocking by merging overlapping clusters.

In order to summarize integrative frameworks for blocking and clustering popular in the literature, we consult an additional survey by Christophides et al. [11]. In Papadakis et al. [1], the entity resolution pipeline is defined as

$$\text{Block Building} \rightarrow \text{Block Cleaning} \rightarrow \text{Comparison Cleaning} \rightarrow \text{Filtering} \rightarrow \text{Verification}$$

where *Block Cleaning* and *Comparison Cleaning* both go by the general term *Block Processing*. In Christophides et al. [11], the pipeline is

$$\text{Blocking} \rightarrow \text{Block Processing} \rightarrow \text{Matching} \rightarrow \text{Clustering}$$

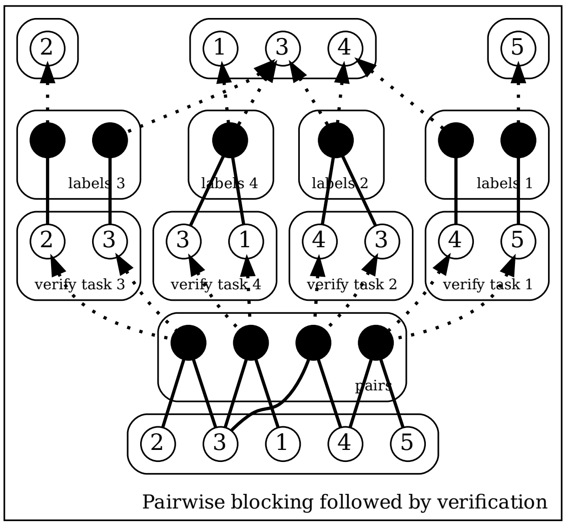


FIGURE 3.20: Relationship between blocking and verification. This is visualized in the same framework as blocking followed by clustering (see Figure 3.19).

suggesting that *Blocking* is the same as *Block Building*. Here, *Filtering* is not considered. *Matching* refers to a pairwise classification of coreference, which should be the same as *Verification*. *Clustering* actually is a misnomer here, because it simply refers to the transitive closure over the matching decisions. As has been pointed out earlier, there is a difference between the transitive closure over pairwise classifier decisions and a "real" clustering process because the latter usually takes into account multiple points to consider to some extend the consequences of its decisions. Figure 3.19 shows on the left how disjoint blocking is followed by a separate cluster task for each block that outputs the final mention clusters considered coreferences. On the right, overlapping clusters make it necessary to keep track of identical mentions in different clusters to later apply a transitive closure over the final labelling and merge clusters with overlapping mentions. Figure 3.20 has a minimal block of size two for each pair suggested by blocking. Upon verification, the coreference is confirmed, otherwise revoked. Again the transitive closure over the final labelling is required to obtain coreference partitions.

In addition to this pipeline arrangement of blocking and clustering or verification, there is a distinction between such classic ER frameworks and those that take a more "integrative" approach. In Figure 3.21, we give a schematic overview of *Static Blocking*, *Dynamic Blocking* and *Real-time ER*. In Papadakis et al. [1], this refers to "static matching awareness", "dynamic matching awareness" and "Real-time Entity Resolution". In Christophides et al. [11], the latter is called "incremental ER", whereof *Dynamic Blocking* is considered a special case along with *Dynamic Matching* and *Dynamic Clustering*. However, we think that there is a clear difference between *Dynamic Blocking* and *Real-time ER*. In *Static Blocking*, the blocking step passes information in the form of blocks to the clustering step, but the latter does not send any information back. Instead, if

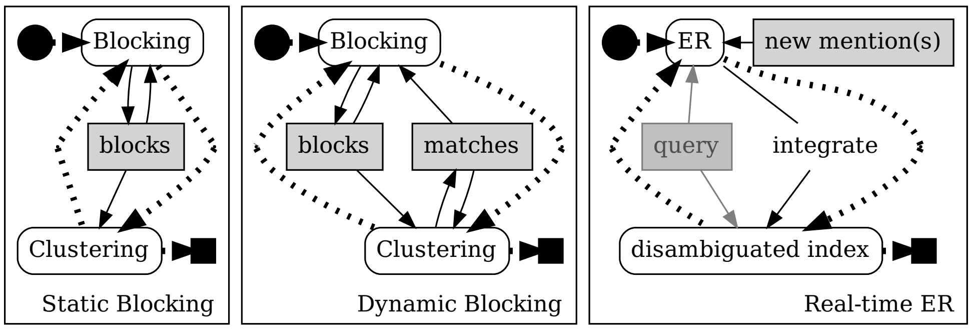


FIGURE 3.21: *Static*, *dynamic* and *real-time* blocking.

Progressive Blocking is performed, the next Blocking iteration only considers its previous output as a starting point to compute the next (larger) blocks. In this case, the one-way pipeline of blocking and clustering is repeated until termination. In *Dynamic Blocking*, the same principles are applied, with the exception that the clustering step feeds back its matching decisions to be considered by the blocking step as an additional input on top of the previous blocking state. Perhaps the clustering step would also consider its previous matching decisions in addition to the new blocks it receives from blocking. Finally, in *Real-time ER*, a completely different setup is assumed. Here, the ER method is continuously presented with a batch of new entity mentions, which it has to integrate into the existing disambiguated collection. In some cases, it may be allowed to query the collection for some context information before sorting the new mentions into the index (cf. *query-based* in Christophides et al. [11]) or it has to go directly to the correct point in the index (cf. *streaming*, ibid.). The dynamicity here lies in the source itself, which forces the ER process to adapt in a dynamic fashion. The main challenge is the efficient implementation of index updates. As this is a very technical question and tightly bound to the deployed technology, it is not useful to go into great detail here. Generally speaking, it requires a pairwise classifier to connect new mentions to the collection and a cheap update of the transitive closure – that is if redundant operations are to be avoided. Otherwise, a simple means is to re-cluster all blocks that at least one new mention has been added to. Coherent integration and resolution in the light of the mutual interdependence of blocking and clustering decisions inherent in the transitive closure of matching decisions (as in *Dynamic Blocking*) presents the greatest challenge to *Real-time* ER. In our work, we assume a static blocking (note that this does include *Progressive blocking* in our view) but acknowledge that only a Real-time ER method can offer full usefulness in a real-world Big Data resolution scenario that is very likely to include a dynamic source of entity mentions. To this end, one should note that even more challenges like the “un-learning” of decisions and facts in the context of privacy-related forget requests may apply [92].

Chapter 4

Clustering: Effective Unsupervised Author Disambiguation with Relative Frequencies

In this chapter, we explore clustering for entity resolution based on the example of grouping author mentions into real-world author entities. This work was undertaken as a first project to explore the task of author disambiguation. The goal was to device a simple method that can be implemented to run at relatively large scale and achieves good performance. Fortunately, it was possible to use the *Web of Science* (*WoS*) as a very large high quality database that even contains a large number of author annotations in the form of researcher-ID's. A focus was put on developing a useful similarity measure for mention-mention and cluster-cluster similarity in an agglomerative clustering framework. In addition to the clustering method itself, this work has given hints towards the importance of the blocking scheme, the peculiarities of name-matching, the importance of a good stopping criterion in the context of the development of the (hidden) precision-recall curve, the potential deceptiveness of plain performance numbers and the necessity of a large-scale annotated benchmark for comparing different author disambiguation methods. The method presented in this work was later re-implemented by Tekles and Bornmann [93] and compared to other methods. It was found that the stopping criterion does not work as well for very large problem sets, but that in general, the similarities perform best among a number other non-rule-based approaches.

4.1 Overview

A similarity measure was invented that has the properties of a conditional probability $p(x|y)$. This means that it is not symmetric, which is why implicitly, the maximum of $p(x|y)$ and $p(y|x)$ is used, i.e. it is sufficient if the similarity in any direction is below the threshold, and possible merges are sorted regardless of the direction. Whether a merge of x and y is due to $p(x|y)$ or $p(y|x)$ is obviously irrelevant for the result. The conditional probability considers the expressiveness of a feature by automatically weighting rare features higher and it aggregates mention-mention probabilities in an additive way. The advantage of additive over multiplicative aggregations (and conditional over joint distributions) is that it is less likely to produce zero or extremely small probabilities. Also, it can be better implemented as a dot product over matrices. A general advantage of a probabilistic measure is that a simple consistency test can be done by adding up all probabilities to check if the result is one. This is a necessary condition for the measure to be implemented correctly and is extremely unlikely to happen due to chance (i.e. it can be considered sufficient for validity). During the deployment of the method, a general suspicion as to whether the results were really as good as measured led to the development of a visualization technique that draws the agglomerative clustering process in terms of precision and recall development as well as the number of clusters. A crucial parameter turned out to be when to stop merging clusters as the similarity is powerful for determining best merges, but does not assess itself as to whether it is high enough. Furthermore, suspicion regarding the comparability of results reported in various related works let to the second focus of the work on investigating possible evaluation pitfalls or ways to manipulate experiments to produce good results for any method. A central aspect in this context is the blocking scheme and the aggregation of the results over all blocks. A blocking scheme that produces small blocks can lead to good results for any clustering method if performance is measured for each block separately (e.g. "How well can we disambiguate J. Smith? How well R. Myers? T. Powell?", etc.) and then aggregated without considering the individual problem sizes (macro average instead of micro average). Using the blocking scheme of *surname,all-initials*, most blocks only contain one annotated author, which means a trivial baseline combining all mentions in a block performs very well. As a consequence, the clustering method was evaluated individually for different problem sizes, where problem size is measured by number of distinct authors. This information is obviously only available to the investigator, not to the clustering method. For larger problem sizes (i.e. more than one real-world author), the trivial merge-all baseline is not competitive. Some experiments were also conducted to obtain insights on the contribution and usefulness of individual features like co-author names.

4.2 Terminology and Notation

In the following we list and explain the basic terminology used in the paper and the notation that was used in this context at the time.

$\mathfrak{N}$ author names, i.e. blocking representations, i.e. blocks, e.g. *J. H. Doe*

X set of all mentions in a certain block, with a certain name

x author mention

$\mathfrak{C}$ clusters, i.e. a partitioning of mentions X

C author, i.e. set of mentions x, i.e. a partition in $\mathfrak{C}$

$F(x)$ bag of features for mention x

f a feature f

$\#(f, x)$ how many times feature f is observed with mention x, basis for all probabilities

$\#(f)$ how many times we have seen feature f overall (either in name or in collection)

$\#(x)$ how many features mention x has

$\#(C)$ how many features cluster C has through its mentions

l stopping criterion, i.e. threshold

α, β parameters of linear function for stopping criterion

$p(C|\dot{C})$ probability of cluster C given cluster $\dot{C}$

$p(x|\dot{x})$ probability of mention x given mention $\dot{x}$

$score(C|\dot{C})$ asymmetric similarity of two clusters aggregated over all feature-types

$ftype$ a feature-type, e.g. *co-author names*, where a feature is e.g. *J. H. Doe*

$p_{ftype}(C|\dot{C})$ probability for one specific feature-type

$p(x|C)$ probability of a mention x to be in a cluster C

$\mathfrak{C}_{cor}$ correct clustering, i.e. gold partitioning of mentions in a block

$\mathfrak{C}_{sys}$ system clustering, i.e. computed partitioning of mentions in a block

$pairs(\mathfrak{C})$ all pairs $(x, \dot{x})$ in a mutual cluster $C \in \mathfrak{C}$

P_{pairF1},R_{pairF1} precision, recall in pairF1 measure

P_{bCube},R_{bCube} precision, recall in bCube measure

4.3 Publication

Title Effective Unsupervised Author Disambiguation with Relative Frequencies

Authors Tobias Backes

Document Type Conference Paper

Venue Proceedings of the 18th ACM/IEEE Joint Conference on Digital Libraries

Copyright © 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.

DOI 10.1145/3197026.3197036

MLA Backes, Tobias. "Effective Unsupervised Author Disambiguation with Relative Frequencies." Proceedings of the 18th ACM/IEEE Joint Conference on Digital Libraries. 2018.

APA Backes, T. (2018, May). Effective Unsupervised Author Disambiguation With Relative Frequencies. In Proceedings of the 18th ACM/IEEE Joint Conference on Digital Libraries (pp. 203-212).

ISO 690 BACKES, Tobias. Effective Unsupervised Author Disambiguation with Relative Frequencies. In: Proceedings of the 18th ACM/IEEE Joint Conference on Digital Libraries. 2018. S. 203-212.

4.4 Discussion

4.4.1 Probabilistic Similarity Measure

There is a small mistake in the paper as in joint probability discussed in Section "Variations" is never actually defined. It should be simply

$$p(C, \dot{C}) = p(C|\dot{C}) \cdot p(\dot{C})$$

where $p(\dot{C})$ was defined in the paper. It turned out that the conditional probabilities offer a much better estimate of cluster similarity than the joint probability, the reason for which was guessed to be the fact that the factor $p(\dot{C})$ rewards merging large clusters, which establishes a self-reinforcing tendency because merged clusters are larger and are then more likely to be merged again, and so on. Instead of taking the sum over all $p(x|\dot{x})$ to aggregate $p(C|\dot{C})$, the max could also be taken, implementing a single-link clustering. However the results were not better and the computational cost was higher as it cannot be implemented in basic matrix multiplication. The smoothing parameter ϵ was not tuned, which has been criticized in the reviewing process. However its only purpose was to prevent zero-probabilities. A faster dynamic-programming implementation of the conditional probabilities was later developed and is presented in Chapter 6. Here, also a slight modification was introduced where we multiply $\#(f)$ by 1000 to prevent very small values, which led to better performance. As $p(C|\dot{C})$ is computed over all pairs $x, \dot{x} \in C \times \dot{C}$, the score for a merge of clusters C and $\dot{C}$ cannot be computed for very large clusters due to quadratic complexity. For example two clusters of size 10k require 100M comparisons, which should be roughly the borderline of what is feasible. Remember that this has to be computed for *all* possible cluster pairings. Due to the sparsity of the mention-feature matrix $\#(\vec{f}, x)$ underlying $\#(f, x)$, using the sparse dot-product of $\#(\vec{f}, x) \cdot \#(\vec{f}, x)^T$ should improve the feasibility considerably, although this does not affect the theoretical worst case complexity.

4.4.2 Feature Types and Weights

Eight feature-types were used and listed in the paper. Their selection was not tuned but based on common sense and what was available in the data. We can add the following grouping of feature types according to their rough function for determining the likelihood of two mentions referring to the same real-world author:

Topics *Terms* and *Categories*

Strong author indicators *Affiliation* and *Email*

Peers *Coauthornames* and *Refauthornames*

Mostly irrelevant *Keywords* and *Publication Year*

It was attempted to learn a weighting of these feature-types with respect to their importance for disambiguation. For this purpose, eight-dimensional vectors were built with the conditional probabilities $p_{ftype}(x|\dot{x})$ for each feature-type. From the researcherID annotation, a balanced number of mention-cluster pairs (x, C) to be classified as *belonging* or *not belonging* were sampled. The weights learned by a linear classifier were used as feature-type weights. The following weighting can be assumed to reflect the importance of the individual feature types:

- 33% Topics: 15% Terms and 18% Categories
- 30% Author indicators: 20% Affiliation and 10% Email
- 32% Peers: 20% Coauthornames and 12% Refauthornames
- 5% Mostly irrelevant: 3% Keywords and 2% Publication Year

So the relevant groups of feature-types receive roughly one-third attention. These findings were also supported in an application-based setting by leaving out individual feature-types when performing the author disambiguation task. As a uniform distribution of weights did not give worse results, it was supposed that the inherent weighting of individual features by $\#(f)$ makes feature-types unnecessary. However as the optimal weighting of the selected features happens to be relatively even, this might not be true for all datasets. It can be guessed that using word embeddings instead of terms to detect topical similarity would be better, and in fact, the formula could be adopted to make use of such semantic features, but this was not explored in the work.

4.4.3 Stopping Criterion

While the similarities are surprisingly effective in determining coreference likelihood of authors, the stopping criterion, i.e. the point from which on coreference is unlikely turned out to be quite difficult to determine – especially with these particular probabilistic similarities that have a very different scale depending on the number of mentions to be clustered due to the requirement of summing to one. For problem sizes with up to ten real authors, the stopping criterion's linear dependence on the number of mentions was acceptable, although it could already be noticed that it triggered a little late for size-10 problems. In a later study by Tekles and Bornmann [93], it was found that for larger problems this shift continues, making the stopping criterion a clear weak point of the model. Therefore they specifically trained a stopping criterion for various ranges of number of mentions to be clustered. In a later work (see Chapter 6), we tried to device a non-linear function with three parameters. This worked a little better but was very tedious to tune. Hence, the problem of finding a good estimate of when the clustering is optimal remains.

4.4.4 Blocking Scheme

In the paper, the *surname,all-initials* blocking scheme was used, which produces relatively small blocks. Another popular scheme produces much larger blocks separating mentions by *surname,first-initial*. Neither of those really models the matching possibilities between different name-based representations. For example *J. Herbert Doe* and *John H. Doe* match. When this project was undertaken, the relevance of such matching relations became apparent, but was not yet formalized well. This was taken up in the second and third work (see Chapters 5 and 6).

4.4.5 Evaluation

In the paper, it was noted that at least five major steps in which experiments of author disambiguation can differ:

1. data
2. annotation
3. blocking
4. disambiguation
5. evaluation

When comparing different AD methods, usually only the disambiguation step should be changed. When comparing different blocking methods, only the blocking step should be changed. However, when comparing experiments from the literature, it becomes clear that all of the above aspects differ or may differ (i.e. not all details are reported). For example in our evaluation, we count coreference labels for all mention-mention pairs including the obvious (x, x) and $(x, \dot{x})$ when $(\dot{x}, x)$ is already recorded. This corresponds to taking the entire mention-mention matrix into account, while Levin et al. [94] suggest using only the upper diagonal, which however is not defined for size-one clusters. A more detailed discussion of comparability issues can be found in the paper in Chapter 6. Currently no large-scale benchmark exists that keeps data and annotation fixed.

Blocking techniques could be reproduced relatively easily and evaluation measures could also be standardized if attention is payed to details like micro- or macro average. In principle the Web of Science could be used to create a constant benchmark as it is of large scale and has a large number of annotated author identities. However, this would require to create a snapshot of the continuously changing collection. Such re-processing is impeded by the proprietary nature of the data, which also makes it impossible to share the benchmark. Consequently, the only option is currently to re-implement existing methods, which is very time consuming and error-prone. Regardless, Tekles and Bornmann [93] have taken on this challenge and compared a number of methods along with our proposal. In the paper presented in this chapter, another approximate solution was found. By separating results by true problem size (number of real author identities in the current block), the dependence on the factors data and annotation is reduced and shifted towards a collection independent measure of difficulty, i.e. the number of authors that would have to be distinguished in a perfect result. The goal was to encourage other groups to also report results in this way, so that some comparability is achieved. Of course, this is not failproof as the size-distribution in terms of number of mentions over the different real authors also matters. For example if most correct clusterings have only one large cluster while the remaining ones are of minimal size, then the difficulty in terms of distinguishing coreferent and non-coreferent pairs is actually similar to having just one real author. Our evaluation in this paper can be criticized retrospectively for only showing results of problems with up to ten distinct authors. There are some names for which there are thousands of different annotated authors. A bigger problem is that the recall estimates were only taken within each block, i.e. missed coreferences across blocks were not recorded. The reason is the growing computational complexity of finding them. The task of evaluation can be broken down to finding the number of true pairs (T), positive pairs (P) and true-positive pairs (TP). P can be computed easily as the sum over the squared system cluster sizes, likewise T as the sum over the squared gold cluster sizes. TP however is more expensive to compute as for each pair a comparison is required (e.g. check for each positive pair whether it is true). Later, some tricks have been developed to make this more feasible. These are described in Chapter 7. From our work in Chapter 5 we know that the blocking-based recall for the problem sizes 1-10 and the all-initials scheme is 96%, i.e. 4% of the pairs that should be found are missed in T for this paper as they cross block boundaries. For larger problem sizes, this goes down to almost 90%, which makes the disambiguation task seem considerably easier than it actually is. A good compromise is to compute recall within *surname,first-initial* blocks as this is as high as 98% across all problem sizes.

Chapter 5

Matching and Blocking: The Impact of Name-matching and Blocking on Author Disambiguation

In this chapter, we model the relationship between matching and contradictory blocking representations based on the example of author mentions in the Web of Science. The underlying work was a result of tackling open questions arising from the realization of the blocking step's importance in the previous work. The literature on AD blocking was very limited. It was unclear to which extend the disambiguation task is already solved by blocking but first indications (i.e. the merge-all baseline) pointed into the direction that blocking and clustering follow the Pareto principle in that blocking is the proverbial 20% effort to achieve 80% of performance (in fact the ratio is more extreme). It was apparent that blocking in AD was always done based on name-components such as surname(s), first names and first name initials, and that different levels of name specificity (in the sense of presence of those components) were available in the data and required in the blocking schemes. However, there was hardly any literature describing the distribution of such levels of specificity in datasets and none describing the relationship between different name representations or blocking schemes in general. This was despite the fact that it was clear such relationships exist and are non-trivial as they do not form an equivalence relation, for example *John Doe* and *J. H. Doe* matching and *John Doe* and *Jack Doe* being contradictory although sharing the generalization *J. Doe*. The goal of the paper was to (a) shed a light on the distribution of completeness of name representation in the actual data (again, the WoS was used) and (b) uncover the matching relationship between different blocking schemes. It was only natural to also compare the performance of different blocking schemes in this context.

5.1 Overview

In the proposed model for name-based blocking, relationships between different schemes (uninstantiated name representations) and between matching names (instantiated representations) are the same. Not all feature-combinations make sense. For example it makes no sense to consider a first name to be present but not the corresponding initial.The nodes in the matching graph in Figure 2 were established starting from the most general assumed representation *surname, first initial* using the COMPLETE and the ADD relation, adding first name information if only an initial was present or another initial, respectively. This allowed to order different name representations in a directed acyclic graph as in Figure 3, and to identify existing blocking schemes in this graph, as well as to define new schemes based on the observations. In general, a blocking scheme was defined to be a set of nodes to be isolated by removing all edges to and from it. For example the *surname, first initial* scheme can be implemented by isolating all nodes that do not have a surname and a first initial. *J. Doe* and *K. Doe* would thus be separated by isolating *Doe*. Unfortunately, the *all-initials* scheme cannot be implemented in this way, but has to be computed differently. Nevertheless, it was easy to compare different blocking schemes against the WoS gold annotation in this framework. This answers both the question of which percentage of the overall disambiguation performance is already achieved by blocking and which blocking scheme is the most effective in terms of precision and recall. An additional new type of blocking scheme was proposed on the basis of frequency information annotated in the graphs of name representations. If a name has an even distribution of specifications, then it is isolated, because intuitively, there is a lot of confusion regarding the description of the people going by those names, which is likely to be the result of most of them being separate individuals. For example, if *J. Doe* has a lot of specifications like *John Doe*, *Jinger Doe*, *Jenna Doe*, *J. K. Doe*, *J.Y. Doe*, then these are likely to correspond to a relatively even distribution of observations and to mostly different people. On the other hand if *Jinger Doe* only has one specification *Jinger A. Doe*, then this is likely to describe the same person. This is also an indirect measure of the "parent's" specificity in the data, because unspecific names will have more observed specifications. The evenness of the distribution was measured by entropy, which was modified in later works to be a conditional distribution corresponding to the *confidence* measure in association rule learning. The main research questions were to compare blocking schemes against each other and to assess their contribution to disambiguation in absolute terms as well as to determine to which extend name-matching could be used as a replacement for author annotation. To this end, three evaluation scenarios were undertaken by comparing blocking against gold annotation, blocking against matching and finally matching against gold annotation.

5.2 Terminology and Notation

In the following we list and explain the basic terminology used in the paper and the notation that was used in this context at the time.

$\mathcal{N}$ set of all names N

N Name, a set of mention x

x mention

X set of all mentions

$\mathcal{A}$ system authors (clusters)

$\mathcal{G}$ gold authors (clusters)

$\mathcal{B}$ set of blocks B

$\mathcal{M}$ matching pairs (not a partitioning)

B block, set of mentions x

PARENT cover-relation in subset partial order of name representations (ADD + COMPLETE)

MATCH all pairs of non-contradictory name representations

BLOCK the pairs $B \times B$ for all $B \in \mathcal{B}$

GOLD the pairs $G \times G$ for all $G \in \mathcal{G}$

ADD those pairs of PARENT where a first name is added

COMPLETE those pairs of PARENT where an initial is added

H entropy

$C(N)$ covers of a name/node N in PARENT

$p(N)$ distribution of carry counts over the node N's covers

$\mathcal{P}$ matrix, reflexive closure of PARENT

$\mathcal{Q}$ matrix, reflexive, symmetric closure of PARENT

$\mathcal{P}'$ matrix, edges removed from $\mathcal{P}$ according to blocking scheme

$\mathcal{Q}'$ matrix, edges removed from $\mathcal{Q}$ according to blocking scheme

${}^{\mathcal{B}}/_{\mathcal{M}}$ blocking evaluated against matching

${}^{\mathcal{B}}/_{\mathcal{G}}$ blocking evaluated against gold identities

${}^{\mathcal{M}}/_{\mathcal{G}}$ matching evaluated against gold identities

S blocking scheme (except [inits]), i.e. names to be isolated

γ problem size in terms of # distinct rID's

$\mathcal{N}_\gamma$ set of all names with problem size γ

5.3 Publication

Title The Impact of Name-matching and Blocking on Author Disambiguation

Authors Tobias Backes

Document Type Conference Paper

Venue Proceedings of the 27th ACM International Conference on Information and Knowledge Management

Copyright © 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.

DOI 10.1145/3269206.3271699

MLA Backes, Tobias. "The Impact of Name-matching and Blocking on Author Disambiguation." Proceedings of the 27th ACM International Conference on Information and Knowledge Management. 2018.

APA Backes, T. (2018, October). The Impact of Name-matching and Blocking on Author Disambiguation. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management (pp. 803-812).

ISO 690 BACKES, Tobias. The Impact of Name-matching and Blocking on Author Disambiguation. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. 2018. S. 803-812.

5.4 Discussion

5.4.1 Erratum

There is a typo in the top-right corner of page 807, where “complete and initial” should read “complete an initial”. There are further mistakes in the captions of Figures 8, 9 and 10. In Figure 8, it should read “compared to **GOLD** relation”. In Figure 9, it should read “compared to **MATCH** relation” and the words “and complexity” should be cancelled. In Figure 10, the words “and complexity” should be cancelled.

It seems very strange that [inits] has 90% precision in Table 3 for surnames with more than 500 distinct rIDs, where we expect much more synonymy, even with up to three initials for such frequent names. The same problem seems to apply for smaller problem sizes as well as the most frequent surnames in Table 4. These questionable numbers also appear in the plots in Figures 7 and 8, making this a quite profound problem. These suspiciously high numbers also correspond to a strange mismatch compared to the paper in Chapter 4, where the precision of all clusters being merged (Figures 6 and 7) is around 30-40% (here [inits] has been used as a blocking scheme) – while the numbers in this paper are 99-100%. In fact, we cannot reproduce the numbers reported in this paper. It seems likely that the results have been created separating problem sizes by number of rID annotations rather than number of *distinct* rIDs. In this case, we can reproduce the results with small deviations that might be due to negligible factors. In Table 5.1, we report the new numbers for the [inits] scheme and the [f1] scheme both using *distinct* rIDs and number of annotated rIDs for problem size.

