Blocking Techniques for efficient Entity Resolution over large, highly heterogeneous Information Spaces

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

DOKTOR DER NATURWISSENSCHAFTEN
Dr. rer. nat.

genehmigte Dissertation
von

Dipl. Ing. Georgios Papadakis

geboren am 2. Juli 1984 in Heraklion, Kreta, Griechenland

2013

Referent: Prof. Dr. Wolfgang Nejdl
Korreferent: Prof. Dr. Themis Palpanas
Tag der Promotion: 7 Juni 2013

Acknowledgments

This dissertation consolidates my research on Entity Resolution in the time period between August 2008 and August 2012. In these four years, many people have influenced and contributed to my work, both directly and indirectly. I am deeply grateful to all of them, especially to those mentioned by name in the following.

First and foremost, I would like to thank my advisor, Prof. Wolfgang Nejdl, for giving me the opportunity to join the lively and inspiring team of the *L3S* Research Center. I am particularly grateful for the active support he continued to offer me, even after I took the big decision to move back to Greece, in December 2010. Indisputably, this work would have been impossible without his help.

I am also sincerely indebted to my mentor at *L3S*, Claudia Niederée, for she was the one that encouraged me to pursue this specific topic in the context of my PhD studies. She has been apt to provide valuable feedback and guidance throughout these four years, particularly during my first steps.

I would also like to express my gratitude to Prof. Themis Palpanas for the close collaboration and the insightful discussions we had during the last 3 years. He has always provided me with interesting comments and ideas for improving and extending my work.

Quite crucial was also the support of *L3S*' Greek team, consisting of Ekaterini Ioannou, Odysseas Papapetrou and Dimitris Skoutas. Their help and professional advice were invaluable while working at *L3S*, whereas their company during free time made life in Hanover much funnier and easier. The same applies to most of my *L3S* colleagues, especially to Ricardo Kawase, Marco Fisichella and Gian Luca Volpato. Working and hanging out with *L3S* people has been a great experience that is etched in my memory and personality.

Special thanks goes to all my co-authors, as well. It has always been a pleasure to discuss and work with them on challenging research issues. I would also like to thank my office-mates at *L3S*, especially Sukriti Ramesh and Nina Tahmasebi, for

the quiet, pleasant and inspiring environment we had at the 15th floor.

Last but not least, I would like to express my deepest respect and appreciation for my people in Greece. They fully supported me after my return to Athens, even though they all disagreed with my choice. This pertains particularly to my parents, Urania and Antony, who always provided me with ample financial and moral help, and to my sister, Katerina, who has taken care of me from the beginning of my undergraduate studies. As a member of this family, I already regard Christina-Maria Kastorini, who has stood firmly by me the last two years. I will be eternally grateful to all of you.

Athens, December 2012

Zusammenfassung

Aufgrund der guten Unterstützung für ihre Publikation und der weitgehend autonomen und verteilten Art ihrer Produktion ist in den letzten Jahre ein Boom im Bereich von Daten im Web (Web Data) zu beobachten: Unternehmen und Organisationen jeder Größe, einzelne Benutzer sowie automatische Extraktionswerkzeuge tragen ein schnell anwachsendes Volumen von diversen, aber auch sehr heterogenen und verrauschten Informationen bei. Entity Resolution (ER), d.h. die Erkennung von Duplikaten, hilft dabei, die Entropie zu verringern und die fragmentierten Daten im Web effektiver zu nutzen, indem sie Repräsentationen von Entitäten identifiziert, die sich auf die gleichen Objekte der realen Welt beziehen. Um ER für große und sehr große Datenmengen zu skalieren, kommt typischerweise der Ansatz des Blockings zum Einsatz, d.h. die Aufteilung der zu untersuchenden Datenmenge in Blöcke von Duplikatkandidaten. Blocking-Methoden beruhen jedoch auf der Nutzung von Schemainformation, wodurch sie in der betrachteten hoch-heterogenen Situation der Web-Daten nicht einsetzbar sind. Somit werden neue Ansätze benötigt.

Diese Dissertation führt eine innovative Methode des Blocking ein, welche inhärent auf die sehr großen, sehr heterogenen und verrauschten Daten im Web eingestellt ist. Sie geht damit über existierende Blocking-Methoden hinaus. Sie setzt sich aus drei wichtigen sich ergänzenden Bausteinen zusammen: Zunächst kommt für den Aufbau der Blöcke eine schema-agnostische Funktion zum Einsatz, welche jede Entität mit mehreren Blöcken assoziiert. Auch im anspruchsvollen Kontext der Web-Daten minimiert dies die Wahrscheinlichkeit Duplikate zu verfehlen. Der zweite Baustein, Meta-Blocking, restrukturiert die Blöcke aufgrund einer Analyse der überlappungsmuster von Entitäten. Dadurch lassen sich die Berechnungskosten senken ohne die Effektivität nennenswert zu reduzieren. Der dritte Baustein, innovative Methoden zur Blockabarbeitung, trägt zur weiteren Steigerung der Effizienz bei, indem die Anzahl überflüssiger Vergleiche systematisch reduziert wird. Die Leistungsfähigkeit der einzelnen Bausteine wird in extensiven experimentellen Analysen mit drei umfangreichen, realen Datenkollektionen untersucht. Die Ergebnisse der Experimente bestätigen die ausgezeichnete Balance zwischen Effektivität und Effizienz, welche durch unseren Ansatz erreicht wird.