The way the implementation of the evaluation was described in the paper is unnecessarily complicated. instead, the following formulas can be used, which correspond to the following *SQLite* queries, with P the number of positive pairs, T the number of true pairs and TP the number of true-positives. These values are aggregated over all surnames

Table 5.1: New numbers computed with number of distinct rIDs rather than number of rID annotations

	f2		***inits***		
sizes	**P**	**R**	**P**	**R**	**#**
1-10	96	98	99	91	123k
11-25	72	99	86	91	1349
26-50	59	99	81	92	387
51-100	38	99	64	89	186
101-250	25	99	53	86	105
251-500	12	99	23	77	21
501-1000	4	99	11	76	8

where the number of *distinct* rIDs is in the respective range, and then precision is computes as $\frac{TP}{P}$ and recall as $\frac{TP}{T}$.

$$P = \sum_{B \in \mathcal{B}} |B|^2$$

```
SELECT SUM(size) FROM (SELECT pow(COUNT(*),2) AS size FROM names WHERE l=
$surname AND rID IS NOT NULL GROUP BY i1,i2,i3)";
```

$$T = \sum_{G \in \mathcal{G}} |G|^2$$

```
SELECT SUM(size) FROM (SELECT pow(COUNT(*),2) AS size FROM names WHERE l=
$surname AND rID IS NOT NULL GROUP BY rID)";
```

$$TP = \sum_{B,G \in \mathcal{B} \times \mathcal{G}} |B \cap G|^2$$

```
SELECT SUM(size) FROM (SELECT pow(COUNT(*),2) AS size FROM names WHERE l=
$surname AND rID IS NOT NULL GROUP BY rID,f1,f2,f3)";
```

The surnames are selected as follows:

```
SELECT l FROM (SELECT l,COUNT(DISTINCT rID) as freq FROM names GROUP BY l)
WHERE l IS NOT NULL AND freq BETWEEN $fro AND $to;
```

where we assume that before, the DISTINCT was essentially dropped.

It can be expected that using distinct rIDs, the results in general are shifted to much lower numbers but that the comparison between the different schemes remains the same as the methods and their performance are probably not different, only the aggregation of the results by problem size. Note that the isolation-based formulation of blocking schemes in this paper has an advantage over a positive formulation, in that it *separates* all names that are more general than the required scheme, while a positive formulation iterating over all variations of the scheme's features (as above over first, second and third initial i1,i2,i3) merges them all in one block (i1=NULL, i2=NULL, i3=NULL). However in the [inits] scheme, this does not make a difference because only names with only surname(s) are not covered by the scheme. It would however make a difference for schemes f3 and f4. It was also criticized in this paper that one can only guess what happens in other works to mentions with names that do not meet the minimal requirements. At least in this work, they are *not* discarded.

5.4.2 Shortcomings and Later Improvements

When the paper was written, it was neither noticed that the resulting graph was a subset of the subset/superset partial order over name representations, nor that the presented graph for uninstantiated representations forms a lattice and the instantiated one forms a semilattice. Neither was the relationship to formal concept analysis and association rule learning discovered yet. For the sake of blocking, intermediate hypothetical nodes between observed representations are actually unnecessary as they do not change the connectedness nor the edge weights in the graph. Hypothetical common specifications of observed representations should not be created as there are simply too many possible unions of observed representations. In formal concept analysis, this is usually done to create a lattice structure, but it is only feasible over a small vocabulary. It is generally desirable to create such hypothetical specifications as they show the logical consequences of assuming equivalence of two matching representations. For example if we say that *John Doe* and *J. Herbert Doe* refer to the same person, than the true name would be at least as specific as *John Herbert Doe*. If we can falsify this hypothesis, then the two cannot be merged. But this has not been realized even in follow-up work due to the above-mentioned complexity problems.

What is referred to as the PARENT relation in this paper is actually the the covering relation known from lattice-theory. We cite the Wikipedia definition:

> *In mathematics, especially order theory, the covering relation of a partially ordered set is the binary relation which holds between comparable elements that are immediate neighbours. The covering relation is commonly used to graphically express the partial order by means of the Hasse diagram.*
>
> – Wikipedia article on *Covering relation*[1]

what is called “cover” in this paper is a misnomer because it means almost the opposite of the above definition. Instead, we have called it “carry” as a representation that carries another one lies below it. This leads to the fact that the plots should be inverted top to bottom with the most general representations at the bottom and the most specific at the top to reflect the conventional mathematical notion.

Finally, the terms “parent” and “children” as used in this paper are potentially misleading as they suggest that the DAG is a tree, while if anything, it is a semilattice. However, we are also not aware of better terms for the general case of a DAG.

[1] https://en.wikipedia.org/wiki/Covering_relation, retrieved June 16th 2021

In this paper, have not yet used progressive blocking. All the blocking schemes derived from the name-matching model are fixed. However, it is only a small step from this model to introduce progressivity by iteratively merging nodes with edges indicating a tight correlation. In contrast to assuming all names as separate blocks and merging correlated nodes, isolating nodes in an otherwise connected graph as done in this paper is less intuitive. This is particularly striking as the [inits] scheme could not be implemented as a set of nodes to be isolated anyway.

5.4.3 Evaluation

Three major experiments, called $^{\mathcal{B}}/_{\mathcal{G}}$, $^{\mathcal{M}}/_{\mathcal{G}}$ and $^{\mathcal{B}}/_{\mathcal{M}}$ are done. $^{\mathcal{B}}/_{\mathcal{G}}$ compares pairs in the same block against pairs in the same gold author clustering and is the normal evaluation of a blocking scheme against a gold standard. Here we have computed new numbers as it seems that in the paper the number of rID annotations was used and not the number of annotated authors. $^{\mathcal{M}}/_{\mathcal{G}}$ compares matching pairs against pairs in the same gold author clustering and measures the usefulness of matching as a proxy-annotation or for semi-supervised learning. $^{\mathcal{B}}/_{\mathcal{M}}$ compares pairs in the same block against matching pairs and measures by recall the matches missed by blocking and by precision the dilution of the match relation by the transitive closure over the (mathematical) covering relation (covering is a subset of matching). In retrospect, it seems that this evaluation is not very useful at least regarding recall, as there are many matches that are not really desirable to model. For example, the more popular a surname is, the more likely it is that there is at least one name with *only* the surname specified. This representation matches with all other representations that are more specific. So a reasonable blocking method which isolates the *surname-only* representation will miss a lot of matches, but these matches were not important to model in the first place.

We now go through the six research questions from the paper and check if their answers as given in the paper's conclusion section have passed the test of time.

RQ1: How to structure names, their variations and relations? The model reflected in Figure 2 of the paper is still valid and very useful. It has also been generalized to any kinds of features. Some more conventions and relations to other (mathematical) theories and concepts have been discovered, but this mostly concerns the terminology.

RQ2: How to include name-matching in the blocking? The basic theory has not been changed here. However, blocking schemes are now defined by than the set of nodes to be merged rather than by the set of nodes to be isolated.

RQ3: How do different blocking schemes perform? Although we have corrected the compared blocking schemes' results in absolute terms, we assume that their relative ordering remains the same. Therefore the statements regarding their relative performance are probably still true.

RQ4: Are there better blocking schemes? This also concerns the relative performance of different blocking schemes, which should not have changed.

RQ5: Can name-matching substitute author annotation? We would now answer this question with "No", because of the simple reason that the matching relation is not transitive and therefore not directly comparable to an equivalence relation suggested by the system. Applying the transitive closure to the matching relation will add a large amount of false-positives in many cases.

RQ6: What impact has blocking on (the evaluation of) AND? The overall contribution of the blocking step towards the final disambiguation result is diminished if the results are aggregated by number of *distinct* rIDs, i.e. true number of authors (with a certain surname). For smaller problems, blocking still contributes a large part of the final performance. This was also confirmed in later research (see Chapter 6). As stressed in the paper, performing a macro average over results for different blocks is not acceptable as it mainly reflect the distribution of problem sizes rather than the performance of the model. Usually there are many trivial tasks that are perfectly disambiguated and their ratio in proportion to the overall set of tasks then dominates the result. This was discussed in some more detail describing problem size as a *confounder* in the paper in Chapter 6. Optimally however, no average should be taken, but the relevant counts of P, T and TP should be aggregated over all tasks to avoid what is known as *Simpson's paradox* [95].

Chapter 6

Progressive Blocking: Lattice-based Progressive Author Disambiguation

This work was meant to combine and refine the insights and methods from the clustering paper (Chapter 4) and the blocking paper (Chapter 5) to directly view the contribution of each task to the overall author disambiguation performance. In particular, from the clustering work, we take the probabilistic similarity measure and agglomerative clustering as a blocking method and from the blocking work, we adopt and expand the idea of arranging name-based representations of author mentions by the subset partial as well as considering these representations' observation frequency into account. In addition, a number of improvements motivated by the literature and the ideas already established in the previous projects were pending that should be implemented in this context. The main such improvement was to embrace the notion of progressive resolution for author disambiguation. Another was to better formalize the notions from the previous work, like in particular the relationship between name-matching and derived blocking-based equivalence relations. Due to the large scale, especially of the author annotation, we have continued to use the Web of Science for our experiments.

In this chapter, we complete the definition of the name matching problem as building the subset partial order over all entity representations under a minimal representation (here surname, first initial). Then we derive and compare means for progressive resolution through sequential block merging in the subset partial order, leading to specific hierarchies of record partitions (HRP) for the author disambiguation problem.

6.1 Overview

In this work, we propose continued block merging based on name matching and observation frequency to realize a progressive framework for author disambiguation. Initially, each representation is considered a separate block. Then, blocks are iteratively merged and clustered if their representations are decided to be sufficiently similar. This similarity is efficiently implemented as edge weights in the subset partial order defined over representations. These weights are assumed to indicate the coreference likelihood of the author mentions in the union of the respective mention sets. A number of different edge-weighting schemes are compared, but the main method, which also outperforms all others is based on the representations' observation frequency, specifically it is similar to the *confidence* value as known from *association rule learning*. Progressiveness also requires a new type of evaluation. We use the dedicated *ec** measure to space out the iterations on the x-axis proportional to the number of comparisons done during it. We measure the number of *emitted comparisons* (*ec*) as $\sum_{B \in \mathcal{B}_i} |B|^2$, where $\mathcal{B}_i$ denotes the set of blocks in iteration i. This is the same as the number of pairs suggested by the blocking scheme, i.e. P. In *ec** as opposed to *ec*, this is normalized as $\frac{P}{T}$. The objective is to maximize precision $\frac{TP}{P}$ and recall $\frac{TP}{T}$ and minimize $\frac{P}{T}$. When optimizing recall against ec*, this trade-off is equivalent to optimizing precision for the suggested pairs of the current blocks, thereby forcing an efficient suggestion with few false-positives. Assuming that the set of suggested comparisons of a later iteration includes those suggested earlier, this requirement formulates the goal of suggesting as many correct pairs as possible as early as possible. To compare the effect of progressive blocking through iterative block merging and within-block clustering, one can plot the performance of both against the ec* values of the respective progressive blocking scheme (i.e. performance of blocking alone as well as blocking in conjunction with clustering – but both plotted against the number of pairs suggested by blocking). In this work, edge weight modification during the unfolding progression plays a central role, as performance varies greatly depending on how the initial edge weights are modified. For example, we consider a redundancy measure to discourage merging of small and large blocks as this can offer little gain in true-positives. Hence, a number of possible baselines and configurations are compared. The results are promising as our main method outperforms all other variations and baselines. In the context of the evaluation, we have elaborated further on the potential confounder task-scale that was already eluded to in the previous works. We have also improved the implementation of the probabilistic measure introduced in Chapter 4 by dynamic programming.

6.2 Terminology and Notation

In the following we list and explain the basic terminology used in the paper and the notation that was used in this context at the time.

x mention

X set of all mentions

R_x representation of x, set of attribute-value pairs

A set of all attributes

V set of all values

T_{R_x} type of R_x, set of all attributes in R_x

$\#(R_x)$ observation count of R_x, how many times R_x is exactly observed

$\breve{\#}(R_x)$ carry count of R_x, how many times R_x or a specification is observed

$p(R_x|R_{x'})$ probability of R_x given R'_x, edge weight between them

$\mu_{R_x R_{x'}}$ cost factor for merging the node of R_x with that of R'_x

δ discounting parameter, % of observation passed on to specifications

$\Delta_{R_x \leftarrow R_{x'}}$ discount mass from $R_{x'}$ to R_x

t_b blocking threshold, merge all blocks connected by higher weighted edges

d number of attributes, i.e. $|A|$

m number of mentions, i.e. $|X|$

n number of distinct representations

$p(C|C')$ conditional probability of one cluster C given another one C'

$p(x|x')$ and so on... see Chapter 4

$f(C), f(C,C'), f(x,x'), f(\{x_1,x_2\}|\{x'\}), f(\{x\}|\{x'_1,x'_1\})$ components of $p(C|C')$

t_d disambiguation threshold

G, f_0 parameters of S-curve for t_d

[**sufi**] surname, first-initial blocking baseline

[**asis**] use name as is blocking baseline

[**milo** use all initials unless no other second initial exists

[**inits**] surname all-initials baseline

[**rdm**] completely random merge blocking baseline

[**nbrdm**] random edge-contraction blocking baseline [**base**] constant edge weights blocking baseline

[**pnew**] blocking baseline solely based on $\mu_{R_x R_{x'}}$

[**base+pnew**] blocking scheme combining constant edge weights with $\mu_{R_x R_{x'}}$

[**base+disc**] blocking scheme using discounted edge weights

[**pnew+disc**] blocking scheme combining discounted edge weights and $\mu_{R_x R_{x'}}$

[**1link**] single-link clustering baseline with $p(x|x')$

[**1link+cosim**] single-link clustering baseline with cosine similarity

TP true-positive mention pairs

P positive mention pairs

T true mention pairs

WEST selection of frequent English, German and Iberic names

MIDDLE selection of frequent Indian and Arabic names

EAST selection of frequent Chinese, Japanese and Korean names

6.3 Publication

Title Lattice-based Progressive Author Disambiguation

Authors Tobias Backes, Stefan Dietze

Document Type Journal Article

Venue Information Systems, Elsevier

Copyright © 2022 Copyright held by the owner/author(s). Publication rights licensed to Elsevier.

DOI 10.1016/j.is.2022.102056

MLA Backes, Tobias, and Stefan Dietze. "Lattice-Based Progressive Author Disambiguation." Information Systems, vol. 109, 2022.

APA Backes, T., & Dietze, S. (2022). Lattice-based Progressive Author Disambiguation. Information Systems, 109.

ISO 690 BACKES, Tobias; DIETZE, Stefan. Lattice-based Progressive Author Disambiguation. Information Systems, vol. 109, 2022.

6.4 Discussion

6.4.1 Shortcomings and Potential Improvements

It should be mentioned that Figure 3a was taken directly from the paper in Chapter 5.

In this paper, we refer to edges as implementing the covering relation, but for example in Figure 6, there are additional edges (e.g. from *M. Ho* to *Michael Ho*). Strictly speaking, this notion only holds for the first iteration. These additional redundant edges do play a role in discounting observation mass up the graph, but their effect has not been studied. They could be removed by transitive reduction. There are still unnecessary unobserved intermediate nodes. These are automatically removed in the first iteration as they have outgoing edges weighted 1 and are merged with their specifications without changing any existing blocks, since the blocks (i.e. sets of mentions) of unobserved representations are empty.

As discussed in Chapter 2.2.6, contracting all edges with a weight above the current blocking threshold is not strictly determined by the merging probability with the probabilities encoded in the edge weights, as this would require a random walk. Furthermore, an important simplified variant was not implemented and compared: Contract edges top to bottom in the hierarchies, merging the most specific representation pairs first. Despite being simpler, this might have results similar to the discounting model. Also, within each specification level, pairs could be ordered according to the edge weights between them.

We have not studied how to decide a good point for stopping the progression. Although using a good clustering method, this is not required to prevent a drop in overall F1 (see Figure 11a), it should be desirable to know when performance is stagnating to save computational resources. Obviously, this has to be done based on proxy measures that somehow approximate the performance development against the annotated data.

In the matrix-based view introduced in Chapter 3.7, our model can be formalized as the dot product of the mention-representations matrix with itself and an intermediate representation-representation matrix that stores the current state of the graph (see Chapter 3.7.7). The resulting sparse mention-mention matrix suggests all the pairs to be compared. This supports the impression that our approach could probably be simplified considerably to foster easier adaption. For example: (a) for each super-block, e.g. surname, first initial combination, build the covering relation over the subset partial order of name representations; (b) sparsify the covering relation depending on the state of the progressive blocking scheme; (c) use the above dot product to obtain the pairs of mentions that are in the sparsified covering relation and their representations are sufficiently similar; (d) modify the dimensionality of the representations based on which ones have been merged.

One should note that we aggregate the results iteration for iteration. This means that we do not assume a parallelization where each *surname, first initial* super-block is processed individually, but a turn-based model, where the first iteration is computed for each super-block, then the second, and so on. The assumption of processing the entire dataset in parallel poses the problem that the state of each super-block would either have to be kept in main memory or stored on disk, where the latter can be expected to require considerably more time. This problem was not addressed in this work. Also, the problem of introduced redundancy is not solved. When two blocks are merged, their union is clustered again. All previous clustering processes are wasted (in retrospect). In fact, this problem is ignored as we only plot the cost for the current iteration by ec* and not the aggregated cost from the first to the current iteration (the integral). It was not clear from the ec*-literature if one is expected to do the latter or not. This might be one reason why the variant using the redundancy measure $\mu_{R_x R_{x'}}$ does not perform very well: the redundancy it is trying to avoid is not directly reflected in the plots. In future work, the actual integral over all iterations should at least be included in addition to the other plots. Then, means to avoid recomputation might be rewarded.

Due to the fact that we apply a super-blocking by *surname, first initial*, some pairs of coreferring mentions that cross these boundaries (e.g. where only surname is given for one) might be missed in the targeted gold relation. This makes the task artificially easier. However, the effect of this slight benchmark simplification affects all compared methods and should not change their relative performance.

6.4.2 Evaluation

In the result plots, differences between the performance of some of the compared methods look negligible, which is actually not the case, as is shown in the result Tables 2 and 3. The performance of the different variants can be summarized as follows:

[**rdm**] This is the worst performing variant as it merges any representations randomly and does not consider the covering relation at all.

[**nbrdm**] This variant performs slightly better than complete random merging, but the difference is not pronounced when evaluated on the entire dataset.

[**pnew**] The same statement holds as under the previous point.

[**base**] For the larger problem sizes, this variant performs considerably better than the previous three, which is encouraging regarding our method of arranging blocks in the subset partial order of their representations.

[**base+pnew**] Adding the redundancy measure to the [base] variant does not improve the performance as shown in our plots.

[**base+disc**] This is our preferred variant and it outperforms all others, often quite distinctively. However, as pointed out above, it might be that this is simply due to the fact that it approximates a level-by level merging from top to bottom.

[**pnew+disc**] This is the most elaborate variant in that it includes both the previous one and the redundancy measure from [pnew]. However its addition does very slightly deteriorate the performance as displayed in our plots.

Performance of the different baselines can be summarized as follows:

[**sufi**] As we separate the data by *surname, first-initial*, this static baseline amounts to merging all representations, that is the end point of our result plots. It's poor performance is obvious.

[**asis**] This static baseline leaves all names as they are and uses the initial blocks, meaning that its performance corresponds to the starting point of our plots. It is clearly better than the previous baseline, but more advanced methods allow considerable progressive improvements.

[**inits**] This is a progressive baseline that represents mentions only by surname and all available initials, that is it removes first name information. Furthermore, we modify the edge weights in that they are computed exactly as the *confidence* value from association rule learning, which avoids reflexivity.

[**alli**] The static *surname, all-initals* baseline constitutes the starting point of the above progressive baseline. It is much more expensive than the starting point of the other progressive variants as removing first name information creates larger blocks. In some subsets, this baseline performs quite well, in particular if one ignores how expensive it is.

[**milo**] This static baseline has its name from the method proposed by Milojević [35]. In the performance plots, it describes a state that can be localized as somewhere after the starting point of the progressive [inits] baseline. Since this baseline continuously deteriorates from the starting point on, it does not seem that it is superior to the [alli] baseline, in particular if one considers that it is more complicated.

Retrospectively, we note that there is the danger of a confusion between the progressive [inits] baseline in this paper and the fixed blocking scheme of the same name in previous work, as described in Chapter 5, which in turn corresponds to the [alli] baseline. Clustering F1 is congruent with blocking F1 as long as blocking F1 increases and then continues to grow while blocking F1 starts to deteriorate due to a loss of precision. Depending on the clustering method, clustering F1 will continue to grow until everything is compared, or starts to deteriorate slowly at some point (see Figure 11a). Apparently, a within-block clustering using additional features can help absorbing some loss of blocking precision while preserving continued gains in recall.

Chapter 7

Hierarchy Extraction: Towards Hierarchical Affiliation Resolution

In this chapter, we study the use of the subset partial order of entity representations not only for illustrating name matching or deriving progressive blocking schemes, but for describing hierarchical relationships in the real world. By putting constraints on which representations can be merged, we are able to extract institutional hierarchies from large amounts of affiliation strings. We define a modular framework in which components can be exchanged as long as they perform the required task on the respective input. We give one example implementation for each component to obtain a proof-of-concept. Most of the work on this topic was made in the context of a project that was proposed to the KB-Infrastructure Group and had been granted three-month of funding. The focus was on the application of previous work around the subset partial order of blocking representations on institutions as a new type of entities to disambiguate international affiliations. Hierarchical aspects derive naturally from the subset partial order and had already been reported to be relevant by other groups because some affiliations can belong to multiple top-level institutions. The mechanisms by which the partial order interacts with the concept of institutional hierarchies mostly worked as expected in the proposal. However, a number of additional insights regarding concrete sub-tasks, their requirements and potential solutions were identified during the course of the project and when writing up the results. Ultimately, the new task of hierarchical institution resolution could not be solved completely in the first approach, but the structuring elements of the task as discovered by us can be considered to constitute the main contribution of this work. In addition, a number of sub-tasks could already be solved to a satisfactory degree and a comprehensive error analysis as well as a concrete plan for next steps outlined how future work can address the remaining sub-tasks were our initial baselines had not been able to achieve sufficient results yet.

7.1 Overview

In addition to author names, scientific publications are usually attached affiliation strings. There is a great variation in how affiliations are reported, both syntactically in terms of how an institution is referenced and semantically in terms of which degree of hierarchical specification of the author's affiliation is given (e.g. only the university or maybe the university, department and chair). The semantic variations point to a hidden underlying institutional hierarchy, which reveals two potential opportunities for data mining in the set of affiliations: (a) Inducing a hidden, real-world institutional hierarchy; (b) ordering the observed affiliations in a hierarchical way. These tasks are obviously strongly related, but in the first case we might also need to hypothesize unmentioned intermediate institutional nodes, while these are mostly irrelevant in the second case. For example given an affiliation with some university and another with the same plus some department and chair, it is not necessary to hypothesize the department as an individual entity under the university to know that the second affiliation is hierarchically ordered under the first. The approach proposed in our work is based on a pipeline framework going from creating affiliation representations over separating individual top-level institutions, hypothesizing intermediate nodes and ordering affiliations hierarchically to merging equivalent nodes adjacent in the hierarchy. The latter task essentially deals with the distinction of syntactic and semantic differences, where syntactically different but semantically equivalent affiliation representations are to be merged. In the paper, we first describe the logic behind this framework and then proceed to present first baselines for each of the framework's components. An additional challenge lies in the evaluation of hierarchical affiliation resolution and its sub-tasks. Our work is based on affiliation strings from the Web of Science. For top-level and hierarchical evaluation, we deploy an existing WoS-based top-level gold standard and enrich an existing gold hierarchy with WoS affiliations. Results are mixed in that some tasks work very well, while others, in particular those further downstream in the pipeline (e.g. deciding for adjacent nodes whether they are equivalent) are found to be challenging. Nevertheless, for all tasks a credible path to success could be laid out, although it might often require to dedicate a full project's worth of effort to a single sub-tasks, e.g. to learn a (rule-based) binary equivalence classifier over affiliation representations adjacent in the induced hierarchy.

7.2 Terminology and Notation

In this work, we have mostly refrained from using mathematical notation. Therefore, we list instead a number of terms that are introduced in the paper to describe specific concepts and are not necessarily commonly used in this way.

representation set of attribute-value pairs representing an affiliation, corresponds to a node in the inferred institutional hierarchy

affiliation the mention of an institution of any hierarchical level on a document

institution a real world institution of any hierarchical level

top-level institution a real world institution such that that has no higher-level institution

top-level resolution assigning affiliations to the top-level institution(s) that their referenced institution belongs to

hierarchical resolution assigning affiliations to the specific institution in the hierarchy that they refer to

representation task the task of creating meaning and useful representations from affiliation strings

interpolation task the task of deriving unobserved intermediate institutions in the hierarchy from observed representations

collocation task the task of ordering observed affiliation representations hierarchically

conflation task the task of determining whether representations adjacent in the inferred graph are equivalent and then merging them

separation task the task of identifying top-level institutions in the data and separating all affiliations that belong to different top-level institutions for efficiency reasons

label also *attribute*, one of a predefined set of institutional functions like *university*, is paired with some observed phrase of the affiliation string

term also *value*, the normalization of a term in a labelled phrase of the affiliation string, forms a feature pair with this label

part phrase or substring of the affiliation string determined to describe one particular institutional function, then to be labelled and split into terms to derive attribute-value pairs

minimal element representation that no subset exists of in the set of affiliation representations, assumed to reference a (potential) top-level institution

7.3 Publication

Title Towards Hierarchical Affiliation Resolution: Framework, Baselines, Dataset

Authors Tobias Backes, Daniel Hienert, Stefan Dietze

Document Type Journal Article

Venue International Journal on Digital Libraries, Springer

Copyright © 2022 Copyright held by the owner/author(s). Publication rights licensed to Springer.

DOI 10.1007/s00799-022-00326-1

MLA Backes, Tobias, Hienert Daniel, and Dietze, Stefan. "Towards Hierarchical Affiliation Resolution: Framework, Baselines, Dataset." International Journal on Digital Libraries, 28 May 2022.

APA Backes, T., Hienert, D., & Dietze, S. (2022). Towards Hierarchical Affiliation Resolution: Framework, Baselines, Dataset. International Journal on Digital Libraries.

ISO 690 BACKES, TOBIAS, HIENERT, DANIEL, and DIETZE, STEFAN, 2022, Towards Hierarchical Affiliation Resolution: Framework, Baselines, Dataset. International Journal on Digital Libraries. 2022.

7.4 Discussion

7.4.1 Erratum

There is a formatting problem that happened during the publication process: Under Section 5.2, the first enumeration goes from 1–4, followed by one sentence “These aspects are assessed by objective evaluation:” and another enumeration that goes from 5–8. In the formatting, the intersecting sentence was incorrectly but understandingly assigned to point 4 of the first enumeration.

7.4.2 Shortcomings and Potential Improvements

As there are no other existing works to compare our approach against, the main questions to be discussed in the context of this work are interpreting the methods and their results as to how satisfactory they are as well as viewing them in the light of what can be expected in general, given the discrepancy between the data and the hypothetical target outcome. We have dissected the problem into a number of sub-tasks such that their solutions are necessary and sufficient conditions to reach the target. How well each task could be solved by our first baselines has been discussed in detail in the paper. The problematic tasks concern the discovery of real institutions (top- or lower-level) in the collection of affiliation strings. The tasks that link affiliations to discovered institutions achieve much better results as of now. In addition to the sub-tasks T1-T5 that make up the overall task, we have presented five components that make up the overall solution (in our framework). As for the sub-tasks, we have analyzed how well each of these components works at this point and how it may be improved in future work:

Representation This step is not trivial but probably not impossible either, it just requires a lot of further focused effort that would optimally culminate in a proper affiliation parser outputting a standardized representation with all hierarchical levels properly separated and labelled. Due to the unreliability of the keywords used in the data (e.g. department) this parser can only work when trained over the entire corpus and learning in one way or another which terms in which context refer to which real-world hierarchy level.

Separation The success of this step depends a lot on the quality of the representations. It is rendered much easier if we know the top-level institutions, which is actually not an unrealistic assumption, given that there are a number of large databases of international higher education institutions (e.g. the *World Higher Education*

Database[1]), although these will miss smaller institutions like companies or other non-teaching institutes that also produce scientific output.

Interpolation This step could be improved further, which mostly depends on knowing the real labels in the representation to remove labelled components in the correct order. However this step is also not really required except if one aims at inducing the true hidden institutional hierarchy to the greatest details possible.

Collocation This is the central aspect of our approach. There is nothing to be changed or improved as it is simply partial order production using the subset relation of affiliation representations.

Conflation This is currently the most lacking component. It requires a binary equivalence classifier for representations adjacent in the collocation result, which again depends mostly on the quality of the labelling of institutional functions in the representations.

All in all, it is clear that most gains are to be expected from better affiliation representations, so this would be the most urgent component to improve. Still, our experiments have shown that it is a major uncertainty whether it is possible to guess all the hierarchical relationships that are objectively not even indicated in the affiliation strings – that is if the data is sufficient to induce the majority of relationships. In addition, a peculiarity that has not been addressed yet is that the true institutional hierarchies are actually changing over time, which means that changes needed to be tracked in the historical data that ranges from 1980 to 2020 in order to show the relationships only for a specified point in time. In conclusion, hierarchical institution resolution is an extremely challenging task if the goal is to achieve close to perfect results. Currently, only in some cases indications of hierarchical relationships can be extracted. A certain potential for improvement is graspable with predictable next steps. The question whether the task can be solved to a sufficient degree is open. This raises the question of the usefulness of out work. In addition to the hierarchies that have already been discovered, one advantage of our work lies in the visualizations that contribute to the problem solving process as they allow to both browse the data and inspect the outcomes of applied methods. Another merit of our proposed approach and method is that it can be considered a milestone on the way to the solution. Despite remaining problems, the state of the solution achieved so far is not an impasse, but can be improved further as is – and it has been discussed in great detail how that would have to be done.

[1]`https://www.whed.net/home.php`, retrieved June 25th 2021

7.4.3 Evaluation

Evaluation in this work was challenging. Top-level resolution could be easily evaluated using the silver-standard created at Bielefeld University which allows to put the affiliation string and the assigned top-level institution side-by-side. However this was restricted to German Institutions. We have not evaluated international institutions except that the prototype was run on the entire WoS as a proof of concept and some inspection of the output was done. The top-level silver-standard inherently contains two pieces of information: (a) the correct grouping of affiliation strings (point 5: top-level resolution) and (b) the correct assignment of affiliation strings (point 6: top-level linking). In the first case, we know for each pair of affiliation strings whether they should be assigned to the same top-level institution. In the second case, we know for each institution, which affiliation strings should be assigned to it. The first allows the usual pairwise clustering evaluation over the entire dataset. The second allows to pick a certain top-level institution and use a basic descriptive representation thereof (e.g. *UNI:*{*Heidelberg*}) to see which affiliation string representations are supersets thereof. False positives then are those supersets that are not annotated with the current top-level institution. False negatives are those annotated by the latter, but not returned by the superset query.

Hierarchical resolution suffered from the non-existence of a hierarchical gold standard. Here, we would need affiliation strings correctly assigned to nodes in correct institutional hierarchies. Fortunately, we have identified the GERiT database as a correct institutional hierarchy for German institutions. Unfortunately, this does not assign actual "dirty" affiliation strings to these hierarchically ordered lower-level institutions. Therefore, we have manually assigned real affiliation strings from the Web of Science to the nodes of this correct hierarchy for universities *Bonn*, *Trier* and *Heidelberg*, which has provided us with a small hierarchical gold standard. Similar to top-level annotation, this allows two types of evaluation: (a) the correct grouping of affiliation strings (point 7: lower-level resolution) and (b) the correct hierarchical ordering of affiliation strings (point 8: hierarchical resolution). In the first case, we know for each pair of manually assigned affiliation strings whether they should be assigned to the same lower-level institution. In the second case, we know for each pair of manually assigned affiliation strings in which hierarchical relation they stand. We decide to interpret "hierarchical relation" between two affiliation strings a, b as being above or below ($a < b$), equal ($a = b$) or unrelated. Of course, this does not reflect for example how many levels a should be below b, but if the entire hierarchy is correct and all pairs of affiliation strings are in the correct relationship, then a must also be in the right level under b, or there will be mistakes with other pairs. This means that we have two gold relations $<$ and $=$, as well as their union $\leq$, which contains all the pairs of affiliation strings that should be in the respective relation. Then we can compare all pairs of our system's hierarchical output to those gold-standard pairs just as with the equivalence relation evaluated in a clustering setup. False positives are pairs that our system returns under $<$ or $=$ but that are not ordered like this in the gold standard. False negatives are pairs ordered like $<$ or $=$ in the gold standard, for which our system returns no relation – or the wrong.

In the following, we briefly summarize the results of the above described evaluation:

1. **Point 5: top-level resolution**. This amounts to the separation step by which connected components in the subset partial order are identified. Alternatively, minimal elements can be used to define overlapping blocks of affiliations. There are two major problems that lead to oversize institution blocks with many false-positives: (a) overly general representations like *CLIN:* {*Univ*} – these collect to many supersets under them – and (b) lower-level institutions that actually belong to multiple top-level institutions – these lead to oversize connected components even if correctly represented. While the second problem is not relevant when minimal elements are used instead of connected components, the first still applies. We have evaluated by minimal elements, while in the work described in Chapter 8, we have used connected components. Top-level resolution still suffers from a challenging detection of overly general representations and therefore only gives 23% precision at 63% recall.

2. **Point 6: top-level linking**. For the tested three institutions and the corresponding representations *UNI:*{*Trier*}, *UNI:*{*Bonn*} and *UNI:*{*Heidelberg*}, top-level linking works well. This means that most affiliation strings which the top-level gold standard by Bielefeld University assigns to these institutions are actually represented by our method as supersets of the respective representations above. For *Trier*, 1% of true affiliations are missed, 11% for *Bonn* and 19% for *Heidelberg*. For *Trier*, 5% of supersets are not actually referring to Trier University, 1% for *Bonn* and 3% for *Heidelberg*.

3. **Point 7: lower-level resolution**. Here, we measure the overlap between the equivalence of lower-level affiliations ($=$) both in the hierarchical gold-standard and in the hierarchical output of our system. Precision is always 100%, meaning that no two affiliation strings with the same representation should actually be in a hierarchical relation. In other words, our representation module is not dropping information significant for establishing a hierarchical relationship between the annotated affiliations. For *Trier*, 3% of all pairs that should be identical have a non-identical representation and are therefore missed, likewise 18% for *Bonn* and 11% for *Heidelberg*. As indicated by the $\leq$ relation, for *Trier* only 1% of all pairs

are missed and are not in a hierarchical relationship either, likewise 13% for *Bonn* and 11% for *Heidelberg*.

4. **Point 8: hierarchical resolution**. Here, results are poor. For *Trier*, only 3 hierarchical relations have been discovered correctly. Most of the superset relationships actually refer to equivalent representations. This is similar for *Bonn*. For Heidelberg, 30% of the superset relationships refer to actual hierarchical relationships, which corresponds to 4% of all actually annotated hierarchical relationships found.

Chapter 8

Super-blocking: Partial Order Components for Blocking Billion Entities

In this chapter, we study the separation problem, the task of creating super-blocks inside which progressive resolution is applied. This amounts to finding a relation that combines all equivalent mention pairs but still separates most pairs in the dataset, thereby limiting the super-block size to an extend that allows building the subset partial order and progressively merging blocks. In the author disambiguation example, we have previously considered surname, first initial as fixed super-blocks. In hierarchical institution resolution, we have considered minimal elements as super-blocks that correspond to top-level institutions. In this work, the objective is to compute connected components in the subset partial order of entity representations by relating every representation to its minimal element and using this condensed representation to compute connected components. In the same process, we could build the full subset partial order, although this will require more space. It is assumed that the full partial order is built only for one or some individual super-block(s) at a time. The focus of this work is to identify options for balancing speed and space requirements to enable fast search of connected components with custom RAM requirements in the subset partial order over roughly one billion entity representations. A suitable application scenario is finding duplicates in the CORE repository. On top, we also use this approach to find super-blocks on authors and institutions in the Web of Science. In particular, we use two functions to enable effective resolution: (1) The specification function, which identifies and isolates overly general representations and (2) the generalization function, which creates hypothetical generalizations for each observed representation to define which is the minimal information overlap required for equivalence.

8.1 Overview

In this work, we present as major contributions: (a) a formal abstraction of classification, clustering and blocking as bipartite graph operations; (b) the matrix-based abstraction that was already discussed in Chapter 3.7 with some of the blocking methods instantiated therein as examples; (c) a simple parallel algorithm for minimal element search with configurable space/time tradeoff and (d) a blocking method based on (a) and (c) that considers connected components in the subset partial order of set-based representations as blocks and can also be described by (b). The bipartite graph described under (a) is computed using the algorithm in (c) by relating any representation to all minimal element in the sub-/superset partial order that are subsets thereof. The properties unique to the blocking method (d) are mainly the *representation*, *specification* and *generalization* steps that were already sketched in the hierarchy extraction work described in Chapter 7. We have given a proof-of-concept for all four contributions: (a) the bipartite graph abstraction was proven to be useful by describing in it *single-link clustering*, *DBSCAN*, *k-Means*, the *EM algorithm*, as well as any classification model and any blocking model; (b) the matrix-based abstraction was likewise proven to be useful by describing in it specifically *Standard Blocking*, *Suffix-Arrays*, *Sorted Neighborhood*, *Attribute Cluster Blocking* and *iMatch* blocking and generally *Hash-based*, *Sort-based*, *hybrid*, *learning-based* and *schema-agnostic* blocking (see Papadakis et al. [1] for this categorization); (c) the algorithm was proven to find minimal elements in more than one Billion records within 24h, which could probably be considerably faster with a C++ implementation; (d) the blocking method itself with a number of different configurations was evaluated on large gold-annotated subsets of three large datasets for duplicate detection, author disambiguation and institution resolution, respectively. For the main application scenario of duplicate detection, this was also compared against a SimHash baseline and on a smaller dataset for which results of other baselines have been reported by Gyawali et al. [96]. Results suggest that our method delivers performance similar to an infeasible vector-similarity-based method [96] for duplicate detection. For author disambiguation, it regresses to classic surname, first-initial super-blocking that was also used in the work described in Chapter 6 and for affiliation resolution, it confirms and explains the problems already noted in the separation step of the work described in Chapter 7.

8.2 Terminology and Notation

In this work, we have mostly refrained from using mathematical notation. Therefore, we list instead a number of terms that are introduced in the paper to describe specific concepts and are not necessarily commonly used in this way.

necessary conditions for blocking equivalence here which features need to be present in either representation to consider connecting two representations, corresponds to minimum requirements during the specification step

sufficient conditions for blocking equivalence here one representation being a subset of another is sufficient for blocking equivalence; further, the transitive symmetric closure of this relation creates blocks, i.e. sharing at least one minimal element is sufficient, as well as being in the same connected component in the corresponding bipartite graph

necessary conditions for clustering equivalence here being in the same block is necessary for clustering equivalence

sufficient conditions for clustering equivalence being in the same cluster is sufficient for clustering equivalence, however clustering is not studied in this work

super-blocking first coarse blocking that produces blocks small enough for more advanced blocking algorithms like the on described in Chapter 6 but possibly too larger for expensive clustering methods

bipartite graph graph whose vertices are in either one of two sets such that no edge connects two nodes in the same set

point any data sample, node in the bipartite graph

connector node in one of the two parts of the bipartite graph; indirectly connects points in the other set which have both edges going to this connector

connected component element of the partitioning of a graph's nodes (and edges) such that all nodes in the same component are connected to all other nodes in it by some path and no edges are between nodes in different components

transitive closure of the relation describing the bipartite graph's edges: corresponds to connected components

subset partial order the partial order defined by the subset relation; alternatively by the superset relation, as we here consider its symmetric closure, i.e. an undirected graph where the direction (subset/superset) does not matter

minimal element $\check{R}$ here any representation for which there is no subset in the data

representation step process by which representations are created from the data entries, i.e. sets of features, where the latter are usually attribute-value pairs

specification step process by which all representations that do not meet the minimal requirements are isolated by adding to them either their representation-ID or their mention-ID

generalization step process by which we add unobserved representations for any observed representation according to some generalization rule by dropping certain features from the observed representation; such generalization should correspond to a minimum required overlap with other potentially coreferring representations

X the set of all mentions, sample, data points

$\mathcal{R}$ the set of all representations (or blocking keys) of mentions $x \in X$ over some feature set $\mathcal{F}$

$\mathcal{F}$ set of all features

${}^{x}\overset{k}{\boxplus}$ the $|X| \times |\mathcal{R}|$ mention:key matrix, which assigns each mention one or more blocking keys.

${}^{k}\overset{k}{\boxplus}$ the $|\mathcal{R}| \times |\mathcal{R}|$ key:key matrix, which stores the relations between blocking keys.

${}^{x}\overset{x}{\boxplus}$ the $|X| \times |X|$ mention:mention matrix, which is the target and holds all pairs of mentions to be verified

$\check{R}_{+}$ minimal element that has at least one superset

Я inverted index pointing from features $f \in \mathcal{F}$ to all representations $\text{Я}(f) \in \{R \in \mathcal{R} \mid f \in R\}$ that include them

patch processed iteratively; corresponds to a vertical slice in a matrix with potential subsets along the first dimension and potential supersets along the second; larger patches require more memory and are more time-efficient

batch processed in parallel; corresponds to a horizontal slice within a patch; list of potential subsets defining an independent search task

8.3 Publication

Working title Subset Partial Order Components for Blocking Billion Entities

Authors Tobias Backes, Stefan Dietze

Document Type Rejected Conference Paper

Venue Rejected at 49th International Conference on Very Large Databases (VLDB)

Copyright Currently unpublished. Copyright by the authors.

Subset Partial Order Components for Blocking Billion Entities

Tobias Backes
GESIS - Leibniz Institute for the Social Sciences
Cologne, Germany
tobias.backes@gesis.org

Stefan Dietze
GESIS - Leibniz Institute for the Social Sciences
Cologne, Germany
stefan.dietze@gesis.org

ABSTRACT

In entity resolution, *blocking* separates data into chunks that can be further processed by more expensive methods. Two entity mentions are in the same block if they share identical or related *blocking-keys*. Previous work has sometimes related blocking keys by grouping or alphabetically sorting them. As was shown for author disambiguation, the respective equivalences or total orders are not necessarily well-suited to model the logical relationship between blocking keys. Our novel blocking approach exploits the subset *partial* order over entity representations to build a bipartite graph with connected components as blocks. To prevent over- and underconnectedness, we isolate overly general representations by specification and hypothesize feature overlaps sufficient for blocking equivalence by generalization. Complementing less convenient alternatives, we present a new parallellized algorithm with configurable time/space tradeoff for minimal set search in the subset partial order to build the bipartite graph. Experiments on large gold standards for publication records, author mentions and affiliation strings suggest that our approach is competitive in performance and allows better addressing of domain-specific problems by adjusting entity representations and their minimum required overlaps with potentially coreferent mentions.

PVLDB Reference Format:
Tobias Backes and Stefan Dietze. Subset Partial Order Components for Blocking Billion Entities. PVLDB, 16(4): XXX-XXX, 2023.
doi:XX.XX/XXX.XX

PVLDB Availability Tag:
The source code of this research paper has been made publicly available at http://vldb.org/pvldb/format_vol14.html.

1 INTRODUCTION

Grouping entity mentions according to some definition of equivalence is a basic but important task in information processing. Depending on the apriori availability of labels, this is commonly addressed by classification or clustering, i.e. in the form of *Entity Linking* or *Entity Resolution* (*ER*). In classification, it usually suffices to compare each data point to all classes and assign it to the most similar one. In clustering, groupings are obtained by comparing points to each other. Large-scale scenarios usually require a special pre-grouping that is often referred to as *blocking* and essentially constitutes a particularly cheap clustering method. Many different blocking methods have been proposed in the literature. Although best practices vary greatly between different tasks such as duplicate detection and author disambiguation, all approaches aim at identifying entity pairs that share identical or similar *blocking keys*. Task-dependent concerns are which and how many keys are created for an entity mention and the definition of blocking-key relationships. Regarding the latter, current methods either use key equivalence relations (including identity, e.g. *Attribute Cluster Blocking*) or total orders (i.e. alphabetically sorting keys, e.g. *Sorted Neighborhood*) [21]. We argue that neither equivalence relations nor total orders properly model relations between set-based blocking keys and instead propose to arrange them in *partial orders*. In Backes [3] and Backes and Dietze [4], it was shown how the subset partial order expresses matching relationships between author names. Total orders fail to consider logical inclusion, for example, ***Doe, Jac****k Herbert* is alphabetically closer to ***Doe, Jac****queline K.* than to ***Doe, J. Herbert***. It was also shown to be impossible to view name-matching as an equivalence relation, since the former is not transitive.

Transferring these insights from AD blocking to ER in general, we define an adaptable framework for blocking entities in different domains that scales beyond one Billion entity mention representations by using a novel algorithm for minimal set search to identify connected components in the subset partial order. In contrast to existing parallellized solutions [15, 20], our algorithm follows a convenient job-based approach and allows for configurable space-time tradeoff without inter-process synchronization. Our approach can work with author names as well as title word n-grams, publication dates or parsed elements, while allowing the user to define domain-dependent ad-hoc baselines. Our main contributions are:

(C1) We introduce a theoretically justified blocking method based on connected components in the subset partial order of entity representations, which allows us to incorporate the state-of-the-art in author disambiguation, reproduce and model the difficulties in affiliation resolution and par performance of the much more expensive vector-similarity method in (near-)duplicate detection. Our method also outperforms the fast *SimHash* method because its greater tolerance for variation allows it to obtain higher recall.

(C2) We develop a new algorithm for minimal set search that allows parallelization with customizable time-memory trade-off and scales our method to more than one Billion representations.

Next, we review state-of-the-art blocking approaches, successful partial-order applications to ER and algorithms for minimal element search in partial orders. In section 3, we conceptualize the previously described blocking methods in a partial-order-based model of key-overlaps and on this basis contrast them with the description of our suggested approach. In section 4, we describe our algorithm for minimal element search. After assessing and comparing its output for different configurations in section 5, we conclude in section 7.

Proceedings of the VLDB Endowment, Vol. 16, No. 4 ISSN 2150-8097.
doi:XX.XX/XXX.XX

2 RELATED WORK

2.1 State-of-the-art in Blocking Methods

A recent comprehensive survey by Papadakis et al. [21] has summarized the state-of-the-art in blocking methods. We find that these methods can be described by two fundamental properties: (1) blocking-key creation and (2) blocking-key relations. Further, a matrix view can be used to better conceptualize their interaction, using three sparse matrices (see Figure 1):

(1) ${}^{x}\overset{k}{\boxplus}$, the $|X| \times |\mathcal{R}|$ mention:key matrix, which assigns each mention one or more blocking keys.

(2) ${}^{k}\overset{k}{\boxplus}$, the $|\mathcal{R}| \times |\mathcal{R}|$ key:key matrix, which stores the relations between blocking keys.

(3) ${}^{x}\overset{x}{\boxplus}$, the $|X| \times |X|$ mention:mention matrix, which is the target and holds all pairs of mentions to be verified

The target matrix is computed as the sparse dot-product

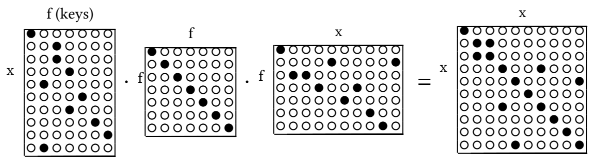
$${}^{x}\overset{x}{\boxplus} = {}^{x}\overset{k}{\boxplus} \cdot {}^{k}\overset{k}{\boxplus} \cdot {}^{k}\overset{x}{\boxplus}$$

where ${}^{k}\overset{x}{\boxplus}$ is simply the transpose of ${}^{x}\overset{k}{\boxplus}$. For (1) blocking-key creation, each key constitutes a conjunctions of features that are combined in a disjunction if one mention has multiple keys in ${}^{x}\overset{k}{\boxplus}$ (i.e. two mentions share at least one key). Regarding (2) key-key relations, these are encoded in ${}^{k}\overset{k}{\boxplus}$ as either identity, equivalence or total orders. The survey [21] distinguishes five classes of *block building* methods, which we describe in the following.

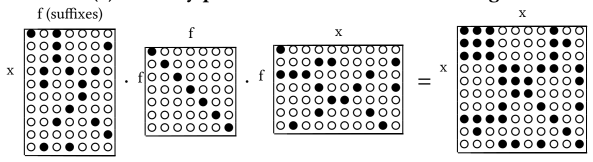
(a) **Hash-based blocking** focuses on blocking schemes that extract features (i.e. conjunctions of essential properties or sufficient overlaps) to create one or more keys per mention. Key-key relations are not used, i.e. the identity matrix is used for ${}^{k}\overset{k}{\boxplus}$. Examples are *Standard Blocking* (Figure 1a) and *Suffix-Arrays* (Figure 1b) as well as *locality sensitive hashing (LSH)* methods like *SimHash* introduced by Charikar [8], evaluated by Henzinger [12] and applied among others in Manku et al. [18] and Ravichandran and Vassilvitski [26]. SimHash is an efficient way of finding pairs of hash keys with large overlaps without having to explicitly create all sufficient overlaps as blocking keys. There are different variants of this method, depending on feature extraction, the hash-function and how feature-hashes are combined into a record-hash.

(b) **Sort-based blocking** uses a total (alphanumerical) order on the blocking keys to relate keys by fixed-size or adaptive windows over this sequence. Here, the key-key relation encoded in ${}^{k}\overset{k}{\boxplus}$ is not transitive and consequently neither is the target relation in ${}^{x}\overset{x}{\boxplus}$, which thus encodes not a block partitioning, but a set of pairwise verification tasks. If the above windows are overlapping as in *Sorted Neighborhood Blocking* (Figure 1c), the transitive closure cannot be applied before verification as it would group all keys and mentions together. One could add delimiters to the alphanumerical order to create multiple sequences with individual minimal elements.

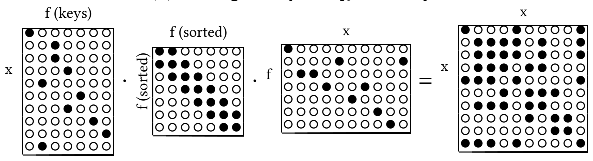
(c) **Hybrid blocking** combines the use of multiple keys with their sorting. The mention-key matrix ${}^{x}\overset{k}{\boxplus}$ can then map more than one key to each mention (as in Figure 1b), which are in turn mapped to each other in ${}^{k}\overset{k}{\boxplus}$ based on their alphanumerical closeness (as in Figure 1c). As both the use of multiple keys (whereof only one has to match) as well as their inter-relation eases the requirements for mentions to be coreferent, it is natural that here keys should be created according to a relatively specific scheme.



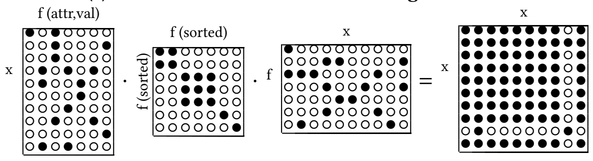
(a) One key per mention: *Standard Blocking*



(b) Multiple keys: *Suffix-Arrays*



(c) Total order windows: *Sorted Neighborhood*



(d) Key grouping: *Attribute Cluster Blocking*

Figure 1: Five different types of blocking methods in the matrix-based framework — with one example per type.

(d) **Learning-based blocking** uses machine-learning to obtain ${}^{x}\overset{k}{\boxplus}$ and possibly ${}^{k}\overset{k}{\boxplus}$. Assuming ${}^{k}\overset{k}{\boxplus}$ is the identity matrix, the blocking method is easily described as a set of DNF formulas with the conjunctions inside the key definitions and their disjunction by ${}^{x}\overset{k}{\boxplus}$. Such can be learned from a mention to feature mapping by dedicated *DNF-learners* that minimize the difference between the output matrix ${}^{x}\overset{x}{\boxplus}$ and a corresponding gold equivalence relation.

(e) **Schema-agnostic blocking** describes approaches that generalize or drop the attribute part of the AND-combined attribute-value pairs to create blocking keys, e.g. using (name-part, Doe) or Doe instead of (surname, Doe). This allows to process unstructured data without reliable field information but diminishes options for modelling logical matching relations.[1]

[1] Any blocking methods realized by comparing relational database table rows based on the similarity or identity of values in the same column will struggle to process data in which multiple features exist for the same (non-)attribute as this prevents a simple one-to-one mapping between attributes and columns. However, this is mostly an implementations issue, as the subset or overlap relations between keys can be easily implemented as set- or bag-relations where blocking keys are considered sets or bags of the features present in the respective conjunctions.

2.2 Entity Resolution with Partial Orders

Backes [3] has described how the subset partial order over parsed name constituents models matching relations between author names using a lattice of author name types. For example *John Doe* is a direct specification of *J. Doe*. Observed author mentions are ordered according to this lattice, producing semilattices under surname,first-initial that encode name-matching and its consequences for blocking. This was further formalized and used for progressive blocking in Backes and Dietze [4]. Backes et al. [5] suggest to use the approach to uncover hierarchical relationships among affiliation strings. To detect unobserved intermediate hierarchy levels, observed affiliations were generalized by dropping the most specific elements from their representations. To isolate minimal elements not corresponding to top-level institutions, overly-general representations were specified by adding unique identifiers.

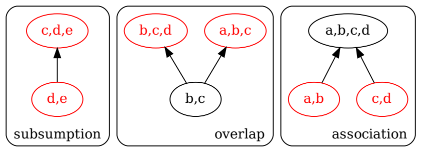
2.3 State-of-the-art in Partial Order Algorithms

Partial order production and extremal sets. A series of works by Pritchard as well as Yellin (with Jutla) [22–25, 32, 33] regarding both the computation of the subset/superset partial order and closely related extremal set search established an $O(N/\log N)$ runtime bound together with two heuristics: (a) cardinality ordering of itemsets and (b) frequency ordering of items with lexicographic ordering of itemsets.While cardinality-ordering exploits the fact that only a larger set can be a proper subset, lexicographical ordering represents itemsets as a sequence of their (frequency-)sorted items before ordering these sequences lexicographically. Then, a subset is either a prefix of the superset, or is guaranteed to follow the superset in the ordering. Both heuristics can be combined by ordering first by cardinality and then lexicographically within each cardinality block. Shen et al. have described two first theoretical parallel algorithms for finding extremal sets [30] and have further shown how to update and maintain extremal set information in a dynamic environment where itemsets are added, removed or modified [29]. In 2011, Bayardo and Panda [6] revisited the problem of extremal set search by simplifying Pritchard's algorithm and defining two separate versions for each of the two heuristics. The question of parallelization remained open except for the recursive algorithm by Leiserson et al. [15] for the general case of partial order. In 2014, Fort et al. [10] proposed GPU-based parallelization. In 2014 and 2016, Marinov et al. [19, 20] added some implementational improvements over Bayardo and Panda [6], which includes caching and also parallelization. However, they require inter-thread communication via shared integer variables, which is a bottleneck for massive parallelization and indeed reported numbers suggest that gains from using an additional worker quickly decline.

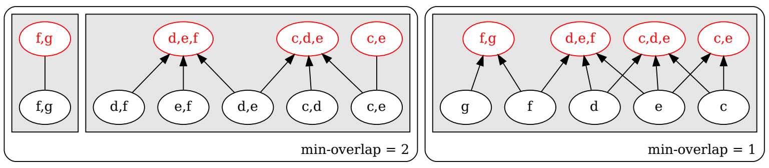
Set containment joins. One can interpret *near-duplicate detection* as *set similarity joins*, which is often centered around *set containment joins*, meaning successive *subset or superset queries*, the latter of which is generally faster [31]. It is common practice to implement superset queries by *intersecting multiple sets* of potential supersets containing the elements of the potential subset as returned from an inverted index. Interestingly, although extremal set search is based on repeatedly identifying subsets or supersets, it is not linked to set containment joins in the literature. There are many publications claiming improvements in time and memory efficiency for set containment joins, most of which have yet to stand the test of time. Still, one should know their underlying principles that are based on problem fundamentals as their usefulness might be context-dependent (e.g. by data properties [16]). A recent paper by Yang et al. [31] distinguishes union-oriented (signature-based blocking with subset enumeration) and intersection-oriented (inverted-index-based superset search) methods. A popular method for the latter is PRETTI by Jampani and Pudi [14], which introduces the concept of a prefix-tree to avoid redundant inverted index lookup and intersection for shared prefixes of potential subsets during superset search. This underlines how set containment join can offer efficiency gains by doing more than just repeated isolated superset queries. A more recent method by Luo et al. [17] is an example for a signature-based, union-oriented set containment approach. Bouros et al. [7] improve PRETTI by (a) limiting prefix-tree construction for potential subsets and (b) progressively building the inverted index for the potential supersets by sorting both by rarest element. Luo et al. [16] present a compromise of signature-hashing without subset enumeration meant to reduce the space cost of intersection-oriented methods while maintaining their speed-wise superiority. Yang et al. [31] refine the intersection-oriented approach by tackling the intersection of the postings lists, iterating over the posting lists' elements in parallel, instead of progressively intersecting list by list. There are many more works on set containment. Savnik et al. [28] give a recent overview of relevant research domains and how the problem of set containment is addressed in them. Their own contribution does not address set containment *join* in the sense that they only facilitate individual, not repeated sub-/superset queries.

2.4 Relating our Approach

Our proposed approach is sort-based, but instead of a total order, we use a partial order that could be encoded in $k^{\overset{k}{\boxplus}}$ and may have non-overlapping constituents that are discovered through connected component labelling. It can be schema-aware or agnostic, depending on whether the features in the representations are attribute-value pairs or just values, respectively. These representations correspond to blocking keys, whereof we use only one per mention. We build heavily on prior works described in Section 2.2. In particular the work on affiliation resolution with its modular components is used as a blueprint for our presented solution, where we focus on improving means for the *separation* step. In our experiments, we also impose this super-blocking on the AD task. In addition, we study the duplicate detection task, which has not yet been addressed with partial orders. Our entity representations correspond to itemsets, which our proposed algorithm orders by cardinality when creating jobs for parallel superset search. Instead of a frequency-based lexicographical heuristic, we use a shared inverted index for this task. In contrast to other parallellized approaches discussed in Section 2.3, our solution is not just theoretical (like [30]) requires neither a GPU (like [10]), nor inter-process communication (like [20]). We have implemented the recursive parallellization by Leiserson et al. [15] but found it inconvenient. The inverted index can be improved by exploiting techniques discussed for set containment joins. The work by Shen [29] suggests that efficient maintenance of extremal sets could turn our approach into a *dynamic* blocking method.



(a) Different ways of how keys (in red) may be related in the subset partial order.



(b) Additional keys resulting from applying on mentions with features $\{f, g\}$, $\{d, e, f\}$, $\{c, d, e\}$ and $\{c, e\}$ a minimum overlap of 2 or 1.

Figure 2: Relating blocking keys in the subset partial order of mention representations and minimal overlaps.

3 PRELIMINARIES AND METHOD OVERVIEW

Entity mentions are associated with a number of features like author names, title terms or publication dates, some of which might be particularly useful for blocking. In one way or another, all blocking methods approximate measures of feature-overlap, i.e. they assume that mention pairs likely to be coreferent are recognized by having in common a more significant selection of features than those that are not. Such feature overlap is generally modelled in blocking methods as two separate aspects: (1) blocking-key creation and (2) blocking-key relation. Two mentions are considered likely coreferent if they share identical or related blocking-keys.

3.1 Blocking-keys and key-relations

A blocking key represents a minimal required commonality between two mentions, in other words, a sufficient condition for blocking equivalence and a necessary condition for actual (clustering/verification) equivalence. It is a conjunction of features, e.g.

$$(\text{sur}, \text{Doe}) \wedge (\text{given}_1, \text{John})$$

If for the purpose of blocking, a mention is represented using multiple keys, these form a disjunction, e.g.

$$((\text{sur}, \text{Doe}) \wedge (\text{given}_1, \text{John})) \vee ((\text{sur}, \text{Doe}) \wedge (\text{init}_1, \text{J})) \wedge (\text{init}_2, \text{H}))$$

A set of blocking keys that represents an entity mention is a logical formula in *disjunctive normal form* (*DNF*) that specifies which features need to be present in another mention's representation in order for it to be taken into the same block. The sufficient conditions for blocking equivalence introduced by blocking keys can be further relaxed by relating keys. Some blocking methods like *Attribute Cluster Blocking* [21] use equivalence relations to relate similar keys in groups. However, in practice it is often difficult to define boundaries such that all keys within a group can be considered equivalent and any keys from different groups are not. For example, it was noted in Backes [3] that the matching relation between person names used for blocking in author disambiguation is not an equivalence relation. Other approaches like *Sorted Neighborhood* [21] relate blocking keys in a total order by alphabetically sorting them. However, this assumes that a key is only a string while using the more versatile above definition, a blocking key is actually a set of features. Sets are not sorted in a total, but in a *partial order* by the subset relation. Conveniently, this relation also reflects matching relationships [3]: two representations match if they have a common superset in the set of all legal representations. For example *John Doe* and *J. Doe* match because the former is their common superset. *John H. Doe* and *J. Herbert Doe* match as well, because there is a common legal superset *John Herbert Doe* to both.

3.2 Key-connectedness and -equivalence

As their number is potentially infinite, it is not possible to enumerate all legal common supersets of all representation pairs. However, prior work in the author disambiguation domain [3, 4] suggests that matching entity representations are usually also related by sharing a common *subset* (cf. Figure 2a). Due to the incompatibility of the matching and the blocking relation, supersets sharing a common subset need to be placed in the same block as coreferent mentions in either specification could also be referenced by their common generalization. For example, although *John Doe* and *Jack Doe* may be considered contradictory, mentions represented by both may be coreferent with those represented by *J. Doe* and therefore cannot be separated without the risk of separating coreferent mentions [3]. In this example, each author mention is represented by one key (a conjunction of labelled name constituents) and keys are related in the *subset partial order* which constitutes a skeleton for the equivalence relation induced by its *symmetric closure*, or the connected components of the corresponding *directed acyclic graph*. Even more, the *covering relation* (see Backes and Dietze [4]) corresponding to the transitive reduction of the subset partial order suffices to describe this key-relation. Further, a bipartite graph in which each representation is only related to those of its subsets that are also *minimal sets*, is sufficient to maintain the same connectivity. A minimal set in the subset partial order is any set without a superset. Finally, it should be noted that it is cheaper to label each key by the graph component it is in than to compute the (symmetric) transitive closure of the underlying relation (i.e. the set of all pairs in a connected component). While the worst case complexity of the latter cannot be lower than the quadratic size of the output and is more exactly bound by the best known algorithm for matrix multiplication [1], the former is easily obtained in linear time [13].

3.3 Blocking-keys as minimum overlaps and mention-connectedness by key-sharing

In the duplicate detection domain, it can be more convenient to think about sufficient- or *minimum overlaps* than about matching relationships. We say it is sufficient for two mentions x_1, x_2 to be in the same block if they share an identical or related key. This means that each key corresponds to a sufficient overlap of features $f \in F$. If we have for example a mention x associated with 8 features $f_1, \ldots, f_8$ and we define a sufficient overlap as any subset thereof with at least 6 out of 8 features, then there are $\binom{8}{6}$ sufficient overlaps

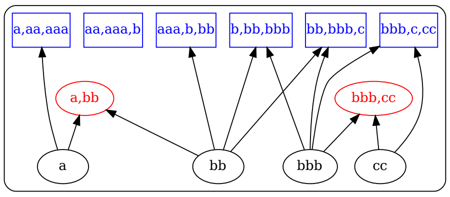


Figure 3: Mentions with features $\{a, bb\}$ and $\{bbb, cc\}$ are connected via their keys bb and bbb, respectively, which are in turn related by key windows $\{b, bb, bbb\}$ and $\{bb, bbb, c\}$.

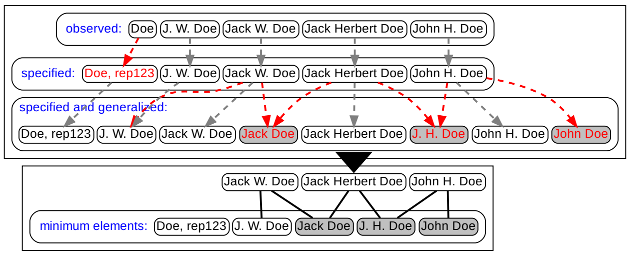


Figure 4: Example data being preprocessed and linked.

for this mention, for example $\{f_1, f_3, \dots f_8\}$ and $\{f_1, \dots f_6, f_8\}$. In the popular SimHash method, all mentions are assigned one fixed-length hash, e.g. a length-64 bitstring, where each bit corresponds to one feature f that is either present or not. Here, there are only $|F| = 128$ different features, i.e. each of the 64 positions/attributes with a value of either 0 or 1. If we define a maximum difference of 7 bits, i.e. a minimum overlap of 57 bits for two mentions to be coreferent, this corresponds to $\binom{64}{57} \approx 621M$ sufficient overlaps, i.e. blocking keys. The advantage of the algorithmic part of the SimHash concept is that it needs not create these hypothetical overlaps in order to find the keys that are sufficiently similar.

If we understand the blocking keys of a mention as subsets of its features, we can also use all of a mention's features as a key and consider it the mention's *representation*. Then, the subset partial order will relate representations of any two mentions that share a blocking key, i.e. some sufficient subset of features. In a simple blocking-scheme, we can create all subsets with a certain size as keys (see Figure 2b), but we may also deploy more advanced schemes that can decide for each individual representation which features to remove in order to obtain minimum overlaps. In the same way, known equivalences between keys can be added as the union of their feature sets. Figure 3 shows how a concept of sorted neighborhood can be realized in this context. This also corresponds to the fact that we can merge representations of equivalent mentions (c.f. Backes and Dietze [4]) and to the above mentioned matching relationship expressed by common legal supersets.

3.4 Method Overview and Rationale

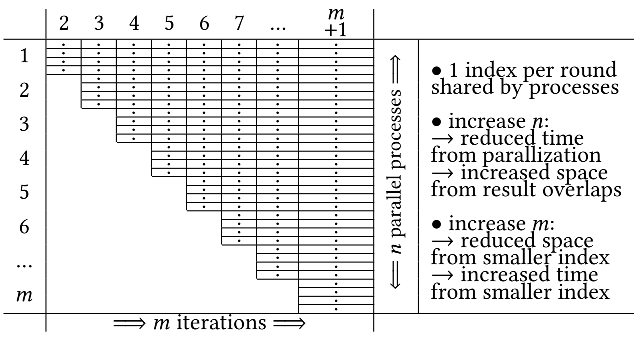
In the subset partial order, two entity representations / blocking keys are related by sharing a common sub- or superset (including the case that one is a subset of the other). Subsets either correspond to sufficient overlaps, or are overly general representations. Sufficient overlaps can be generated by feature removal from observed representations. Overly general representations can be isolated by adding unique features to them. Therefore, our proposed method uses the following preprocessing steps: *(A) Specification*: Identify representations that are too general to be considered their own blocking keys because that would connect too many (unrelated) representations as supersets thereof. Specify these by adding a unique identifier to isolate them; *(B) Generalization*: Define a mentions minimum overlaps as subsets of its representation. When we have created the required representations, we have to produce their partial ordering and then consider any mentions to be in the same block if their representations are connected in the corresponding graph. For efficiency reasons, we do not build the partial order itself, but instead compute a smaller *bipartite graph* with the same connectivity by relating all keys to their minimal elements and ignoring all keys with neither sub- nor supersets. Then we use simple connected component search to label all representations with a component label and add the unrelated keys as singleton blocks. As a result of generalization, the size of the collection can grow largely during preprocessing, depending on the number of added generalizations. For example, if we decide to ignore one of a representation's 10 features (e.g. to account for one potential misspelling), there are $\binom{10}{1} = 10$ generalizations – and $\binom{10}{2} = 45$ for two dropped features. If this applies to many data points and the resulting generalizations are not already part of the collection or previously added points, then the data can easily grow many-fold. Although this appears as a disadvantage, the following counter-points should be considered: (i) smarter generalization schemes should be able to reduce the number of generalizations; (ii) the collection growing faster than the number of distinct features increases the chance that a hypothesized generalization is already there (whether from the original data or previous generalizations); (iii) the growth in the collection during preprocessing corresponds to a growth in the complexity of the blocking scheme and the presupposition of explicit overlaps is the only way to avoid pairwise comparisons; (iv) if our blocking scheme leads to unreasonable assumptions about the number of possible overlaps, then we observe this already during preprocessing and can react; (v) since our algorithm iteratively loads batches of configurable size from disk, the larger size of the collection does not affect memory requirements.

4 PARALLEL MINIMAL ELEMENT SEARCH WITH SPACE/TIME TRADEOFF

In this section, we describe our algorithm for minimal element search in the subset partial order. There are a few parallel algorithms related to partial order production, but we were looking for a pragmatic implementation, which none of them provided. Shen and Evans [30] is only theoretical, Leiserson et al. [15] uses recursive parallellization, Fort et al. [10] is developed for GPU and Marinov et al. [20] requires inter-process communication. Instead, we developed a parallelization framework based on batches that are retrieved from a job queue and independently processed by workers. In order to explain our algorithm, first a number of basic principles shall be explained: Minimal elements $\check{R}$ in the subset partial order (i.e. minimal sets) are all sets that are not proper subsets: $\check{R} \in \{R \in \mathcal{R} | \neg\exists R' : R' \supset R\}$. This means that they are either proper subsets or neither sub- nor supersets. In fact, we are only interested in those minimal sets $\check{R}_+$ that have at least one superset. These we can obtain by searching all supersets and removing from them all subsets: $\check{R}_+ = \{R \in \mathcal{R} | \exists R' : R \supset R'\} \setminus \{R \in \mathcal{R} | \exists R' : R \subset R'\}$. It means that – in contrast to searching just all supersets – we cannot stop when we have found one subset of a set, but we need to find all of them. In general, we assume that it is easier to find all supersets for a given set than the other way round, because one can deploy an inverted index $\mathcal{A}$ pointing from features $f \in \mathcal{F}$ to all representations $\mathcal{A}(f) \in \{R \in \mathcal{R} \mid f \in R\}$ that include them and then simply compute the intersection of the representation-IDs returned for all features in a potential subset to obtain the supersets: $\{R' \in \mathcal{R} | R' \supset R\} = \bigcap_{f \in R} \mathcal{A}(f)$. This is equivalent to Boolean retrieval with only conjunctions as queries. Finding subsets should be considerably harder because we do not know which features to omit to obtain the subsets. Known shortcuts have been used to avoid unnecessary computations: (a) features f in a representation R are sorted by their frequency $\#(f)$ in the collection, because when we start the intersection with the rarest features, we narrow down the set of candidate representations faster (for example a representation with the rarest feature occurring only once in the collection allows us to break immediately, as it cannot have any supersets other than itself); (b) representations R are sorted by cardinality $|R|$, because a proper superset is always larger than its proper subsets; (c) the representations $\mathcal{A}$(f) containing a certain feature f are sorted by an integer index, so that during the intersection, we know when we have reached a larger integer than the one we are looking for, we can stop.

Thus, in our approach, the central part of searching minimum elements is finding all supersets of a given set. To ensure sub-quadratic complexity, the set of all supersets for a given set should be retrieved in less than linear time, as we have to repeat this for almost every element in the collection. For this, there are a number of methods from Boolean retrieval and related areas. We assume that $\{R' \in \mathcal{R} | R' \supset R\} = \bigcap_{f \in R} \mathcal{A}(f)$ is computed efficiently and focus on how to exploit further the properties of the task to allow convenient parallelization. Here further improvements can be realized using the techniques discussed for set containment joins in Section 2.3. In our parallelization framework, we introduce the concept of *patches and batches*. Patches are processed iteratively. For each patch, the data is loaded from disk and intermediate results are aggregated. Patch size determines the memory footprint and inverted index efficiency. Larger patches require more memory and are more time-efficient. Each patch is further divided into batches, which are simultaneously and independently processed by any number of workers. Table 1 shows how it works: Imagine a matrix with potential subsets along the first dimension and potential supersets along the second. Representations along both dimensions are sorted by set cardinality. All sets of the smallest observed cardinality cannot be supersets and all sets of the largest cardinality cannot be subsets. Patches correspond to vertical slices, that include the borders between different cardinalities (between these borders, more patches can be cut). Batches correspond to horizontal slices within each patch. Batches are stored in a queue that is accessed by parallel workers. In the example, in the first patch, we are looking for size-2 supersets, which amounts to building an inverted index for these representations and sending subset queries for all size-1 potential subsets. Batches are lists of potential subsets, which are independent search tasks since no equal size sets can be proper subsets of each other. As the 'tiles' of search space defined by patches and batches are not overlapping, all tasks can be viewed independently. Patches and batches need not be more than ranges of representation index integers and the features may be loaded from disk as late as possible to save memory. A major assumption of this model is that the parallel workers all *read*-access the same inverted index without it being copied in the RAM. If the next index is built while using the current one, then two indices will have to be stored in memory at the same time. We do not assume that the inverted index can be *written* in parallel, so index creation can be a bottleneck if next index creation takes longer than current index usage. In practice, this is usually not the case. Our algorithm

Table 1: Using n parallel processes and m sequential rounds.


2
3
4
5
6
7
...
m +1
1
2
3
4
5
6
...
m
⟹ m iterations ⟹
⟸ n parallel processes ⟹
• 1 index per round shared by processes
• increase n: → reduced time from parallization → increased space from result overlaps
• increase m: → reduced space from smaller index → increased time from smaller index

```
Algorithm 1: Parallel minel– and superset search
  Data: potential subsets $Y$ and potential supersets $X$
  Result: mapping $f$ : $minels \times supersets$ of minimum
          elements $minels \subset Y$ to their $supersets \subset X$
$f, supersets := \{\} \to \{\}, \{\}$;
$index :=$ make_index($sizes_1$,$X$);
forall $k \in 1 \ldots |sizes|$ do
    $batches :=$ make_batches($sizes_{k-1}$,$Y$);
    forall $worker \in workers$ do
        $worker$.start($batches$, $index$);
    end
    $index' := \begin{cases} \text{make\_index}(sizes_{k+1},X) & \text{if } k+1 < |sizes| \\ \text{NULL} & \text{otherwise} \end{cases}$
    $subsets' \times supersets' := \bigcup_{worker \in workers} worker$.join();
    $supersets := supersets \cup supersets'$;
    $minels' := subsets' - supersets$;
    forall $sub, supers \in subsets' \times supersets'$ do
        if $sub \in minels'$ then
            $f[sub] = f[sub] \cup supers$;
        end
    end
    $index := index'$;
end
```

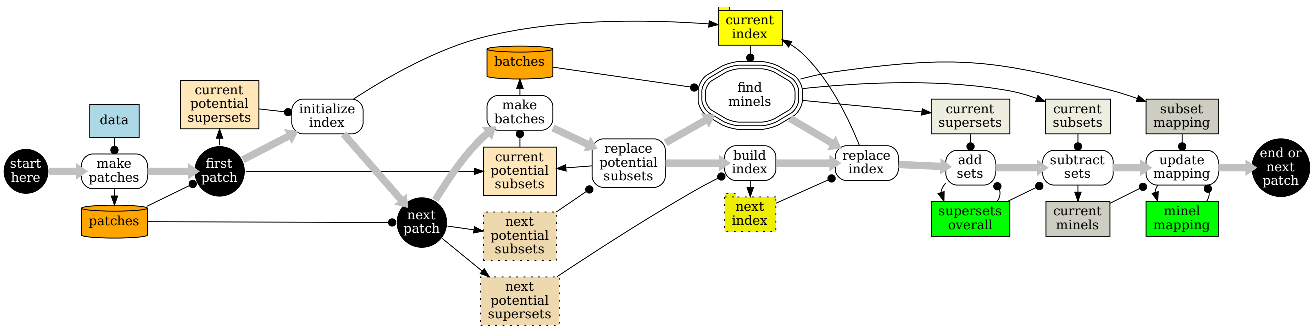


Figure 5: Our algorithm for minimal element search in the subset partial order: (1) make patches from data; (2) draw first patch to (3) initialize index from potential supersets and set potential subsets; (4) draw next patch; (5) make batches from potential subsets; (6) replace current potential subsets by next ones; (7) build the next index and at the same time run a number of parallel workers that process batches using the index; (8) replace current by next index; (9) add current supersets to overall supersets; (10) get current minels by subtracting from overall supersets the current (new) subsets; (11) update the minel-to-superset mapping by adding the links from current minels to their new subsets; end if done or repeat with next patch.

is visualized in Figure 5 and given as pseudocode in Algorithm 1. In each patch, new supersets among the current patch's potential supersets are found for all possible subsets as defined in the batches – and aggregated along with their subsets. When a set is determined subset of a size-k superset, it could be a minimal element, if it is not a superset. Hence we obtain the new minimum elements respective our knowledge of supersets up to this patch by subtracting the overall supersets from the new subsets found during this patch. Since the cardinality of the potential supersets is increasing over the iterations, any subset that was not determined a superset in previous iterations, is known to be minimal. Therefore, the new minimum elements need only be added to the final minel mapping, although later, more discovered supersets for already found minimal elements might be added. The steps taken by this algorithm are the same ones required to build the full subset partial order, because every subset-superset tuple is discovered at some point. However, we save memory and by only storing the relations between minimal elements and their proper supersets. As stated earlier, this yields the bipartite graph that has the same connectivity as the full partial order. Representations that are neither proper sub- not supersets are ignored. They are added later for evaluation.

5 EXPERIMENTAL SETUP

In order to assess our framework on real-world data and tasks, We compare method variants and baselines for duplicate detection of publications, authors and institutions on large-scale datasets.

5.1 Datasets

For publication records, we use the CORE data dump from 2020[2] with 119M publication records [big] using all DOI annotations and removing those that doi.org cannot resolve as well as those that crossref.org does not label either *'book-chapter'*, *'disseration'*, *'monograph'*, *'journal-article'*, *'proceedings-article'* or *'report'*. This removes among others DOIs that refer to complete journals. 24M DOI-annotated records remain along with the unannotated records.

[2]https://core.ac.uk/services/dataset

Further, we use the small [core] gold dataset described in Gyawali et al. [11] for direct comparison.[3] For author mentions, we use the Web of Science from 1980 to mid 2012, with 171M author mentions and 17.5M distinct names [auth]. 7M mentions are annotated with researcherID's (rID). This is the same dataset as used in Backes [2], Backes [3] and Backes and Dietze [4]. For affiliation strings, we deploy a subset of a newer version of the Web of Science in which top-level institutions were assigned to all German affiliation strings by Rimmert et al. [27] and Donner et al. [9], thus resolving 6.5M affiliations to 2K top-level institutions [bfd]. This dataset was used in Backes et al. [5]. To summarize, the following datasets are used:

- **[big]** Cleaned DOIs from *core.ac.uk* for publication records
- **[core]** Original deduplication dataset from *core.ac.uk*
- **[auth]** WoS author mentions annotated by researcher-IDs
- **[bfd]** WoS affiliation strings disambiguated by [9]

5.2 Baselines

We compare different configurations of our method against two baselines: (a) Use only the publication title to determine equivalence, as in Gyawali et al. [11]; (b) use SimHash [8]. SimHash is not a specific method, but a wide range of functions depending on (i) *feature extraction* (we use words and character ngrams of title and author names); (ii) feature weighting (we use uniform); feature hashing (FNV1a), feature hash aggregation (average) and k in k-bits divergence threshold (we use k = 7 out of 8). To summarize, the following baselines are used:

- **exact** Use only the publication title
- **SimHash** Max-7 divergence in averages over uniformly weighted FNV1a hashes of title features
- **base** Separate by representations

5.3 Evaluation Measures

We deploy two different evaluation measures. In addition to the more complicated and expensive evaluation metric used in Gyawali

[3]Not all record ID's used in [core] could be found in [big] and not all DOIs can be resolved or they describe non-individual publication items.

et al. [11] on the [core] dataset, for [big], we compute precision and recall of finding duplicate pairs among all possible pairs. We count as T all true pairs, that is all pairs for which it is known that they are duplicate. We count as P all positive pairs (that are annotated in whichever way), i.e. the respective method under evaluation considers them duplicate. We count as TP the set of all pairs that are both annotated as duplicate and determined to be so by the evaluated method. Then we compute precision as $\frac{TP}{P}$ and recall as $\frac{TP}{T}$. Given duplicate blocks $B \in \mathcal{B}$ returned by the method and gold blocks $G \in \mathcal{G}$ given in the annotation, $P = \sum_{B \in \mathcal{B}} |B|^2$, $T = \sum_{G \in \mathcal{G}} |G|^2$ and $TP = \sum_{B,G \in \mathcal{B} \times \mathcal{G}} |B \cap G|^2$.

5.4 Experiments with Publication Records

Section 3.4 describes the steps (1) representation, (2) specification and (3) generalization. For **representation**, we experiment with straight-forward options of feature extraction to cover a conventional range of entity descriptions with reasonable granularities. We extract the usual author name components (surname, surname+initial and surname+firstname) for the first four authors, publication year ±1 (e.g. *19992000*, *20002001* for the year 2000 to allow for feature overlap with the previous or consecutive year) and either **(1a)** first six title words[4], **(1b)** first six title word-bigrams[5] or **(1c)** first six title character-5grams. (1a) and (1b) are used with our method, while (1c) is used with SimHash as it is usually applied on such *shingles*. In addition, we use the exact-title baseline from Gyawali et al. [11] to check for unwanted divergence in the dataset or evaluation measure. During **specification**, we require as an absolute minimal representation one surname and one title term and either another surname, another term or the author's first initial. For representations that do not meet these requirements, we distinguish two basic strategies that can be easily applied to deal with overly general representations: **(2A)** isolate them into one big block or **(2B)** isolate them into individual mentions (singleton blocks). For **generalization**, we deploy *Integer Linear Programming* as a simple method to derive complete generalization schemes from feature-type-weightings and the following default transformation:

surnames	sur+initial	sur+firstname	terms	years
$4 \longrightarrow 4$	$4 \longrightarrow 4$	$4 \longrightarrow 0$	$6 \longrightarrow 4$	$2 \longrightarrow 1$

This produces all subsets with the respective number of features as well as similarly general transformations with proportional feature-drops for smaller representations. The weightings for *surname*, *sur+initial*, *terms* and *years* are **(3a)** 8-2-8-1, **(3b)** 6-2-4-1, **(3c)** 4-2-6-1 and roughly indicate how much it 'costs' to drop a feature of the respective type. **(3d)** is not to generalize at all. To simulate a realistic scenario and scale, we always disambiguate the entire [big] dataset and evaluate using the annotated mentions[6]. Despite our efforts to consolidate the DOI-annotation, a number of suspiciously large annotated duplicate sets exist in [big]. To evaluate our method on a possibly more realistic definition of duplicates, we compute results while ignoring mentions annotated with DOIs that are observed more often than a threshold t_g between 10 and no-limit. Also, similar to Ravichandran and Vassilvitski [26], we notice a cases where our method suggests unreasonably large blocks and therefore allow it to split all blocks with a size over a threshold t_m of 10^k for $k \in \{1 \ldots 6\}$ into individual representations. While t_g refers to evaluation modes, the t_m establishes different method variants.

[4]Here we use a simple method to ignore stopwords and obtain base-forms.
[5]As above; no bigrams across stopwords or punctuation.
[6]This is different from disambiguating only the annotated mentions because two annotated mentions might only be connected via an unannotated one.

5.5 Experiments with Author Mentions

We represent author mentions either by **(1a)** conventional labelled name parts (surname, 1st-initial, 1st-name, 2nd-init, 2nd-name, 3rd-init, 3rd-name) as in [3, 4] or by **(1b)** these unlabelled as a schema-agnostic representation or **(1c)** character-5grams of the full name. We use (1a) with our method (requiring at least the traditional minimum of surname and 1st-initial during specification) and (1b) or (1c) with SimHash (requiring at least four parts or ngrams). We have also experimented with different generalization schemes, but found that the respective results exhibit no relevant differences.

5.6 Experiments with Affiliation Strings

We represent affiliation strings by **(1a)** labelled parts (as in Backes et al. [5]), **(1b)** unlabelled parts as a schema-agnostic representation or **(1c)** character-5grams. We use (1a) and (1b) in our method as well as (1b) and (1c) with SimHash. During specification, we require as absolute minimum at least one top-level field, one part or four ngrams, respectively. Alternatively, we try a method that selects small representations ($|R| \leq 2$) for specification by applying a threshold ($t = 1$) on how many times a representation is observed relative to how many specifications it has. We do not generalize as we expect that the data holds at least one affiliation string for each top-level institution (such as "*Univ Heidelberg*").

6 RESULTS

6.1 Results for Publication Records

Out of the 119M mentions, we generate (1a) 88M distinct representations using title words, (1b) 89M using title word-bigrams and (1c) 86M with character-ngrams. For word bigrams, the most frequent representation is observed 411K times and contains no feature at all. The next frequent (241K) has only a publication year of 2014. The tenth one (9.7K) contains the word-bigram "legislative document" and the year 1985. All of these are not only observed many times, but are also obvious cases for specification. After identifying such overly general representations, the most frequent unspecified one is observed 8K times and has the features *(init, hofmann_j)*, *(first, hofmann_jutta)*, *(surname, hofmann)*, *(bigram, tree ring)*, *(bigram, ring width)*, *(bigram, picea aby)*, *(bigram, aby kirsten)*, *(bigram, historical object)*, *(bigram, object sample)*, *(year, 20062007)* and *(year, 20072008)*. Different generalization schemes create a multitude of additional subsets for these observed representations. For bigrams, the numbers increase to (3a) 1.1B, (3b) 494M and (3c) 938M, giving us the chance to test our blocking method on very large data. This constant factor implemented as a concrete 5–10-fold increases the likelihood of common subsets that connect similar representations without pairwise comparison. Without generalizations, building the bipartite graph over these (bigram) representations as described in Section 3.4 and detecting its connected components takes about (3d) 1:45h on 8 cores. Likewise, (3a) 21h, (3b) 10h and (3c) 20h. We expect that translating our Python prototype into C++ could increase its speed by an order of magnitude.

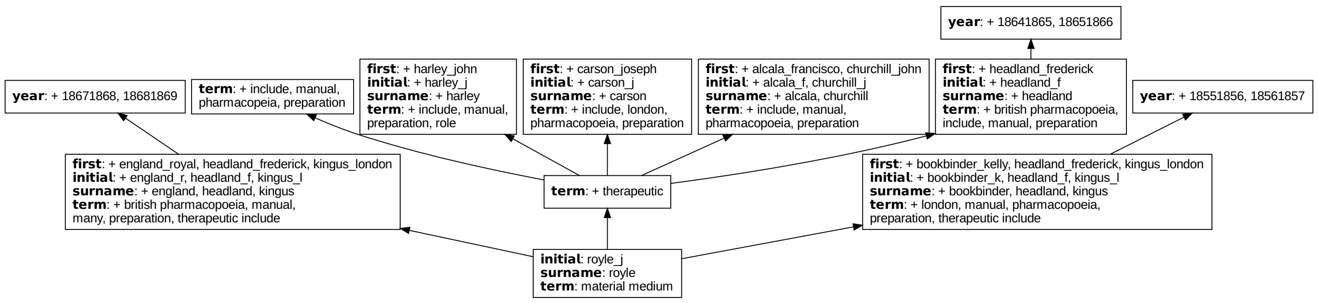

year: + 18641865, 18651866
year: + 18671868, 18681869
term: + include, manual, pharmacopeia, preparation
first: + harley_john
initial: + harley_j
surname: + harley
term: + include, manual, preparation, role
first: + carson_joseph
initial: + carson_j
surname: + carson
term: + include, london, pharmacopoeia, preparation
first: + alcala_francisco, churchill_john
initial: + alcala_f, churchill_j
surname: + alcala, churchill
term: + include, manual, pharmacopoeia, preparation
first: + headland_frederick
initial: + headland_f
surname: + headland
term: + british pharmacopoeia, include, manual, preparation
year: + 18551856, 18561857
first: + england_royal, headland_frederick, kingus_london
initial: + england_r, headland_f, kingus_l
surname: + england, headland, kingus
term: + british pharmacopoeia, manual, many, preparation, therapeutic include
term: + therapeutic
first: + bookbinder_kelly, headland_frederick, kingus_london
initial: + bookbinder_k, headland_f, kingus_l
surname: + bookbinder, headland, kingus
term: + london, manual, pharmacopoeia, preparation, therapeutic include
initial: royle_j
surname: royle
term: material medium

Figure 6: Component referring to '*Materia Medica*' by *J. Forbes Royle* – 'material medium' is a normalization mistake.

For bigrams, there are around 89M blocks (54M of which are singletons) for (3a–c) and 91M for (3d), the latter containing 88M singletons. The three largest blocks contain (3a) 125(32)K / 99(4.2)K / 64(2.4)K, (3b) 82(32)K / 64(4.6)K / 43(3.4)K, (3c) 108(32)K / 105(3.2)K / 82(4.1)K and (3d) 32K / 17K / 7K representations (in brackets only observed representations). Apparently, the largest blocks by all representations are not necessarily largest without the generalizations. Upon inspection, the single largest block looks to be the same for (3a–d). To provide an idea of their content, Figures 7 and 8 show word clouds for the largest and third largest block in (3a). Figure 6 shows the subset partial order for a small block. It is unlikely that the data contains thousands of duplicates for the same record. The two largest blocks using (1b) and (3a) are unfortunate unions of many smaller components that are connected by many individual overlaps (minimum elements). For example Figure 7 reveals a great variance of names, terms and publication years clearly signalling that the corresponding block does not reflect duplication. Here, (2B) can mitigate the damage over (2A), generally increasing precision at the cost of recall by isolating all mentions with the same overly general representation. However, this does not directly tackle the largest blocks mentioned above, which consist of thousands of acceptable representations. For this purpose, we use the earlier described variants that split any blocks with more than k representations into one block per representation. Finally, our task is not to identify actual duplicates, but to separate the data into reasonably sized chunks of similar or related records so that as many duplicate pairs as possible lie within. For example it is fine if all editions of a book lie within one block, even though one might not consider them duplicates. Hence, despite its size of 2.4K distinct observed representations, the third largest block visualized in Figure 8 looks tightly interrelated given that it contains only records by two authors, from a few consecutive years and titled with Cyrillic words.

We evaluate our blocking as a deduplication method, using the DOI-annotated records available and consider recall to be more important than precision for the above reasons. Results on [big] are displayed in Figure 9. The most obvious observation is that overly general representations are better split by mentions in addition to being isolated from other representations (cf. Figures 9a vs. 9b). Precision is much better, while recall remains practically the same. Overall, we can achieve recall between 78–80% identified duplicate pairs when including suspiciously large gold duplicate groups from the annotation, improving to 86% when only evaluating against small ($\leq$ 10) ones. Splitting oversize blocks is also very important as the great differences in variants (colored curves per plot) suggest. A good balance appears to be the green one (split all blocks with more than 1k representations). Using title bigrams instead of words in the representation tends to slightly increase precision and recall. Generalization schemes (3a) 8281 and (3c) 4261 give the best results, although it is probably not worth generating two times more generalizations for about a percentage point higher recall than (3b) 6241. Using no generalizations at all remains an option due to its much lower overhead, but the effect of generalization is visible in its recall being about 2.5 percentage points lower.

Table 2c compares results of (1b) and (3a) ("poset fields") on the smaller [core] dataset used by Gyawali et al. [11]. Here we have tried to implement the evaluation measure that they have described to obtain precision and recall for detection of duplicates and non-duplicates. The similarity between ours and their reported title baseline results suggests that the numbers are roughly comparable. The divergences should also be in part due to the fact that not all record IDs in [core] could be found in [big] (there was also a smaller file corruption in the original CORE dump used for [big]). Our method is further compared to a simple SimHash baseline, which however is different from the one used in Gyawali et al. [11]. Our observation is that SimHash is naturally faster than our approach, however it only finds extremely similar records and on the borderline between similar and dissimilar shows some arbitrariness due to the random distributions inherent to the method. This can be seen when comparing the performance of using block labels (bloc) against that of using exact representations, which is practically the same for SimHash, but clearly different for our method (i.e. a larger

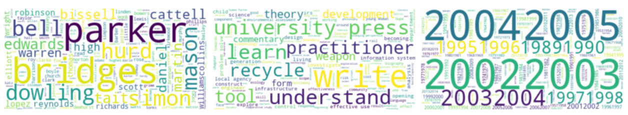


Figure 7: Oversize block: word clouds (surname, terms, year).

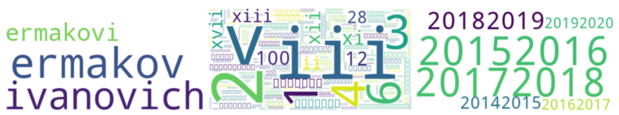
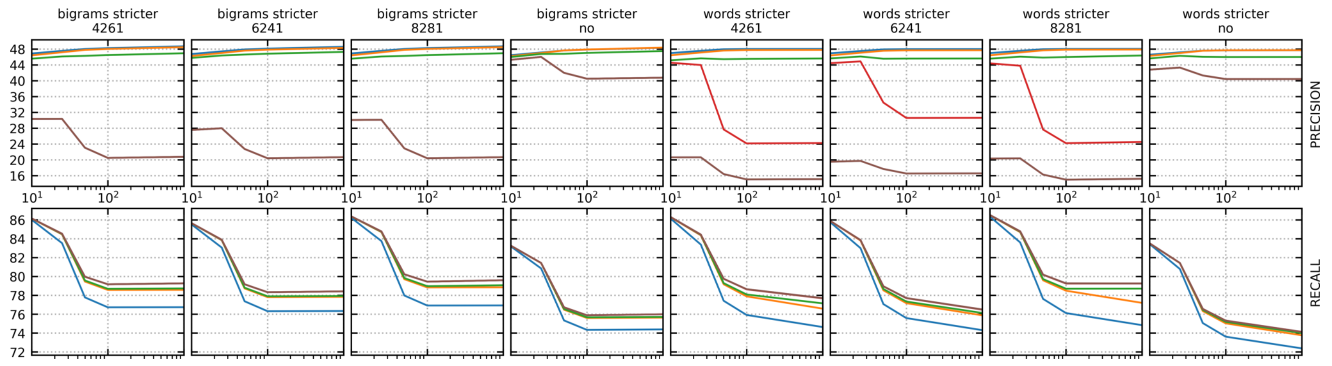
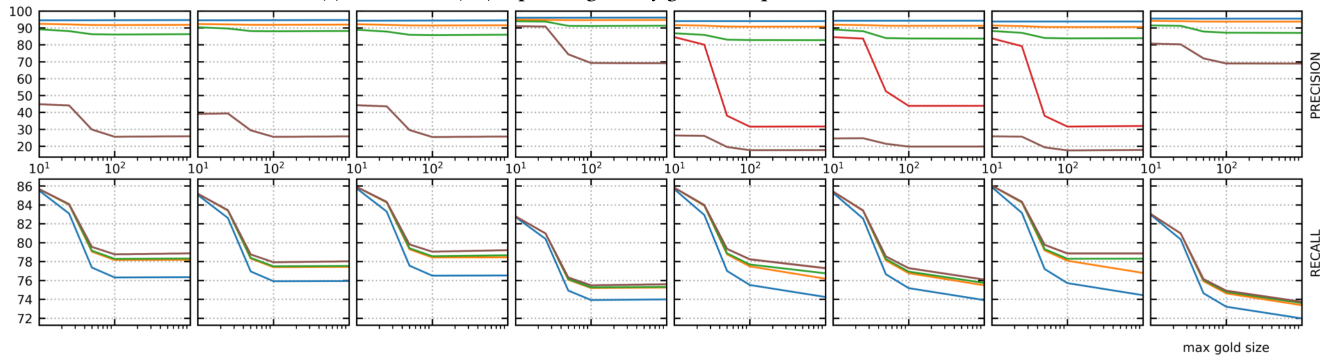


Figure 8: Acceptable block: word clouds (Cyrillic broken).



(a) Results for (2A) separating overly general representations from others.



(b) Results for (2B) separating overly general representations from others and splitting them into one mention per block.

Figure 9: Blocking results compared. Every colored line corresponds to one method configuration and shows its performance in different evaluation scenarios. One column per combination of representation, specification and generalization. One colored line per oversize-block variant. i.e. maximum size 10^1 to 10^5. On the x-axis the lower limit on gold duplicate blocks (by DOI) to be included in the evaluation, i.e. 10^1 to 10^3 (larger values omitted due to no more result changes).

number of different representations are connected into the same block). Interestingly the character-ngram-based representations do better when used alone. Our main finding with this table is that our method's performance looks remarkably similar to that reported for the vector similarity method described by Gyawali et al. [11], but does not require pairwise comparison and instead processes more than 1B records in less than a day, even with Python.

When analysing the errors, we are interested in false-positives (as measured by precision) and false-negatives (as measured by recall). We can obtain a precision value for each block returned by our method. We can obtain a recall value for each gold identifier (DOI) provided by the dataset. A simple size-based average shows that lower precision values unsurprisingly occur mostly in the larger blocks. Small blocks with <10 annotated mentions have an average precision of >80%. Blocks with >100 annotated mentions have a low average precision in the single digits, with occasional exceptions of medium or high precision. This explains the significance of the maximum-block-size threshold described above. On the other hand, one might expect the recall to be high for small duplicate groups and low for larger ones. We do not observe this behavior. Instead, average recall is 9̃0% for size-2 groups, 8̃0% for size-3, 7̃5% for size-4 and 71–73% for sizes 5–10. Lowest averages are centered around size-50 groups while very large groups of sizes >100 have remarkably high recall of 90–100% with a few exceptions. Looking at the baseline that uses simply identical representations as blocks, we find that the larger the true duplicate groups, the greater the contribution of the representations themselves to the recall. This explains the high recall for large duplicate groups as these seem to be especially often the result of very similar or identical records.

6.2 Results for Author Mentions

Out of the 171M mentions, we generate (1a) 17.6M distinct representations using labelled name parts (*'fields'*), (1b) 16.5M using unlabelled name parts and (1c) 13.9M with character-ngrams. As an example, for fields, the most frequent representation is observed 2.8M times and contains no features at all. The next frequent (29.2K) corresponds to *Y. Wang*. The tenth one (18.8K) is for *Y. Liu*. The first one is not only observed many times, but is also an obvious case for specification. The same holds for representations that have only a surname or the first letter of it – although these are not among the most frequent representations. After identifying such overly general representations, the most frequent unspecified one remains *Y. Wang*. Without generalizations, building the bipartite graph over these (field-based) representations and detecting its connected components takes about 8min on 8 cores.

poset		simhash				base	
fields		parts		ngrams			
36	90	85	86	35	35	100	**P**
96	59	60	60	64	64	0.8	**R**
bloc	reps	bloc	reps	bloc	reps		

(a) authors

poset						simhash				base	
fields				parts				ngrams			
1 field		threshold		1 part				4 ngrams			
1.4	94	1.4	94	1.4	94	94	94	98	98	100	**P**
99	8	93	8	100	9	9	9	0.2	0.2	0.0	**R**
bloc	reps	bloc	reps	bloc	reps	bloc	reps	bloc	reps		

(b) institutions

	poset		simhash		exact	
dupli	92	90	91	91	83	**P**
	77	55	59	59	51	**R**
nodup	85	75	77	77	75	**P**
	95	96	96	96	93	**R**
	bloc	reps	bloc	reps		

(c) publications

Table 2: Results for authors on [auth], institutions on [bfd] and publications on [core].

For fields there are around 7.2M blocks (4.7M of which are singletons). The three largest blocks by distinct representations (*S. Lee*, *J. Lee*, *S. Kim*) contain 5736 / 5726 / 4898 representations.[7] See [backes2022a] for some graph visualizations of author names in the subset partial order.

Table 2a compares results of (1a) ("poset fields") on the [auth] dataset used in Backes [2], Backes [3] and Backes and Dietze [4]. Our method is compared to our simple SimHash baseline. Our observation is that SimHash is not useful for blocking author mentions as it cannot understand the matching logics between author names (especially without a weighting of the name parts) and only connects the most similar names. This can be seen when comparing the performance of using block labels (bloc) against that of using exact representations, which is practically the same for SimHash, but clearly different for our method (i.e. a larger number of different representations are connected into the same block). The character-ngram-based representations by themselves have a higher recall than name parts, but much lower precision. This should be mostly due to the limitation of using only the first six features in either case. A more meaningful comparison is between name parts and labelled name parts, where the latter has clearly higher precision. When connecting labelled name parts in the subset partial order, our method increases recall to 96% with considerable loss in precision, where the latter is to be expected in a coarse blocking as targeted here. Under 'base', is shown the result of treating each author mention as a separate author, which shows very low recall and thus underlines the importance of disambiguation. Our main finding for author mentions is that when using a large dataset, it is not necessary to generate the generalizations that are considered the minimal overlap between potential coreferences – most of these are already in the data. We have seen this as the results for all generalization schemes (including generalizing to surname, first initial) were practically the same as not generalizing at all. On the other hand, for specification, the only reasonable specification scheme (just as the literature suggests) is surname, first initial (c.f. Backes [3]), as results were very poor in all other cases. In other words, while it is important to note that our method replicates this proven baseline when surname, first initial is used as maximally general representations during specification, it does not add anything on top (although it was shown in Backes and Dietze [4] how the structure can be further exploited for progressive resolution).

As expected, average precision is high for small blocks and gets much lower with larger suggested blocks. Recall is always very high (almost always over 94%), with a tendency to be even higher (towards 100%) for large blocks. When using individual representations instead of blocks, precision is usually very high independent of the block size, except that for large block sizes, these may correspond to single blocks, and those can occasionally have low precision. Separating by representations instead of blocks, recall is considerably lower for all sizes (mostly 50–70%) after a sharp drop over the smallest few block sizes.

6.3 Results for Affiliation Strings

Out of the 17M affiliations in [bfd], we generate (1a) 1.1M distinct representations using affiliation parsing with labels (*'fields'*), (1b) 961K using unlabelled parts and (1c) 1.9M with character-ngrams. For (1a) fields, the most frequent representation is *{(UNI, Munich)}* and observed 188K times. The next frequent (177K) is *{(UNI, Heidelberg)}*. The tenth one (100K) is *{(UNI, Gottingen)}*. The first one is actually ambiguous as there are two Universitites in Munich. Still, these most frequent representations are very reasonable minimal elements that correspond top-level institutions in the dataset (for example, we have *{(UNI, Berlin), (UNI, Free)}* and *{(UNI,Berlin),(UNI,Tech)}* but no ambiguous *{(UNI, Berlin)}* under the most frequent representations). Two major problems have been noted for the '*separation*' step in Backes et al. [5]: first, there are less frequent, but overly general representations like *{(CLINIC, Univ)}* with 24K mentions. These could potentially be discovered and isolated during specification. Another problem is that even in the real world, almost all larger research institutions in a country are somehow connected by joint departments. In addition, some affiliations might incorrectly reference multiple institutions if the respective author has provided them in one string. The threshold-based specification alternative is quite slow and does not return the desired results. So specification is a very important and challenging step for affiliations. In contrast, generalization is not really necessary as practically all top-level institutions are also used somewhere in the data as an affiliation. Without generalizations, building the bipartite graph over these (field-based) representations and detecting its connected components takes about 66s for [bfd] on 8 cores. When we run it on all (unannotated) Web of Science affiliations, this takes about half an hour.

For (1a) fields there are around 13.5K blocks (9.6K of which are singletons). Unfortunately, all but 20K representations are in the largest block due to the above described high connectivity. For affiliations, minimum elements are better for separation than connected components. Backes et al. [5] gives a detailed discussion

[7]This seems to be a combination of popular Asian surnames that are often combined with various English first names. Normally Asian first names are also diverse, but transcription removes many of the more subtle differences.

of these issues as well as graph visualizations of affiliations in the subset partial order.

Table 2b shows top-level institution resolution results using our method on (1a) labelled ('fields') and (1b) unlabelled ('parts') elements of the affiliation string compared to SimHash on (1b) the latter as well as on (1c) character-ngrams. Parsed elements of the affiliation string are quite precise regardless of whether they are assigned a label or not. Interestingly, character-ngrams have a considerably lower recall, suggesting that the parsing step has already resolved many coreferent but non-identical affiliation strings. As before, the used SimHash method does not connect a significant number of similar representations. As discussed above, the overconnectedness of the data is apparent in the very low precision after finding the connected components with our method.

7 CONCLUSION

In this work, we have motivated, described, implemented and evaluated a new blocking approach that embraces the subset partial order of entity representations to model relationships between blocking keys that correspond to sets of features rather than simple strings. Previous work has used blocking-key-equivalences or total alphabetical orders of blocking keys but these do not accurately model the matching relationship between the sets of features that we propose to use as blocking keys or minimal overlaps / sufficient conditions for equivalent representations. Our approach refines and extends the set of entity representations during the steps *representation*, *specification* and *generalization* introduced in Backes et al. [5] before efficiently building a reduced bipartite version of the subset partial order over them to find connected components. Here we have proposed a new approach to allow parallel minimal element search (or partial order production) with configurable time/space tradeoff that is more convenient than previously proposed alternatives. We have further contributed a large scale silver standard for duplicate detection using automatically cleaned DOI-annotations. For the additional domains of author disambiguation and affiliation resolution, we have also experimented with large-scale datasets. Results are summarized in the following.

For deduplicating publication records, representation efforts such as using normalized word-bigrams instead of words can make a difference, where for example the latter helps to increase blocking precision slightly at no loss of recall and – more importantly – greatly reduces the size of the largest block. Specification aims at avoiding the creation of such oversize blocks by isolating overly general representations either into distinct representations or distinct mentions. Precision is much better for the latter case, at almost no loss of recall. Nevertheless, at least two very large blocks remain that seem to result not from overly general representations but from an overconnectedness that results from a large number of minimal elements. Where necessary, the generalization step helps to moderately increase recall, but the tested schemes increase collection size and processing time by up to one order of magnitude, raising the question of proportionality. On the other hand, the large increase in collection size has allowed us to test-proof that connected components can be found within a reasonable time frame of one day even for more than a Billion representations (see Section 6.1). If a C++ implementation can indeed offer the expected 10x speed-up, processing would only take a few hours. Depending on individual restrictions, our algorithm can be configured to use more or less memory at the expense of processing time. The SimHash baseline can only find the most similar duplicates. Although we have allowed variations in 7 out of 8 hash-bytes, its recall did not increase over that of the underlying representations. In other words, no significant number of similar representations referring to duplicate records were grouped. When comparing to the work of Gyawali et al. [11], it looks as though our method has comparable performance than their pairwise vector-similarity method, which is very promising as the latter is clearly infeasible for any larger collection. While our interpretation of the SimHash methodology has not produced good results for the annotated data at hand, its superior speed was noted. The good performance reported in other publications suggests it can be configured to achieve better results. Hence, further work on comparing the two approaches is necessary.

For author mentions, we have proven experimentally that our method essentially regresses to the established surname, first-initial baseline for pre-separating author mentions if specification ensures that no clearly overly general representations with even less features remain. Labelled name parts are clearly the best features, also rendering the application of SimHash futile as it cannot understand their matching relationships. In other words, specifying any representations more general than surname, first-initial is the only reasonable specification scheme. It was shown not to be necessary to generalize any unobserved surname, first-initial combinations in addition to those that are already in the data. In other words, all generalization schemes performed practically the same as using no generalization. Thus, further exploitation of the subset partial order in Backes and Dietze [4] can be formally stand-alone in that the connections present in the data establish a sufficient 'superclustering' on their own. For affiliation strings, we have reproduced the difficulties observed in Backes et al. [5] and conclude that connected components in this domain suffer in particular from the true interconnectedness of academic institutions, so that once a dedicated specification method can identify incorrect representations to a satisfactory degree, minimal elements (which are also returned by our algorithm) should be used to define overlapping blocks (top-level institutions) instead.

Experimental results suggest that in duplicate detection and author disambiguation, our method offers the expected performance as defined by the vector-similarity baseline in Gyawali et al. [11] and the surname, first-initial baseline (cf. Backes [3] and Backes and Dietze [4]). For top-level institution resolution, our approach has underscored the challenges imminent in the affiliation data (cf. Backes et al. [5]), such as difficult-to-detect underspecified representations and actual institutional interconnections. This shows that our approach not only performs well in the usual domains but also provides new means for describing persistent difficulties that can serve as the basis for further improvements in the future.

ACKNOWLEDGMENTS

This work was supported by the [...] Research Fund of [...] (Number [...]). Additional funding was provided by [...] and [...]. We also thank [...] for contributing [...].

REFERENCES

[1] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. 1972. The transitive reduction of a directed graph. *SIAM J. Comput.* 1, 2 (1972), 131–137.

[2] Tobias Backes. 2018. Effective Unsupervised Author Disambiguation with Relative Frequencies. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries - JCDL '18*. Association for Computing Machinery, 203–212.

[3] Tobias Backes. 2018. The Impact of Name-Matching and Blocking on Author Disambiguation. (2018), 10.

[4] Tobias Backes and Stefan Dietze. 2022. Lattice-based progressive author disambiguation. *Information Systems* 109 (2022).

[5] Tobias Backes, Daniel Hienert, and Stefan Dietze. 2022. Towards hierarchical affiliation resolution: framework, baselines, dataset. *International Journal on Digital Libraries* (2022), 1–22.

[6] Roberto J. Bayardo and Biswanath Panda. 2011. Fast Algorithms for Finding Extremal Sets. In *Proceedings of the 2011 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 25–34.

[7] Panagiotis Bouros, Nikos Mamoulis, Shen Ge, and Manolis Terrovitis. 2016. Set containment join revisited. *Knowledge and Information Systems* 49, 1 (2016), 375–402.

[8] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. Association for Computing Machinery, 380–388.

[9] Paul Donner, Christine Rimmert, and Nees Jan van Eck. 2020. Comparing institutional-level bibliometric research performance indicator values based on different affiliation disambiguation systems. *Quantitative Science Studies* 1, 1 (Feb. 2020), 150–170.

[10] Marta Fort, J Antoni Sellarès, and Nacho Valladares. 2013. Finding extremal sets on the GPU. *J. Parallel and Distrib. Comput.* 74, 1 (2013), 1891–1899.

[11] Bikash Gyawali, Lucas Anastasiou, and Petr Knoth. 2020. Deduplication of Scholarly Documents using Locality Sensitive Hashing and Word Embeddings. In *Proceedings of the 12th Language Resources and Evaluation Conference*. European Language Resources Association.

[12] Monika Henzinger. 2006. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. American Association for Computing Machinery, 284–291.

[13] John Hopcroft and Robert Tarjan. 1973. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM* 16, 6 (1973), 372–378.

[14] Ravindranath Jampani and Vikram Pudi. 2005. Using prefix-trees for efficiently computing set joins. In *International Conference on Database Systems for Advanced Applications*. Springer, 761–772.

[15] Charles E Leiserson, Marc Moreno Maza, Liyun Li, and Yuzhen Xie. 2010. Parallel computation of the minimal elements of a poset. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*. Association for Computing Machinery, 53–62.

[16] Jizhou Luo, Wei Zhang, Shengfei Shi, Hong Gao, Jianzhong Li, Wei Wu, and Shouxu Jiang. 2019. FreshJoin: An Efficient and Adaptive Algorithm for Set Containment Join. *Data Science and Engineering* 4, 4 (2019), 293–308.

[17] Yongming Luo, George HL Fletcher, Jan Hidders, and Paul De Bra. 2015. Efficient and scalable trie-based algorithms for computing set containment relations. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 303–314.

[18] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*. American Association for Computing Machinery, 141–150.

[19] Martin Marinov and D Gregg. 2014. A practical algorithm for finding extremal sets up to permutation. *Journal of Experimental Algorithmics* 9, 4 (2014).

[20] Martin Marinov, Nicholas Nash, and David Gregg. 2016. Practical algorithms for finding extremal sets. *Journal of Experimental Algorithmics (JEA)* 21 (2016), 1–21.

[21] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. *Comput. Surveys* 53, 2 (May 2020), 1–42.

[22] Paul Pritchard. 1991. Opportunistic algorithms for eliminating supersets. *Acta Informatica* 28, 8 (1991), 733–754.

[23] Paul Pritchard. 1995. A simple sub-quadratic algorithm for computing the subset partial order. *Information processing letters* 56, 6 (1995), 337–341.

[24] Paul Pritchard. 1997. An old sub-quadratic algorithm for rinding extremal sets. *Inform. Process. Lett.* 62, 6 (1997), 329–334.

[25] Paul Pritchard. 1999. On computing the subset graph of a collection of sets. *Journal of Algorithms* 33, 2 (1999), 187–203.

[26] Deepak Ravichandran and Sergei Vassilvitski. 2021. *Evaluation of Cohort Algorithms for the FloC API.* Technical Report. Google Research & Ads.

[27] C. Rimmert, H. Schwechheimer, and M. Winterhager. 2017. *Disambiguation of author addresses in bibliometric databases.* Technical Report. Bielefeld University.

[28] Iztok Savnik, Mikita Akulich, Matjaž Krnc, and Riste Škrekovski. 2021. Data structure set-trie for storing and querying sets: Theoretical and empirical analysis. *Plos one* 16, 2 (2021).

[29] Hong Shen. 1998. Fully dynamic algorithms for maintaining extremal sets in a family of sets. *International Journal of Computer Mathematics* 69, 3-4 (Jan. 1998), 203–215.

[30] Hong Shen and David J Evans. 1996. Fast sequential and parallel algorithms for finding extremal sets. *International journal of computer mathematics* 61, 3-4 (1996), 195–211.

[31] Chengcheng Yang, Dong Deng, Shuo Shang, Fan Zhu, Li Liu, and Ling Shao. 2021. Internal and external memory set containment join. *The VLDB Journal* 30, 3 (2021), 447–470.

[32] Daniel M Yellin. 1992. Algorithms for subset testing and finding maximal sets. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*. American Association for Computing Machinery and Society for Industrial and Applied Mathematics, 386–392.

[33] Daniel M Yellin and Charanjit S Jutla. 1993. Finding extremal sets in less than quadratic time. *Information processing letters* 48, 1 (1993), 29–34.

8.4 Discussion

8.4.1 Erratum

This version of the paper that had been submitted to VLDB contains an crucial typo in Section 2.3: Here, the runtime bound is not $O(N/\log N)$, but $O(N^2/\log N)$.

8.4.2 Elaboration

A number of visualizations could not be used in the current version of the paper due to space constraints. They are described in the following.

In Figure 8.1, we display the relationships between different type of sets in the subset partial order. In our case, each set corresponds to one entity representation. In the inner circle, we see that each set is either a proper subset or a maximal set, meaning that maximal sets are those sets that are no proper subsets of any other set, i.e. they have no proper supersets. In the next circle, we see that the set of all proper subsets is partitioned further into those that are both subsets and supersets of some other sets, and those that are only subsets. The set of all maximal sets on the other hand is further partitioned into those that are not supersets of any other set (in addition to not being subsets by definition) and those that are supersets. The former are therefore singletons in the subset relation, i.e. they are not connected to any other representation. We also

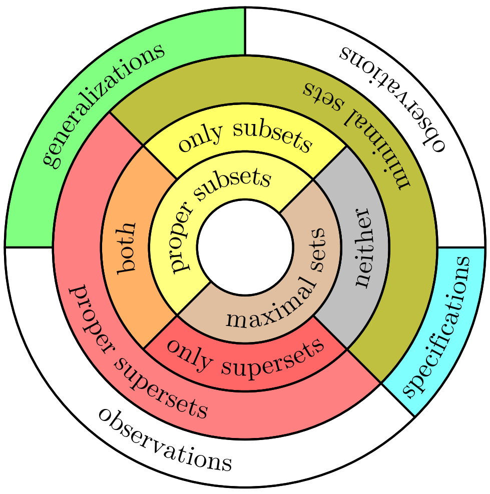


Figure 8.1: Every set that is not a proper superset is minimal. Every set that is not a proper subset is maximal. Proper subsets can also be proper supersets ('intermediate' in poset), otherwise they are minimal. Minimal sets need not be proper subsets, in which case they are 'isolated' in poset. Blocking generalizations can, but need not be minimal. Observations can also be minimal. Specifications are isolated.

see that all of these four partitions are mutually exclusive. In the third circle from the center, we see that proper supersets are obviously the union of all sets that are only supersets and all sets that are both sub- and supersets. On the other hand, we see that minimal sets are the union of all sets that are only subsets and all sets that are neither sub- nor supersets. We also see that each set is either a proper superset of a minimal set. In addition to the above general definitions, we also project the different type of sets as defined in our blocking method into this visualization in the outer circle. Observed representations can be any type of set described above. Generalizations are proper subsets, where they can be only subsets or both sub- and superset. However, they are only useful to increase connectivity if they are only subsets, otherwise they can be ignored. Specifications on the contrary are maximal sets, but specifically those that are neither sub- nor supersets because we *replace* the original underspecified representation by one that contains the representation- or mention-ID. Thereby the respective representations are isolated because no other mention can have the same ID in its representation. In theory, it could be possible to specify in a more sophisticated way so that the specification could be superset of other representations, or even an acceptable subset of others. However, this was not done in our work.

Figure 8.2 compares our algorithm for parallel minimal element search (on the left) against an alternative by Leiserson et al. [2] (on the right). The matrices align potential subsets and potential supersets. Each cell corresponds to one superset check. Our approach simply does the superset check for all potential supersets that are larger than the current potential subset. This creates one iteration for each potential superset size, starting from 2. Since sets of the same size cannot be proper supersets of each other, each phase can be partitioned into independent batches that are processed in parallel. The recursive algorithm by Leiserson et al. [2] on the other hand is more complicated. Here, both potential subsets and potential supersets are partitioned into batches in the beginning. First, each batch is compared to itself, looking for minimal elements. Then batches are merged in pairs ($A \cap B$, $C \cap D$). Now we need to check if the minimal elements from the individual batches merged have subsets in the other batch, respectively. If not, they remain as minimal elements of the union. Then we proceed recursively. The problem is that finding subsets is more expensive than finding supersets and recursive parallelization is problematic to implement. We had implemented this method, but were not convinced that it was convenient when trying to use it on our large datasets.

Figure 8.3 simply shows on the left how representations are mapped to a list of features and on the right how this mapping can be inverted to give all representations that contain a certain feature. This structure can be used in Boolean retrieval tasks like finding all supersets of a potential subset as the intersection of the posting lists for all features in the potential subset. We build such an index for each iteration of our parallel minimal element search algorithm. Except for the initial iteration, it is built in parallel to using the previous index while searching for supersets in that respective iteration.

Table 8.1 gives insights into our duplicate detection silver standard by providing a histogram over the different publication types retrieved from *crossref.org* for which DOI's are present. Here, we selected only *report*, *monograph*, *dissertation*, *book-chapter*, *proceedings-article* and *journal-article*. Any records for which the DOI was not found in both *doi.org* and *crossref.org* are also excluded.

In Figure 8.4, we give a visualization of a report from running our minimal element search algorithm on weighting 4261 in duplicate detection. The overall runtime of the Python implementation amounted to roughly 20 hours in this case. The green part is the parallellized computation, so the corresponding bars would be much taller with only one worker. Getting struct (blue) refers to the inverted index overhead, which is relatively large for small potential subsets. As can be seen, getting the results from the results queue takes particularly long for the first patch of one superset size, because here

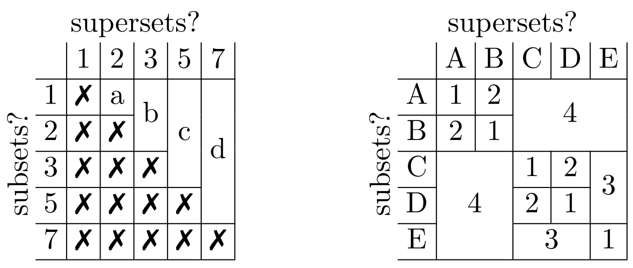


FIGURE 8.2: Algorithms for parallel minimum element search. Left, 1-7 indicate item-set cardinalities and a-d phases. Left our algorithm, right recursive algorithm by Leiserson et al. [2]. Right, A-E indicate batches and 1-9 the recursive steps.

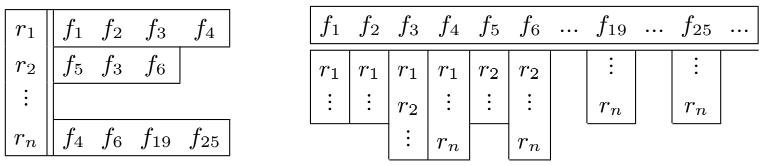

r_1
r_2
r_n
f_1 f_2 f_3 f_4
f_5 f_3 f_6
f_4 f_6 f_{19} f_{25}
f_1 f_2 f_3 f_4 f_5 f_6 … f_{19} … f_{25} …

FIGURE 8.3: Representations and their features (left) and the inverted index (right).

TABLE 8.1: Histogram of crossref types for the distinct DOIs in [big] that doi.org could resolve. Note that these numbers count duplicate dois only once.

TYPE	**#**	**TYPE**	**#**	**TYPE**	**#**
book-track	1	proceedings	1,163	book	93,042
book-section	3	reference-book	2,538	***report***	147,567
journal-volume	5	journal	4,062	***monograph***	157,740
book-set	12	report-series	4,350	***dissertation***	207,747
book-part	18	other	7,109	***book-chapter***	644,464
peer-review	22	dataset	8,071	***proceedings-article***	654,575
proceedings-series	42	reference-entry	11,555	component	1,775,775
book-series	72	journal-issue	14,664	none	3,518,251
standard	73	posted-content	18,589	***journal-article***	14,542,546

the smaller potential subsets are queried, which are more likely to have supersets, so that there are more results for the first patches.

Finally, we have not described in detail how we have obtained the features in the representation step. In the core data dump, the authors are given as a list of strings, where each string corresponds to one author name. Mistakes like "*Royal England*" are unfortunately included here. If there is a comma in the name string, we split on comma and take everything before it as surname and the rest as forenames. If there is no comma, we split on whitespace amd take the last part as surname. In addition, we consider local prepositions like "*de*", "*de la*" or "*van*" as indicators of surnames to correct potential mistakes. Further, we create a database of the frequency of surnames and firstnames in the core data and exchange firstnames and surname if the supposed surname is more frequent as a forename and the supposed first forename is more frequent as a surname. Finally, we normalize to lower case ASCII letters and replace a few no-name terms like anonymous or unknown by NULL. For the publication year, we take the current year and add the surrounding years by string concatenation before interpreting the result as an integer, e.g. with *2000*, we get 19992000 and 200020001. Then, if another representation is published one year earlier, they still share the feature 19992000. For title terms or bigrams, we use the following procedure: First, we split the title by indicators of subtitles, like "*:*" and each of these parts into sections by stop-words (which are thus removed). We tokenize each section and normalize the words into ASCII. Then we look up each word in WordNet and Symspell[1]. If the word is not known in both, we use Symspell to suggest the most similar alternative (if similar enough) in order to fix typos. This sometimes creates different words, especially if the original term was not wrong, but from a non-English language (e.g. "*materia medica*" which was changed to "*material medium*" in Figure 6). However since the words are only used for feature-overlap and the same term will be changed everywhere in the same way, the damage should be limited. For bigrams, we continue to lemmatize all words that are in the dictionary after this transformation by the most frequent part-of-speech in WordNet and get the section-wise word-bigrams. If a section is only of length one (e.g. a word that was surrounded by two stopwords) we take this word instead of a bigram. All the above feature transformations have the effect of making the representations slightly more general than if the information were used exactly as given.

8.4.3 Shortcomings and Potential Improvements

The Python implementation of the parallel algorithm for minimal element search is probably much slower than a good C++ implementation could be. Its performance should

[1]https://github.com/wolfgarbe/SymSpell

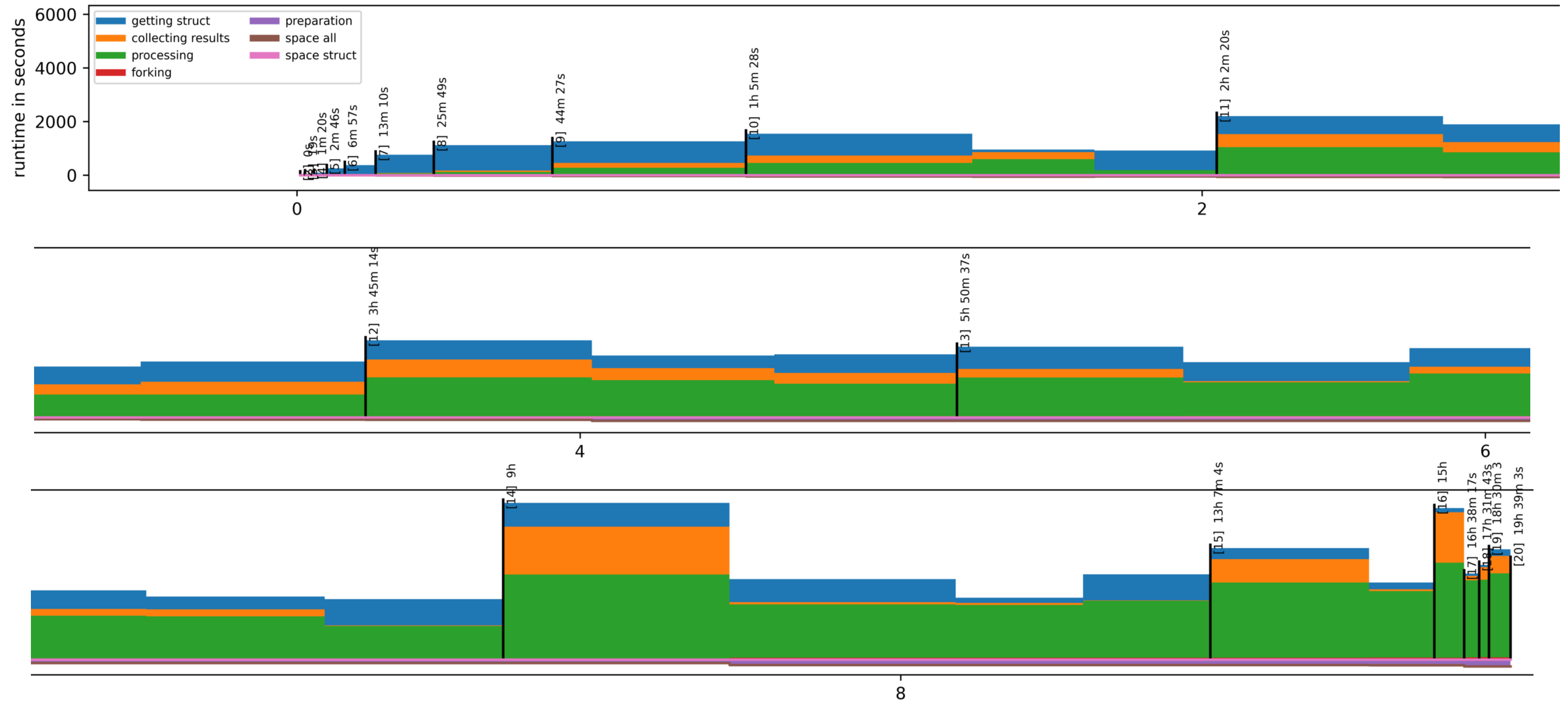


FIGURE 8.4: Visualization of our minimal element search algorithm on weighting 4261 with just under 1B representations. In [square brackets] the size of the supersets currently searched. One stacked bar per patch. The heights of the green bars indicate the actual processing time for this patch. The y-axis is proportional to the number of potential supersets (numbers refer to x hundred million), i.e. the patch size. For example for large potential supersets, there are less candidates, but it takes longer to find them. If the remaining number of potential supersets for one specific [size] is less than two times the desired patch size, it is cut in half.

be much more impressive then. Unfortunately, I do not have the required programming expertise without considerable learning efforts to acquire them. More problematic is the Python-related memory waste. The algorithm exploits the fact that the UNIX operating systems use *copy-on-write* to copy objects in memory only when they are being modified. This allows to use the same inverted index from different independent processes as long as it is not changed. Unfortunately, to detect zero-references that will be dropped, Python garbage collection keeps object reference counters not in a central registry, but with each object it creates. This unfortunate design-choice leads to objects being partly modified by reference. As a result, growing parts of the inverted index are duplicated in memory, leading to large memory consumption that is a factor of the number of parallel processes. The only reason this was feasible is because we have only used 8 parallel processes and because we have an extremely large RAM available, i.e. more than one TB (!). For proper deployment on very large data, the algorithm should definitely be translated in a more suitable programming language, like C++.

Another shortcoming is the fact that regardless of the efforts in the specification step, there are still a small number of very large blocks produced. In previous versions, these were much larger and we have made considerable improvements in the latest version. However at least the largest two blocks are resisting breaking up the over-connectedness. Although evaluation results become very good when simply breaking them up into individual representations using a maximum block size, this is not a very satisfying solution as it contradicts the goals we are trying to achieve in the first place by connecting representations using the subset partial order.

The evaluation could have compared many more alternative blocking methods that were described in the related work section. However, this would have been extremely difficult as most of them introduce a plethora of configuration choices and would have to be re-implemented in most cases. When implementing the SimHash baseline, we noticed that there is no unique definition of this method meaning that even for this one method, there are many different variations possible. Therefore, we have relied on using (a) a non-proprietary large dataset [big] that can be used as a benchmark by others in the future and a small dataset [core] that has already been used to evaluate a few alternatives in Gyawali et al. [96]. Due to inconsistencies in their data, we could only roughly replicate their small dataset, but similar results for the title-baseline suggests that the two benchmarks are sufficiently similar. Another problem for comparability was that they have used a very expensive, complicated and unintuitive evaluation measure, which we can only hope to have properly reproduced from their description. Finally, it would have been good had we been able to reproduce their reported results because there are some surprising numbers: They combine SimHash and vector-similarity, where the former has a precision of roughly 70% and poor recall of 25% (that being consistent with our findings) and the latter 91% and 78%, respectively. For some reason, combining the low-recall SimHash method with the 78% recall vector-similarity method achieves higher recall of 83%. While this is in theory possible if the returned pairs are complementary to a larger degree, it seems rather unlikely to assume that a larger portion of the highly similar SimHash pairs were dissimilar by vector-similarity.

8.4.4 Evaluation

The following datasets were used for evaluation:

big	The self-prepared core dataset with roughly 24M annotated and 95M unannotated records.
core	The reproduced dataset from Gyawali et al. [96], using the core-IDs from their small dataset of aligned duplicates to look up the features [big].
auth	The Web of Science author dataset used in Chapters 4, 5 and 6.
bfd	The top-level silver-annotated institution dataset maintained at Bielefeld University [97] that was used in Chapter 7.

The main evaluation in this work concerned duplicate detection. Using the generalization step, the size of the [big] dataset was increased to over 1B representations for some configurations. Here we have tested the following method configurations:

1. Representation: (a) words or (b) bigrams
2. Specification: at least one surname and one title term and either another surname, title term or the one author's first initial, further distinguish (A) isolate by representation or by (B) mentions.
3. Generalization: Use weightings (a) 8281, (b) 6241, (c) 4261 or (d) none

Results on [big] suggest a good choice is (1b) bigrams, (2B) and modest generalization (e.g. (3b) 6241) if one takes speed into account – otherwise 8281 gives best results.

Comparing on [core] (Table 2), our best configuration showed performance remarkably similar to the vector-similarity method results reported by Gyawali et al. [96] (92/77 vs. 91/78), while being feasible for large datasets. For non-duplicate detection, results are varying a bit more (85/95 vs. 86/99). As the datasets are not exactly identical due to the inconsistencies on their behalf, it was reassuring to see the exact-title baseline results being very similar as well (83/51 vs. 83/50). Again non-duplicate results vary more

(75/93 vs. 71/99), posing the question whether there were unintentional divergences in the non-duplicate evaluation method. While we do not beat their combined method, we had some doubts about these reported numbers (see above) and more importantly, their combined method is clearly infeasible for large-scale blocking as it requires pair-wise comparison. Our SimHash interpretation is clearly different from theirs (91/59 vs. 70/25), where it should be noted that our version did not connect a significant number of different representations, as indicated by the *reps* baseline.

For authors, representation was clearly best with the usual author name parsed components (surname, initials, firstnames). For specification, it became also clear (if it was not already so) that there is no reasonable alternative to using surname and first-initial as the minimum required features. Generalization turned out to be irrelevant w.r.t. the results although it could still make sense to also generalize to surname, first-initial in the cases where such representation is not already in the data, especially if the dataset is smaller. So for author names, our results clearly support the use of the surname, first-initial baseline for super-blocking, which can be easily described in our framework. The fact that no other static blocking scheme presents itself as an alternative while at the same time we know surname, first-initial is at times too expensive (e.g. Wang, Y) supports our work in Chapter 6 where each such super-block is progressively processed further without overloading the clustering method.

For affiliations, we have compared (1a) labelled affiliation string components (*fields*), (1b) unlabelled ones (*parts*) and (1c) character 5-grams (*ngrams*). During specification, we generally assume a minimum of 1 field, 1 part or 4 ngrams respectively. However, as we knew from the work described in Chapter 7 that separation is problematic, we have also tried using a *threshold*-based method to detect overly-general representations that should be specified. Since we knew that the problem is overgeneralization rather than the opposite, we have not applied a generalization step for affiliations. Our results suggest that the labelling of affiliation string components (1a) is not useful (although it was important in the conflation step in Chapter 7) compared to using the string components as such (1b). The threshold version of generalization did not improve the poor precision, but it is possible that this was because of actual interconnections of the institutions. Using SimHash does again not improve over using identical representations as blocks. Overall, the problems regarding separation of different top-level institutions as described in Chapter 7 were reproduced and they remain a considerable challenge.

Chapter 9

Conclusion

In this thesis, the entity resolution problem in the context of digital libraries was studied from different viewpoints to obtain a conceptually sound framework that satisfies a number of desired properties stated in Chapter 1.2. This chapter first summarizes the process of developing this approach and obtaining the required findings through the course of the publications that make up this thesis (Section 9.1). Here we give details on how the individual works have contributed to incorporate the desired properties and how individual aspects can be viewed in the light of existing concepts described in Chapter 2.1. This allows us to repeat the main results of our work by stating how the desired properties described in the beginning of this thesis were incorporated. Next, we summarize limitations inherent to our approach and suggest alternative methods that may be better suited in certain situations (Section 9.2). In addition, for each of the individual research projects, open questions and straightforward next steps have remained as collections of points for future work. Towards the end of the conclusion (Section 9.3), we stress the main contributions of our work, briefly outlining the formalized solution proposed in Chapter 2.2 by describing each of the framework's components in terms of the central features they introduce and the implications they have for the research field of entity resolution. Finally, we give some closing remarks in Section 9.4.

9.1 Summary and Main Results

The framework proposed in this thesis is the result of multiple consecutive research projects, which are summarized in the following.

In the first work, we have studied the problem of author equivalence in a clustering setting. We have implemented a simple agglomerative clustering method based on probabilistic similarities. The result suggested that the probabilistic similarities in conjunction with agglomerative clustering work well for the task of grouping author mentions describing the same real-world person, where the most difficult question was when to stop the agglomerative clustering process. In addition, we have studied properties of the clustering algorithm's solution finding process as well as general coherences of evaluating author names, which has lead to the proposal of evaluating problem size range individually to control for the most crucial data property on the reported performance. Regarding pre-clustering partitioning (blocking), we have chosen the *surname, all-initials* scheme without evaluating the impact of this decision.

This apparent shortcoming has led to the second work in which we study in detail the effect of intransitive name-matching and transitive blocking on the performance of author disambiguation and present a first model to make explicit the relationship between the two (somewhat incompatible) relations. Here, each name-based representation of an author mention constitutes the intension of the respective node in the respective surname's semilattice. Its extension is the mentions represented by it. We have compared performance of the classic categorization-based blocking schemes to our own methods that remove edges in the matching-based blocking graph based on different criteria to obtain smaller connected components to be used as blocks. Except for the *surname, all-initials* scheme, the classic blocking methods can also be expressed by removing edges in this graph. The results suggested that the *surname, all-initials* baseline is a good choice in terms of precision and recall as well as expensiveness, but the proposed entropy-based edge-removal worked also comparatively well. Among others, it was also found that almost all true equivalences are between matching names, which means that name-matching can indeed be considered a necessary condition for equivalence, even if it it certainly not sufficient and its transitive closure inevitably moves a number of contradictory representations into the same block.

Hence, we have continued to explore the matching-based graph induced by the subset partial order over mention representations in the third paper, where we have further refined the formalization as well as the methodology. A major focus in this context was to embrace the concept of progressive entity resolution. It was identified that instead of removing weak edges in the matching-based blocking graph, adjacent nodes representing individual blocks can be successively merged depending on the strength of the edge weights between them. This creates graph minors by means of edge contraction. The partial order of mention representations can be considered a concept hierarchy over author names and the edge weights are computed in a very similar way as the confidence measure from association rule learning. The covering relation over the representations is a way to structure the search space by name-matching. However, not all matches are necessarily modelled as we only order the *observed* representations, which can lead to some missed matches, but allows to avoid the quadratic complexity of deciding matching by pairwise comparison and the even greater complexity of adding all possible unobserved representations. For example the matching of *John H. Doe* and *J. Herbert Doe* is only noticed if we also observe *J. H. Doe* (or a generalization thereof) or *John Herbert Doe* (or a specification thereof). A number of different edge-weighting and edge-weight modification schemes were compared to traditional fixed blocking schemes by plotting their performance against the number of pairwise comparisons suggested by them. This revealed a superior efficiency in terms of precision and recall vs. number of comparisons for the proposed progressive scheme with the preferred variant of edge-weight modification over traditional fixed schemes.

In the fourth project, we have studied a new type of entity, namely author affiliations to be grouped into real-world institutions. In addition, we have focused on making use of the concept hierarchy of affiliation representations such that it approximates the true institutional hierarchies under the respective top-level institutions, which would allow both to infer these hierarchies from the data and at the same time to order the observed affiliations hierarchically, enabling sub-organizational bibliometric comparisons. Here, the edge-contraction previously used in the context of progressive resolution is reinterpreted as a means to merge affiliation representations that describe the same sub-organizational unit. This project has also helped to further segregate the tasks involved in our approach into components of a pipeline. We have identified the sub-tasks *representation* (to create meaningful representations), *separation* (to obtain a more principled super-blocking previously done by *surname, first initial*), *interpolation* (to hypothesize intermediate unobserved representations if required), *collocation* (partial order production), *conflation* (merging through edge contraction) and *evaluation.* This division holds not only for the task of hierarchical affiliation resolution, but for our framework in general. The concrete results of the project are mixed as some sub-tasks were found to be very difficult due to challenging deficiencies and variations in the affiliation strings. Error propagation deteriorates results in later sub-tasks. However the structuring of the problem and the detailed error analysis have laid a solid foundation for future improvements.

In the fifth and last project, we have focused entirely on the separation problem. Again, a new entity type is studied, namely the publication records themselves. The corresponding application is duplicate detection. The proposed method is to find the connected components (as a means of separation) in the subset partial order of observed mention representations and some of their unobserved generalizations. The latter represent the minimum required overlap for each observed representation to be considered for duplicate-verification. The proposed technical algorithm allows parallellization under consideration of custom space limitations, i.e. runtime can be traded off for memory consumption. The algorithm detects minimal elements in the subset partial order on the fly and creates a reduced bipartite graph that links each representation with its minimal elements, which can then be fed to standard connected component algorithms in the form of a sparse matrix that requires considerably less space then the full partial order. Still, the algorithm can also be used to efficiently build the partial order, acting as a means for partial order production. The main contribution of this last project is to ensure that our approach can scale to at least one billion entity mentions.

Our proposed framework presents an answer to the research question how a single, conceptually sound entity resolution framework can integrate all the desired aspects stated earlier. To this end, we restate beneficial properties of ER methods along with an explanation on how our approach features them.

Scaling: 1 billion records A parallellized super-blocking algorithm that finds connected components in the subset partial order can be used to separate the data into smaller subsets without losing any information, as the later steps are also based on the subset partial order and any mention representations that are not connected will also not interfere in any way.

Scaling: progressiveness A progressive framework for block merging is deployed within super-blocks. This is particularly important in cases with increased connectedness resulting in large super-blocks. Then, only the most promising mention pairs are compared until the time runs out.

Scaling: avoid pairwise view Although every clustering algorithm performs pairwise comparison, we ensure that our system's pair selection process ultimately implements a modular decomposition of the complete data. The collection is split into super-blocks, which in turn are split into individual representations. These are progressively merged to larger blocks and in each iteration each block is clustered into a number of clusters that represent real world entities. Thereby, we induce a partitioning by super-blocks, one by blocks and one by clusters. We do not make any independent assertions for individual mention pairs that might establish a non-transitive duplicate relation contradicting the transitive property of the target equivalence relation.

Scaling: parallellizable The super-blocking algorithm was developed with strict regard to enabling parallelization. This includes the ability of trading off memory consumption with some additional runtime. On the blocking level, for each iteration of the progressive resolution, each super-block would have to be loaded, blocks be merged and re-clustered. The overhead of these loading operations was not studied. Developing new parallel *clustering* methods was not the goal of this work, but it should be noted that such do indeed exist (e.g. [98]) and present an interesting opportunity for further parallelization.

Scaling: unsupervised While we have used small classification scenarios in some of our works to estimate parameters, nowhere do we assume the resolution process to be a pairwise classification problem. Instead, the modular decomposition of the collection is driven by different notions of similarity which are theoretically derived and compared to one-another, rather than obtained from a large parameter space through supervised learning.

Adaptivity: schema-aware or -agnostic Our approach is built at its very core on the concept of set-based mention representations (that are ordered by the subset partial order). These sets *can* contain attribute-value pairs, where the attributes may implement a schema, but they can also contain only values, leading to a schema-unaware method. In fact, values and attribute-value pairs can be mixed in the same representation, allowing maximal schema-awareness without requiring this kind of information were it is simply not available.

Adaptivity: model missing values The subset partial order that is at the core of our approach models exactly the logic of missing values if attribute-value pairs are used as features. A representation with missing values is a lower bound of a more complete representation, implementing the fact that the first still matches the latter by assuming that values missing in one representation could in principle be the same as the those observed in its supersets.

Adaptivity: model feature-expressiveness Being a key component for adaptivity, automatic feature weighting on top of feature-sharing is crucial. It is implicitly used both in the probabilistic cluster similarities (in $\frac{1}{\#(f)}$) and in the blocking graph weights, were a representation consisting of general features is more likely to have many low-weighted specifications and an added rare feature's higher discriminating power is reflected in a lower edge weight (and the corresponding higher likelihood of separation) due to the respective representation being less observed.

Adaptivity: embrace world knowledge Our framework allows encoding world knowledge in different components like representation parsing, generalizing observed representations or the introduction of unobserved minimal elements to introduce potential block-defining candidates.

Comprehensiveness: generalize over other approaches Especially in the related literature chapter, we have pointed out how our approach generalizes other common approaches. For example blocking keys can be added as hypothesized representations that can then be identified as minimal elements to tie together all representations that are specifications thereof. Any clustering method can be used in the clustering step. In the blocking papers, static blocking baselines have been explained, implemented and compared within in our framework. We have presented a matrix-view on entity resolution that generalizes most blocking methods, in particular also our blocking approach by using it to determine the representation-representation matrix. Finally, we have presented a bipartite graph view on clustering and shown that (our) blocking approach can also be explained in it.

Soundness: model transitivity Using block merging and clustering instead of a pairwise view on blocking and/or a pairwise-classification for verification, transitivity is modelled easily as all intermediate results constitute equivalence relations that are automatically transitive. The conflict between the intransitive notion of matching and the transitive nature of equivalence has been made explicit in the blocking graph, which was one of the main motivations for deriving it through the subset partial order in the first place.

Soundness: identify well-defined sub-problems In the description of our approach, we have been able to reduce many sub-problems to known basic tasks, mostly in the domain of graph algorithms. Among them are

- representation parsing
- partial order production
- extremal element search
- connected component search
- strongly connected components
- edge contraction
- graph minors
- modular decomposition
- minimum spanning tree
- transitive reduction
- transitive closure
- agglomerative clustering

Accessibility: simple The entire core of our framework is based on the directed acyclic graph induced by the subset partial order over mention representations as well as a small number of operations on this graph (e.g. connected component search, edge contraction, see above).

Accessibility: modular Although everything is build around this core, the tasks can be arranged in a pipeline consisting of the components

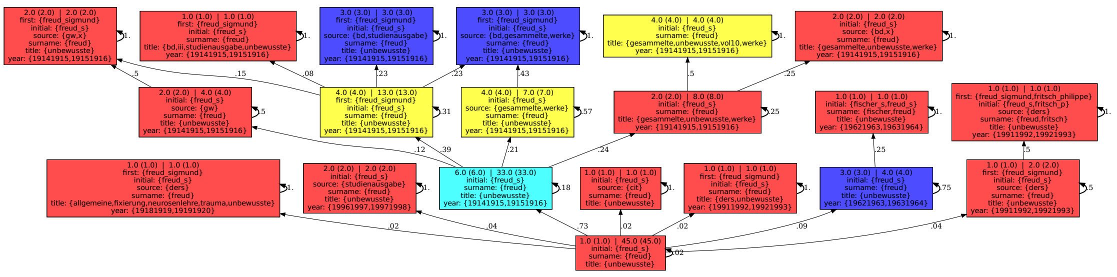

2.0 (2.0) | 2.0 (2.0) first: {freud_sigmund} initial: {freud_s} source: {gw,x} surname: {freud} title: {unbewusste} year: {19141915,19151916}
1.
1.0 (1.0) | 1.0 (1.0) first: {freud_sigmund} initial: {freud_s} surname: {freud} title: {bd,iii,studienausgabe,unbewusste} year: {19141915,19151916}
1.
3.0 (3.0) | 3.0 (3.0) first: {freud_sigmund} initial: {freud_s} source: {bd,studienausgabe} surname: {freud} title: {unbewusste} year: {19141915,19151916}
1.
3.0 (3.0) | 3.0 (3.0) first: {freud_sigmund} initial: {freud_s} source: {bd,gesammelte,werke} surname: {freud} title: {unbewusste} year: {19141915,19151916}
1.
4.0 (4.0) | 4.0 (4.0) initial: {freud_s} surname: {freud} title: {gesammelte,unbewusste,vol10,werke} year: {19141915,19151916}
1.
2.0 (2.0) | 2.0 (2.0) initial: {freud_s} source: {bd,x} surname: {freud} title: {gesammelte,unbewusste,werke} year: {19141915,19151916}
1.
.5
.15
.08
.23
.23
.43
.5
.25
2.0 (2.0) | 4.0 (4.0) initial: {freud_s} source: {gw} surname: {freud} title: {unbewusste} year: {19141915,19151916}
.5
4.0 (4.0) | 13.0 (13.0) first: {freud_sigmund} initial: {freud_s} surname: {freud} title: {unbewusste} year: {19141915,19151916}
.31
4.0 (4.0) | 7.0 (7.0) initial: {freud_s} source: {gesammelte,werke} surname: {freud} title: {unbewusste} year: {19141915,19151916}
.57
2.0 (2.0) | 8.0 (8.0) initial: {freud_s} surname: {freud} title: {gesammelte,unbewusste,werke} year: {19141915,19151916}
.25
1.0 (1.0) | 1.0 (1.0) initial: {fischer_s,freud_s} surname: {fischer,freud} title: {unbewusste} year: {19621963,19631964}
1.
1.0 (1.0) | 1.0 (1.0) first: {freud_sigmund,fritsch_philippe} initial: {freud_s,fritsch_p} source: {ders} surname: {freud,fritsch} title: {unbewusste} year: {19911992,19921993}
1.
.12
.39
.21
.24
.25
.5
1.0 (1.0) | 1.0 (1.0) first: {freud_sigmund} initial: {freud_s} source: {ders} surname: {freud} title: {allgemeine,fixierung,neurosenlehre,trauma,unbewusste} year: {19181919,19191920}
1.
2.0 (2.0) | 2.0 (2.0) initial: {freud_s} source: {studienausgabe} surname: {freud} title: {unbewusste} year: {19961997,19971998}
1.
6.0 (6.0) | 33.0 (33.0) initial: {freud_s} surname: {freud} title: {unbewusste} year: {19141915,19151916}
.18
1.0 (1.0) | 1.0 (1.0) initial: {freud_s} source: {cit} surname: {freud} title: {unbewusste}
1.
1.0 (1.0) | 1.0 (1.0) first: {freud_sigmund} initial: {freud_s} surname: {freud} title: {ders,unbewusste} year: {19911992,19921993}
1.
3.0 (3.0) | 4.0 (4.0) initial: {freud_s} surname: {freud} title: {unbewusste} year: {19621963,19631964}
.75
1.0 (1.0) | 2.0 (2.0) first: {freud_sigmund} initial: {freud_s} source: {ders} surname: {freud} title: {unbewusste} year: {19911992,19921993}
.5
.02
.04
.73
.02
.02
.09
.04
1.0 (1.0) | 45.0 (45.0) initial: {freud_s} surname: {freud} title: {unbewusste}
.02

FIGURE 9.1: Example of a connected component resulting from a quick application of our proposed framework to publication references extracted from fulltext PDFs of the SSOAR repository (https://www.gesis.org/ssoar/home). In contrast to many other ER approaches, our method allows to show the matching relationships and their consequences in concise visualizations like this one.

1. representation
2. separation
3. interpolation/generalization
4. collocation
5. conflation (and verification)
6. evaluation

Accessibility: visualizable The blocking graph with its representation nodes can be visualized well, especially for smaller subsets. It has been shown in different contexts how the plotted graph can be used for error analysis and to obtain a better understanding of the data, the involved logical relationships and the general problem of matching vs. equivalence. Overall, the graph visualization of the subset partial order with the option of weighted edges can convey a large amount of information at one glance (see for example Figure 9.1), which distinguishes it for example from the usual aggregations over total (alphabetical) orderings used in other approaches.

9.2 Limitations, Alternatives and Future Work

An advantage of our modular framework is that it allows to address shortcomings by optimizing individual tasks independently. Such potential improvements are listed as future work later in this section. First, we shall focus on limitations that are inherent to our approach in that they cannot be overcome by improving individual modules. Although some of them may be better modelled by alternative methods, this does not diminish our proposal's overall usefulness. In our partial-order-based approach, similarities between entity mentions are always based on discrete notions of set similarity, such as set containment (i.e. they are defined in an L1 space), which is for example different for *vector-space methods* or similarity measures that are defined in higher dimensional space like *Euclidean (L2) distance*. Despite modelling logical subsumption, the partial order does not separate contradictory representations like *Jack Doe* vs. *John Doe* if a generalization like *J. Doe* is present. This can only be modelled if transitivity is applied after verification, in which case pairs (*John Doe*,*J. Doe*) and (*Jack Doe*,*J. Doe*) can be suggested without including (*Jack Doe*,*John Doe*). Typos and other minor string differences are better discovered by a alphabetical total ordering of the features as for example in *Sorted Neighborhood Blocking*. In our approach, these need to be addressed directly in a normalization preprocessing step or indirectly by generalizing representations in the hope that the dropped features are the ones with typos or other minor string differences. Consequently, arguably the main disadvantage of our method is that it does not consider feature overlaps (only containments) unless they are specifically generated as hypothetical generalizations beforehand or are already present in the data. When mentions are represented by large sets of features (e.g. all abstract or fulltext character n-grams), then all acceptable overlaps cannot be generated as there are too many of them. In such cases, *SimHash* is a better solution. SimHash is a faster way to find very similar representations with many features of the same type and is also easier to implement. In addition, our method requires some human input for guiding the feature extraction or designing rules for representation parsing. In general, the larger and less structured the representations are, the less of a point is there in our approach's logic-based modelling through the subset partial order and schema-agnostic methods like *Attribute Cluster Blocking* are probably more appropriate. In progressive edge contraction, the resulting graph minors do not inherently avoid redundant comparisons which is in contrast to methods like *Progressive Block Scheduling* and many others that propose one pair after the other and compute the transitive closure afterwards (e.g. with a *Union-Find data structure*). Progressive edge contraction requires to keep the semilattices in memory, while other methods like *Ordered List of Records* require less memory to simply suggest the next mention pair to compare. For hierarchical institution resolution, an *entity-linking* approach would be more appropriate if the required gold hierarchies are available. Finally, our approach is not based on machine-learning. Although our representations essentially constitute conjunctions and the possibility to add multiple representations or generalizations thereof as keys for each mention allows to define necessary conditions for equivalence in disjunctive normal forms, we have not elaborated on whether these could be (better) learned by *DNF-learners*.

Due to time limitations and the fact that any research work will inevitably produce further follow-up questions and ideas, this thesis is not comprehensive. In the following, we list the most important points that have been noted for further consideration in future work and assign them to the chapters that have given raise to them or where they best fit content-wise.

9.2.1 Future Work in Clustering

Regarding the clustering step, it would be advisable to develop a faster method than agglomerative clustering, for example by exploring and pushing the performance limits of single-link clustering, which is a very fast clustering method and has a number of convenient properties, like being decomposable into minimal pairwise decisions. When using agglomerative clustering, it has been shown that the calculation of the stopping criterion is essential and needs better automatic adaption to individual disambiguation tasks. Regarding features used for clustering, it is desirable to include word embeddings into whichever clustering method is used, in order to better model semantic similarity. Finally, a proper large-scale benchmark is required to allow fair and solid comparison of various author disambiguation approaches.

9.2.2 Future Work in Matching and Blocking

Despite considerable normalization efforts done by the Web of Science itself, many author names are not optimally represented, which means that an improved parsing and normalization step for author names would be useful. In addition, a number of possible variations could be interpolated for each observed author mention (e.g. add *John Doe* when observing *Johnathan Doe*.

9.2.3 Future Work in Progressive Blocking

Perhaps the most urgent missing part in the work on progressive author disambiguation is to test a baseline that simply merges blocks in top-down order, i.e. that first merges the blocks with the most specific names and then continues with the next most specific ones, etc. In retrospect, it appears likely that the preferred variant used in the experiments comes very close to this behavior. Another interesting method to compare to the current set of variants and baselines would be to use a biased random walk on the blocking graph using the edge weights as transition probabilities to sample paths that hypothesize block equivalences. Another important point is to implement a truly parallel progression that proceeds iteration by iteration through all super-blocks and schedules new blocks to be clustered without using up an enormous amount of memory and producing an I/O bottleneck. In this context a possible improvement is to use the progressive nature of agglomerative clustering to integrate it with the block merging progression in order to obtain a finer-grained resolution progress. Furthermore, it is important to reduce the redundancy resulting from constantly re-clustering merged blocks. In this context, the ec* evaluation x-axis should be swapped against its integral to show not only the effort required in one iteration, but in all iterations up to this point. Also, a stopping criterion for when further progression is more or less futile could be developed. Finally, both dynamicity (clustering/verification results are fed to the next iteration's blocking step) and real-time ER (processing a changing collection by updating the results when mentions are added or removed) would be desirable to achieve a convincing applied solution.

9.2.4 Future Work in Hierarchy Extraction

As discussed in the respective chapter, our work on hierarchical institution resolution left a number of open questions and plenty of room for improvement. In particular, the representation component should be addressed as a proper parsing problem to produce high quality affiliation representations. World knowledge about real-world institutions could be included by consulting comprehensive resources like the *World Higher Education Database* and adding top-level representations derived from their names to the collection together with a number of possible variations, so that observed affiliations can be better contextualized against this background of known institutions. The interpolation component would benefit from some improved rules that can build on the more accurate new representations. The most challenging component of conflation could certainly use an improved pairwise equivalence classifier that determines whether two adjacent representations are equivalent. Finally, the evaluation step should be improved by annotating more gold hierarchies.

9.2.5 Future Work in Connected Components

Although this work was conducted close to the end of the thesis project, we have already noticed a number of shortcomings that could not be addressed within a reasonable timeframe (see Chapter 8.4.3). We expect that it would be particularly rewarding to create a C++ implementation of our algorithm for minimal element search, as that should be much faster than the current Python code. On the research side, the specification and generalization step should attract further attention to improve precision and recall of duplicate detection and split the remaining oversize blocks. This might also be achieved by modifications in the representation step, although we see less room for improvement there, with the exception of language-dependent normalization. It seemed that the largest blocks in duplicate detection were not due to overly general representations, but a result of general interconnectedness, similar to the problem in top-level institution resolution. However, it still needs to be investigated whether for publication records, this is simply a result of many less obvious cases of overly general minimal sets, rather than just one or a few. In affiliation resolution, even clearly overly general representations like *UNIV:*{*Clin*} can not yet be reliably detected, so this would be the first task concerning improvements in this application scenario. However, we would rather see this as future work of the previous project. Next, the use of minimal elements to define overlapping blocks could be studied further where affiliation resolution presents a very interesting test-bed, due to the actual overlays in top-level institutions. For author disambiguation, this work has simply confirmed what was already assumed in our respective work described in Chapter 6. Although it would require great effort, evaluating or approach against more state-of-the-art duplicate detection methods would be an important contribution. Perhaps the large CORE gold standard created by us could be used as a benchmark to allow a competition of alternatives without having to re-implement each. From a more technical point of view it would also be very interesting to explore further the sparse-matrix-based view on blocking as used in the related work chapter to generalize all kinds of blocking methods. This would also allow us to better compare different methods as they essentially constitute parameter variations of this model. In addition, this would allow us to study a pairwise view on duplicate detection that is not necessarily transitive (i.e. by observing the filled cells in the output matrix before applying the transitive closure). From a more application-oriented point of view, we have to remember that our proposed method is actually just a blocking step for duplicate detection, so it would be interesting to use an actual definition of duplicates and realize detection of the latter. Finally, as a mix between technical challenge and application-side requirements, it would be very useful to define and implement an algorithm that makes the current blocking method dynamic, i.e. by adding new records to the right block as they come in, without reprocessing the entire collection.

9.3 Contributions and Implications

The work described in this thesis is centered around the topic of blocking for Entity Resolution. Given the studied application scenarios, it can be contextualized particularly well in the research fields of author disambiguation, institutional disambiguation and duplicate detection. Although there is plenty of literature on blocking techniques in general and dedicated domain applications in specific, we have still found there to be gaps in terms of systematic, overarching conceptualizations as well as adequately modelling missing information in a logically coherent manner. Consequently, we have addressed these gaps by proposing a number of systematic views on blocking and a new method of arranging blocking keys in partial orders. Concerning the former, we have discussed blocking in terms of graph algorithms, matrix operations, logical concepts, information systems and mathematical abstractions. This includes a bipartite graph view on clustering and blocking, extremal set search, graph minors through edge-contraction, a sparse-matrix view, necessary and sufficient conditions for equivalence, rough sets as well as bounds in continuous/discrete spaces, topology and space partitioning. Concerning the latter, we have developed a modular blocking framework comprised of the steps representation, specification, generalization, collocation and conflation that can be integrated with clustering in a progressive way to make it adaptable to requirements of large-scale collections in different domains. In particular, we have addressed gaps in understanding

1. ...the clustering process of author disambiguation.
2. ...the relationship between blocking and clustering in author disambiguation.
3. ...the logical relationship between author names.
4. ...progressive blocking opportunities for author disambiguation.
5. ...the nature of affiliation strings particularly concerning institutional hierarchies.
6. ...feasibility and suitability of partial order components for de-duplicating very-large-scale publication metadata collections.

As highlights of the conducted research can be summarized that our overarching views on blocking allow for systematic method comparison and re-combination in this field, improving evaluation, reproducibility, optimization and innovation. Further, our partial order approach offers a novel, expressive view on key relations that we have proven successfully applicable particularly in the domain of author disambiguation, but also to affiliations and publication records.

The contribution of this thesis lies in the development of a theoretical and implemented framework for entity resolution with a number of desired properties. The approach has been guided and evaluated by various experiments during the course of these studies. The development process has been documented in a sequence of publications that constitute important milestones, address different application scenarios and focus on individual sub-problems. In the following, we summarize the mechanics of our approach, interpreting its individual aspects as our contributions and stating their advantages over conventional methods to underline their relevance.

In **representation**, a representation is parsed from the entity mention string, or the corresponding record. For author disambiguation, this may be based on the author name information. For (hierarchical) institution resolution, this may be based on the individual components of the affiliation string corresponding to different institutional functions. For duplicate detection, any features that indicate record equivalence like title terms or author names may be used. This means that our approach fosters human-readable entity representations that can be configured to express exactly the aspects deemed relevant for the respective domain at hand. As an implication, better understandability, interpretability and domain-adaptivity is now available for entity resolution, which shows adequate appreciation for domain knowledge and invites the involvement of experts.

In **generalization**, these observed representations are generalized to possibly unobserved representations that constitute what is assumed to be a minimal required overlap for equivalence (a necessary condition). For author names, this may be *surname,first-initial* or *surname-only*, so that representations of author mentions with these commonalities end up in the same super-block even if such general representations are not observed. For (hierarchical) institution resolution, these can be representations of top-level institutions or those of intermediate hierarchical nodes, where the latter are often not observed (for example few people would use the faculty-level to state their affiliation, while both the university, or a specific chair under the faculty are more likely to be found in the data). For duplicate detection, one may simply drop a number of insignificant features, or those that are known to experience a lot of variation in the data. Generalization allows the user to control exactly what are the overlaps necessary to compare two mentions while also enabling less costly and specific rules through globally configured feature-dropping schemes. This has the same implications regarding understandability, interpretability and domain-adaptivity as mentioned for the representation step.

In **separation**, we compute a relation that links all representations to their minimal elements in the subset partial order. This relation has the same connectivity as the subset partial order but a lot less edges. In this bipartite graph, connected components can be computed. Anything that is not connected is also not in the subset partial order and can thus be handled separately in the following steps, which improves efficiency and enables our approach to scale to billion entity mentions. Alternatively, the minimal elements can be used as blocking keys, so that overlapping blocks are created, each of which consists of all the specifications of such minimal element. In author disambiguation, we have assumed minimal elements of *surname,first-initial* or *surname-only*, without consulting the data, because we have assumed any more general representation to be underspecified to the extend of being unusable. In these cases, the logic of author names results in the fact that either surnames or surname-first-initial combinations always create a partitioning of author names without overlaps. In (hierarchical) institution resolution, we have separated the data by minimal elements, which were assumed to correspond to top-level institutions. Duplicate detection was the task where we have opted clearly for connected components as opposed to minimal elements to obtain super-blocks. Here, they were also considerably smaller than in the other application scenarios. Oversize blocks become apparent in our method without causing computational problems. Often they are simply the result of a consequent modelling of the matching relation's transitive nature and would also be composed by other block-building methods. Our approach makes the transitive connectedness of entity representations explicit and visible and thereby avoids unpleasant surprises later on, when it may otherwise happen that very large blocks lead to infeasible computational loads or never-ending suggestions of more and more pairs to compare. As an implication, consequences both rooted in the data and resulting from the decisions made in the above steps are clearly unveiled in the process of adapting our approach to a specific application scenario. This avoids disappointing users with unexpected follow-up problems when transferring the approach to other contexts and thereby generally improves reusability of ER methods, which benefits the development of the entire research field as well as the general advancement of such technology's application.

In **collocation**, we simply do partial order production. The algorithm used to detect minimal elements during separation can also be used to compute the subset partial order over a set of representations (in our case over a super-block). In addition, the representations' observation counts can be used to compute edge weights for the edges that correspond to the covering relation of the partial order. This step at the core of our approach is the same procedure for all three application scenarios. To the best of our knowledge, it is the first suggestion that properly models the logical matching relationships between entity representations in domains like author disambiguation. As an implication, better overall results become available, reducing mistakes to more intricate problems rather than trivial inadequacies of the similarity measures used or the thresholds applied on them. Consequently, work in the ER fields can focus on more interesting logical and representational problems rather than fine-tuning conventional similarity-based methods and achieve real gains in knowledge and understanding.

In **progressive merging** or **conflation** many different strategies are possible. What they have in common is that they perform edge contraction on the covering relation to obtain a graph minor. This is the same as adding reverse edges in the covering relation and computing the strongly connected components to obtain the graph's condensation. Usually, the edge contraction is done progressively based on the edge weights, where first all edges are contracted with a weight higher than the threshold and then the threshold is lowered. In addition, the edge weights might be modified during the progression. In author disambiguation, we have proposed a number of different edge weight modification schemes and evaluated their performance for progressive author disambiguation. In (hierarchical) institution resolution, we have interpreted the conflation step as a means to merge equivalent affiliations. In this case, no progression was done, but the weights were simply binary in the sense that they indicate equivalence or not, so that the result would be a graph closer to the true institutional hierarchy. In duplicate detection, conflation corresponds to an additional (optional) step in verifying potential duplicates. One implication of the fact that our approach lends itself well to this kind of progressive resolution is that we have been able to present the first progressive blocking method specifically designed to model the logical relationship between author names in author disambiguation. In general terms, it is now possible to combine the important paradigm of progressive resolution with transparent and adequate logical matching relations, enabling their application on larger datasets than before.

In **clustering**, the goal is to take each block yet unprocessed and apply an expensive clustering method to it that would otherwise be infeasible for larger portions of the data. Any clustering method can be used here. In author disambiguation, we have seen good results for agglomerative clustering rather than single-link clustering. In (hierarchical) institution resolution, this task was completely omitted as it does not offer a way to improve the hierarchical mapping that we were looking for. In duplicate detection, clustering offers a final third step after separation and conflation for even better avoidance of false-positives, i.e. higher precision. The additional blocking opportunities contributed through our approach allow for the application of better and more expensive clustering methods on smaller and more concise blocks. As an implication, improved resolution through expensive clustering methods is now attainable for larger datasets than before.

9.4 Closing

In this thesis, we have studied the task of entity resolution or more specifically, how to efficiently find the most similar pairs of entity mentions without comparing all mention pairs. We have established that conventional (hash-based) blocking is limited in modelling intransitive matching relationships while alternative heuristics like alphabetical order that are sometimes used to progressively suggest pairs make the incorrect assumption that coreference likelihood is approximated by a total ordering. This status quo is problematic because it bypasses the true logical matching problem and suggests to fine-tune inherently flawed notions of equivalence likelihood, thereby pushing ER research into a niche which provides little in the way of actual insights. Our assumptions have been proven both theoretically (by exposing obvious contradictions) and empirically (by outperforming traditional methods) over the course of five projects, whereof four have as of yet been published in top-tier venues. In these, we have overcome the above mentioned limitations of traditional approaches by suggesting to arrange entity mentions in the subset partial order of their set-based representations, which we have found to be better suited for modelling logical matching relationships. In particular the second and third project have contributed to the formal elaboration of the partial-order framework, deriving it bottom-up from the logical matching relations between author names, while the fourth and fifth have contributed further details and solutions for the outstanding task of separation and the framework's final modular structure. The first work has anticipated the requirement for a deeper understanding of the clustering processes happening inside the blocks returned by the rest of the framework and has contributed an effective unsupervised clustering method. This research has allowed us to propose a complete novel ER approach which uniquely combines a large number of beneficial properties in a modular and easily adaptable framework. Its specification, implementation and application for real-world scenarios has provided insights into (progressive) clustering development, matching relationships, matching likelihood, hierarchical relationships and subset-based connectivity in large datasets. Finally, a number of open questions and promising next steps could be summarized for each of our framework's modular steps and provide exciting opportunities for future work to further exploit, refine and challenge our suggestions and assumptions.

Bibliography

[1] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys*, 53(2):1–42, May 2020. ISSN 0360-0300, 1557-7341. doi: 10.1145/3377455. URL https://dl.acm.org/doi/10.1145/3377455.

[2] Charles E. Leiserson, Marc Moreno Maza, Liyun Li, and Yuzhen Xie. Parallel computation of the minimal elements of a poset. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation (PASCO'10)*, page 53, Grenoble, France, 2010. ACM Press. ISBN 978-1-4503-0067-4. doi: 10.1145/1837210.1837221. URL http://portal.acm.org/citation.cfm?doid=1837210.1837221.

[3] I.E.A. Data centres and data transmission networks, 2020. URL https://www.iea.org/reports/data-centres-and-data-transmission-networks.

[4] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web*, pages 141–150, 2007.

[5] Yuhang Zhang, Kee Siong Ng, Tania Churchill, and Peter Christen. Scalable entity resolution using probabilistic signatures on parallel databases. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*, pages 2213–2221, Torino, Italy, 2018. ACM Press. ISBN 978-1-4503-6014-2. doi: 10.1145/3269206.3272016. URL http://dl.acm.org/citation.cfm?doid=3269206.3272016.

[6] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. To compare or not to compare: Making entity resolution more efficient. In *Proceedings of the International Workshop on Semantic Web Information Management*, pages 1–7, 2011.

[7] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems*, 65:137–157, April 2017. ISSN

03064379. doi: 10.1016/j.is.2016.12.001. URL https://linkinghub.elsevier.com/retrieve/pii/S030643791530199X.

[8] Yasser Altowim and Sharad Mehrotra. Parallel progressive approach to entity resolution using MapReduce. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 909–920, San Diego, CA, USA, April 2017. IEEE. ISBN 978-1-5090-6543-1. doi: 10.1109/ICDE.2017.139. URL http://ieeexplore.ieee.org/document/7930035/.

[9] Uwe Draisbach, Peter Christen, and Felix Naumann. Transforming pairwise duplicates to entity clusters for high-quality duplicate detection. *Journal of Data and Information Quality (JDIQ)*, 12(1):1–30, 2019.

[10] Ioannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. MDedup: Duplicate detection with matching dependencies. *Proceedings of the VLDB Endowment*, 13(5):712–725, January 2020. ISSN 2150-8097. doi: 10.14778/3377369.3377379. URL https://dl.acm.org/doi/10.14778/3377369.3377379.

[11] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.*, 53(6), dec 2020. ISSN 0360-0300. doi: 10.1145/3418896. URL https://doi.org/10.1145/3418896.

[12] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *Proceedings of the VLDB Endowment*, 9(4):312–323, December 2015. ISSN 21508097. doi: 10.14778/2856318.2856326. URL http://dl.acm.org/citation.cfm?doid=2856318.2856326.

[13] Giovanni Simonini, George Papadakis, Themis Palpanas, and Sonia Bergamaschi. Schema-agnostic progressive entity resolution. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2018. ISSN 1041-4347. doi: 10.1109/TKDE.2018.2852763. URL https://ieeexplore.ieee.org/document/8403302/.

[14] Yumeng Ye and John Talburt. The effect of transitive closure on the calibration of logistic regression for entity resolution. *Journal of Information Technology Management*, 10(4), January 2019. doi: 10.22059/jitm.2019.270013.2324. URL https://doi.org/10.22059/jitm.2019.270013.2324.

[15] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. Progressive duplicate detection. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1316–1329, May 2015. ISSN 1041-4347. doi: 10.1109/TKDE.2014.2359666. URL http://ieeexplore.ieee.org/document/6910276/.

[16] D H Judson. A partial order approach to record linkage. In *Proceedings of Federal Committee on Statistical Methodology Conference*, 2001.

[17] George Papadakis, George Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. Three-dimensional entity resolution with JedAI. *Information Systems*, 93:101565, November 2020. ISSN 03064379. doi: 10.1016/j.is.2020.101565. URL https://linkinghub.elsevier.com/retrieve/pii/S0306437920300570.

[18] Tobias Backes. Effective unsupervised author disambiguation with relative frequencies. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries (JCDL '18)*, pages 203–212, Fort Worth, Texas, USA, 2018. ACM Press. ISBN 978-1-4503-5178-2. doi: 10.1145/3197026.3197036. URL http://dl.acm.org/citation.cfm?doid=3197026.3197036.

[19] Tobias Backes. The impact of name-matching and blocking on author disambiguation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*, page 803–812, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360142. doi: 10.1145/3269206.3271699. URL https://doi.org/10.1145/3269206.3271699.

[20] Tobias Backes and Stefan Dietze. Lattice-based progressive author disambiguation. *Information Systems*, 109:102056, 2022. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2022.102056. URL https://www.sciencedirect.com/science/article/pii/S0306437922000473.

[21] Tobias Backes, Daniel Hienert, and Stefan Dietze. Towards hierarchical affiliation resolution: Framework, baselines, dataset. *International Journal on Digital Libraries*, May 2022. ISSN 1432-1300. doi: 10.1007/s00799-022-00326-1. URL https://doi.org/10.1007/s00799-022-00326-1.

[22] Lars Kolb, Ziad Sehili, and Erhard Rahm. Iterative computation of connected graph components with MapReduce. *Datenbank-Spektrum*, 14(2):107–117, 2014. ISSN 1618-2162, 1610-1995. doi: 10.1007/s13222-014-0154-1. URL http://link.springer.com/10.1007/s13222-014-0154-1.

[23] A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. doi: 10.1137/0201008. URL https://doi.org/10.1137/0201008.

[24] Mayank Kejriwal and Daniel P. Miranker. A DNF blocking scheme learner for heterogeneous datasets. *arXiv:1501.01694 [cs]*, January 2015. URL http://arxiv.org/abs/1501.01694. arXiv: 1501.01694.

[25] Mikhail Bilenko, Beena Kamath, and Raymond Mooney. Adaptive blocking: Learning to scale up record linkage. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 87–96, Hong Kong, China, December 2006. IEEE. doi: 10.1109/ICDM.2006.13. URL http://ieeexplore.ieee.org/document/4053037/. ISSN: 1550-4786.

[26] Mayank Kejriwal and Daniel P. Miranker. An unsupervised algorithm for learning blocking schemes. In *2013 IEEE 13th International Conference on Data Mining (ICDM'13)*, pages 340–349, Dallas, TX, USA, December 2013. IEEE. ISBN 978-0-7695-5108-1. doi: 10.1109/ICDM.2013.60. URL http://ieeexplore.ieee.org/document/6729518/.

[27] Matthew Michelson and Craig A Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, volume 6, pages 440–445, 2006.

[28] Kunho Kim, Athar Sefid, and C. Lee Giles. Learning CNF blocking for large-scale author name disambiguation. In *Proceedings of the first Workshop on Scholarly Document Processing*, pages 72–80, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.sdp-1.8. URL https://www.aclweb.org/anthology/2020.sdp-1.8.

[29] Ting Liu, Charles Rosenberg, and Henry A Rowley. Clustering billions of images with large scale nearest neighbor search. In *Proceedings of the 2007 IEEE Workshop on Applications of Computer Vision (WACV'07)*, pages 28–28. IEEE, 2007.

[30] Levent Ertoz, Michael Steinbach, and Vipin Kumar. A new shared nearest neighbor clustering algorithm and its applications. In *Proceedings of the Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining*, volume 8, 2002.

[31] Pasi Franti, Olli Virmajoki, and Ville Hautamaki. Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(11):1875–1881, 2006.

[32] Ian Davidson and SS Ravi. Intractability and clustering with constraints. In *Proceedings of the 24th International Conference on Machine Learning*, pages 201–208, 2007.

[33] Alvaro Monge and Charles Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records, 1997.

[34] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: A generic approach to entity resolution. *The VLDB Journal*, 18(1):255–276, January 2009. ISSN 1066-8888, 0949-877X. doi: 10.1007/s00778-008-0098-x. URL http://link.springer.com/10.1007/s00778-008-0098-x.

[35] Staša Milojević. Accuracy of simple, initials-based methods for author name disambiguation. *Journal of Informetrics*, 7(4):767–773, 2013.

[36] Donatella Firmani, Barna Saha, and Divesh Srivastava. Online entity resolution using an oracle. *Proceedings of the VLDB Endowment*, 9(5):384–395, 2016.

[37] Cheng Chen, Daniel Pullen, Reed H. Petty, and John R. Talburt. Methodology for large-scale entity resolution without pairwise matching. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 204–210, Atlantic City, NJ, USA, November 2015. IEEE. ISBN 978-1-4673-8493-3. doi: 10.1109/ICDMW.2015.197. URL http://ieeexplore.ieee.org/document/7395672/.

[38] Bingyi Zhong and John Talburt. Using iterative computation of connected graph components for post-entity resolution transitive closure. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 164–168, December 2018. doi: 10.1109/CSCI46756.2018.00039.

[39] K Chidananda Gowda and G Krishna. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern recognition*, 10(2):105–112, 1978.

[40] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems*, 42(3):1–21, August 2017. ISSN 0362-5915, 1557-4644. doi: 10.1145/3068335. URL https://dl.acm.org/doi/10.1145/3068335.

[41] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. *ACM SIGMOD Record*, 24(2):127–138, May 1995. ISSN 0163-5808. doi: 10.1145/568271.223807. URL https://doi.org/10.1145/568271.223807.

[42] Xubo Wang, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. Leveraging set relations in exact set similarity join. *Proceedings of the VLDB Endowment*, 10(9):925–936, May 2017. ISSN 2150-8097. doi: 10.14778/3099622.3099624. URL https://dl.acm.org/doi/10.14778/3099622.3099624.

[43] Batya Kenig and Avigdor Gal. MFIBlocks: An effective blocking algorithm for entity resolution. *Information Systems*, 38(6):908–926, September 2013. ISSN 03064379. doi: 10.1016/j.is.2012.11.008. URL https://linkinghub.elsevier.com/retrieve/pii/S0306437912001500.

[44] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. Cost-effective crowdsourced entity resolution: A partial-order approach. In *Proceedings of the 2016 International Conference on Management of Data*, pages 969–984, San Francisco California USA, June 2016. ACM. ISBN 978-1-4503-3531-7. doi: 10.1145/2882903.2915252. URL https://dl.acm.org/doi/10.1145/2882903.2915252.

[45] Lotfi Lakhal and Gerd Stumme. Efficient mining of association rules based on formal concept analysis. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors, *Formal concept analysis*, volume 3626, pages 180–195. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-27891-7 978-3-540-31881-1. doi: 10.1007/11528784_10. URL http://link.springer.com/10.1007/11528784_10. Series Title: Lecture Notes in Computer Science.

[46] Kartick Chandra Mondal, Nicolas Pasquier, Anirban Mukhopadhyay, Ujjwal Maulik, and Sanghamitra Bandhopadyay. A new approach for association rule mining and bi-clustering using formal concept analysis. In *Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 86–101. Springer, 2012.

[47] Sergei O Kuznetsov and Jonas Poelmans. Knowledge representation and processing with formal concept analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3(3):200–215, 2013.

[48] Yong Liu and Xueqing Li. Application of formal concept analysis in association rule mining. In *Proceedings of the 2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 203–207. IEEE, 2017.

[49] YY Yao. Concept lattices in rough set theory. In *Proceedings of the 2004 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS'04)*, volume 2, pages 796–801. IEEE, 2004.

[50] Yiyu Yao. A comparative study of formal concept analysis and rough set theory in data analysis. In Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard

Weikum, Shusaku Tsumoto, Roman Słowiński, Jan Komorowski, and Jerzy W. Grzymała-Busse, editors, *Rough sets and current trends in computing*, volume 3066, pages 59–68. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-22117-3 978-3-540-25929-9. doi: 10.1007/978-3-540-25929-9_6. URL http://link.springer.com/10.1007/978-3-540-25929-9_6. Series Title: Lecture Notes in Computer Science.

[51] Hongliang Lai and Dexue Zhang. Concept lattices of fuzzy contexts: Formal concept analysis vs. rough set theory. *International Journal of Approximate Reasoning*, 50 (5):695–707, May 2009. ISSN 0888613X. doi: 10.1016/j.ijar.2008.12.002. URL https://linkinghub.elsevier.com/retrieve/pii/S0888613X08001837.

[52] Wei-Zhi Wu, Yee Leung, and Ju-Sheng Mi. Granular computing and knowledge reduction in formal contexts. *IEEE Transactions on Knowledge and Data Engineering*, 21(10):1461–1474, October 2009. ISSN 1041-4347. doi: 10.1109/TKDE.2008.223. URL http://ieeexplore.ieee.org/document/4663069/.

[53] KA Vidhya and TV Geetha. Resolving entity on a large scale: Determining linked entities and grouping similar attributes represented in assorted terminologies. *Distributed and Parallel Databases*, 35(3):303–332, 2017.

[54] KA Vidhya and TV Geetha. Rough set theory for document clustering: A review. *Journal of Intelligent & Fuzzy Systems*, 32(3):2165–2185, 2017.

[55] KA Vidhya and TV Geetha. Entity resolution framework using rough set blocking for heterogeneous web of data. *Journal of Intelligent & Fuzzy Systems*, 34(1):659–675, 2018.

[56] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 2019.

[57] Xiaofeng Zhu, Xuelong Li, Shichao Zhang, Zongben Xu, Litao Yu, and Can Wang. Graph PCA hashing for similarity search. *IEEE Transactions on Multimedia*, 19 (9):2033–2044, 2017.

[58] Roberto J Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on the World Wide Web*, pages 131–140, 2007.

[59] Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. String similarity search and join: A survey. *Frontiers of Computer Science*, 10(3):399–417, 2016.

[60] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment*, 9(9):636–647, May 2016. ISSN 21508097. doi: 10.14778/2947618.2947620. URL http://dl.acm.org/citation.cfm?doid=2947618.2947620.

[61] Pengfei Xu and Jiaheng Lu. Efficient taxonomic similarity joins with adaptive overlap constraint. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM'18)*, pages 1563–1566, Torino, Italy, 2018. ACM Press. ISBN 978-1-4503-6014-2. doi: 10.1145/3269206.3269236. URL http://dl.acm.org/citation.cfm?doid=3269206.3269236.

[62] Tobias Christiani, Rasmus Pagh, and Johan Sivertsen. Scalable and robust set similarity join. *arXiv:1707.06814 [cs]*, March 2018. URL http://arxiv.org/abs/1707.06814. arXiv: 1707.06814.

[63] Yasin N Silva, Walid G Aref, and Mohamed H Ali. The similarity join database operator. In *Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 892–903. IEEE, 2010.

[64] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997 (SEQUENCES'97)*, pages 21–29. IEEE, 1997.

[65] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

[66] Junfeng He, Wei Liu, and Shih-Fu Chang. Scalable similarity search with optimized kernel hashing. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)*, page 1129, Washington, DC, USA, 2010. ACM Press. ISBN 978-1-4503-0055-1. doi: 10.1145/1835804.1835946. URL http://dl.acm.org/citation.cfm?doid=1835804.1835946.

[67] Pankaj Malhotra, Puneet Agarwal, and Gautam M Shroff. Graph-parallel entity resolution using LSH & IMM. In *Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference*, pages 41–49, 2014.

[68] Huizhi Liang, Yanzhe Wang, Peter Christen, and Ross Gayler. Noise-tolerant approximate blocking for dynamic real-time entity resolution. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 449–460. Springer, 2014.

[69] Qing Wang, Mingyuan Cui, and Huizhi Liang. Semantic-aware blocking for entity resolution. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):166–180, 2015.

[70] Luca Gagliardelli, Giovanni Simonini, Domenico Beneventano, and Sonia Bergamaschi. Sparker: Scaling entity resolution in spark. In *Proceedings of the 22nd International Conference on Extending Database Technology (EDBT'19)*. PRT, 2019.

[71] Vasilis Verroios and Hector Garcia-Molina. Top-k entity resolution with adaptive locality-sensitive hashing. In *Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1718–1721. IEEE, 2019.

[72] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[73] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[74] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[75] Jeffrey K Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

[76] Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 136–150. IEEE, 2015.

[77] Bin Fan, Qingqun Kong, Baoqian Zhang, Hongmin Liu, Chunhong Pan, and Jiwen Lu. Efficient nearest neighbor search in high dimensional hamming space. *Pattern Recognition*, 99:107082, 2020.

[78] Mani Malek Esmaeili, Rabab Kreidieh Ward, and Mehrdad Fatourechi. A fast approximate nearest neighbor search algorithm in the hamming space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2481–2488, 2012.

[79] Sepehr Eghbali, Hassan Ashtiani, and Ladan Tahvildari. Online nearest neighbor search in binary space. In *Proceedings of the 2017 IEEE International Conference on Data Mining (ICDM'17)*, pages 853–858. IEEE, 2017.

[80] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(6):1107–1119, 2013.

[81] Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence (AAAI/IAAI'00)*, 2000.

[82] Pierre Gançarski, Bruno Crémilleux, Germain Forestier, Thomas Lampert, et al. Constrained clustering: Current and new trends. In *A guided tour of artificial intelligence research*, pages 447–484. Springer, 2020.

[83] Kiri L Wagstaff, Sugato Basu, and Ian Davidson. When is constrained clustering beneficial, and why? *Ionosphere*, 58(60.1):62–63, 2006.

[84] Sugato Basu and Ian Davidson. Clustering with constraints. URL: http://www.cs.albany.edu/~davidson/Publications/KDDSlides.pdf, 2006.

[85] Eric P Xing, Andrew Y Ng, Michael I Jordan, and Stuart Russell. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, page 12, 2002.

[86] Sugato Basu, Mikhail Bilenko, and Raymond J Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 59–68, 2004.

[87] Steven Euijong Whang and Hector Garcia-Molina. Entity resolution with evolving rules. *Proceedings of the VLDB Endowment*, 3(1-2):1326–1337, September 2010. ISSN 2150-8097. doi: 10.14778/1920841.1921004. URL https://dl.acm.org/doi/10.14778/1920841.1921004.

[88] Steven Euijong Whang and Hector Garcia-Molina. Incremental entity resolution on rules and data. *The VLDB Journal*, 23(1):77–102, February 2014. ISSN 1066-8888, 0949-877X. doi: 10.1007/s00778-013-0315-0. URL http://link.springer.com/10.1007/s00778-013-0315-0.

[89] Hugo Steinhaus. Sur la division des corps matériels en parties. *Bull. Acad. Polon. Sci*, 1(804):801, 1956.

[90] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[91] Robert R Sokal. A statistical method for evaluating systematic relationships. *Univ. Kansas, Sci. Bull.*, 38:1409–1438, 1958.

[92] Eric D Ragan, Hye-Chung Kum, Gurudev Ilangovan, and Han Wang. Balancing privacy and information disclosure in interactive record linkage with visual masking. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[93] Alexander Tekles and Lutz Bornmann. Author name disambiguation of bibliometric data: A comparison of several unsupervised approaches. *Quantitative Science Studies*, 1(4):1510–1528, 2020.

[94] Michael Levin, Stefan Krawczyk, Steven Bethard, and Dan Jurafsky. Citation-based bootstrapping for large-scale author disambiguation. *Journal of the American Society for Information Science and Technology*, 63(5):1030–1047, 2012. ISSN 1532-2890. doi: 10.1002/asi.22621. URL https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.22621. _eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/asi.22621.

[95] Edward H Simpson. The interpretation of interaction in contingency tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 13(2):238–241, 1951.

[96] Bikash Gyawali, Lucas Anastasiou, and Petr Knoth. Deduplication of scholarly documents using locality sensitive hashing and word embeddings. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 901–910, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.113.

[97] Paul Donner, Christine Rimmert, and Nees Jan van Eck. Comparing institutional-level bibliometric research performance indicator values based on different affiliation disambiguation systems. *Quantitative Science Studies*, 1(1):150–170, February 2020. ISSN 2641-3337. doi: 10.1162/qss_a_00013. URL https://www.mitpressjournals.org/doi/abs/10.1162/qss_a_00013.

[98] Sanguthevar Rajasekaran. Efficient parallel hierarchical clustering algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):497–502, 2005.