Schlagworte: Datenintegration, Duplikaterkennung, Blocking-Methoden.

Abstract

Web Data have boomed during the last decade, due to their largely distributed way of production: corporations of any size, individual users as well as automatic extraction tools have contributed a constantly increasing volume of diverse, but also very heterogeneous and noisy information. Entity Resolution (ER) helps to reduce the entropy, leveraging the value of the fragmented Web Data by identifying those pieces of information that refer to the same real-world objects. To scale ER to the large and very large data sets, such as Web Data, data blocking techniques are typically employed. However, most of them rely on schema information and, thus, are inapplicable to the highly heterogeneous settings of Web Data — a situation that calls for novel approaches.

This dissertation goes beyond existing blocking techniques, by introducing a novel methodology that is inherently crafted for the voluminous, highly heterogeneous, noisy collections of Web Data. Its goal is to place every pair of matching entities in at least one common block, while minimizing the number of unnecessary comparisons. At its core lie three independent, but complementary phases: first, block building techniques aim at clustering entities into blocks through a redundancy-bearing, schema-agnostic functionality, which associates every entity with multiple blocks without considering any schema information. In this way, the likelihood of missed matches is minimized even in the challenging context of Web Data. Second, meta-blocking relies on patterns of co-occurrence among entities sharing multiple blocks in order to restructure the resulting block collection into a new one that identifies practically the same portion of duplicates, while reducing significantly the computational cost. Third, block processing techniques further enhance efficiency, by discarding comparisons or even entire blocks that involve non-matching entities. We analytically examine the performance of every phase through a thorough experimental study that involves three large-scale, real-world data sets. Its outcomes demonstrate that our methodology achieves an excellent balance between effectiveness and efficiency.

Keywords: Data Integration, Entity Resolution, Blocking Techniques.

Contents

List of Tables

List of Algorithms

List of Figures

Chapter 1

Introduction

The amount of global, digital information has grown by a factor of 9 between 2006 and 2011, reaching the unprecedented levels of 1.8 Zettabytes[1] by the end of 2011[2]. This information deluge includes not only unstructured data in the form of raw, textual content (e.g., Web pages), but also semi-structured and structured information that follow arbitrary schemas. Numerous factors account for this phenomenon: the distributed production of information in businesses and organizations, the increased ability and interest for automatic information extraction from raw data as well as the prolific activity of individual users all over the world, who constantly and voluntarily contribute new information through Web 2.0 tools. The combined effect of these factors gives rise to *highly heterogeneous information spaces* (HHIS), which encompass the (semi-)structured data that are manifested in Dataspaces [HFM06] and the Web of Data [BHBLBL09].

To leverage the investment in creating and collecting the massive volume of (semi-)structured data in HHIS, the *Linked Data vision* has been recently proposed [BHBLBL09]. It essentially advocates the combination of related resources in a unified way that enhances the usefulness and the usability of the interlinked data. A core part of this large-scale integration process is *Entity Resolution* (ER), i.e., the process of automatically identifying sets of entity profiles that pertain to the same real-world object. ER constitutes an inherently quadratic task: in principle, every entity (of the one collection) has to be compared with all others (of the other collection). As a result, ER is typically made scalable to large volumes of data through *approximate techniques*. These techniques significantly enhance *efficiency* (i.e., the

[1]A Zettabyte is equal to 10^{21} bytes in the SI metric system.
[2]http://www.emc.com/collateral/about/news/idc-emc-digital-universe-2011-infographic.pdf

required number of pairwise comparisons), by sacrificing some *effectiveness* (i.e., the portion of detected duplicates).

The most prominent among these approximation techniques is *data blocking*, which aims at clustering similar entities into blocks so that it suffices to perform comparisons only among entities within the same block. There is a plethora of techniques in this field, but their vast majority is crafted for *homogeneous information spaces* (HOIS), such as databases. These differ from HHIS in that they are described by a predetermined schema and all their data adhere to it. In this way, HOIS fulfill an essential prerequisite for the majority of existing blocking techniques, which rely on a-priori schema knowledge in order to select the most reliable and distinctive attributes for producing blocks of high effectiveness (i.e., a large portion of the matching entities shares at least one block) [Chr12b, Chr12a, NMMMBLP07]. These methods are practically inapplicable to HHIS, due to the absence of reliable, compact and binding schema information that are suitable for blocking.

The blocking techniques presented in this dissertation go beyond the existing ones, as they are inherently crafted for HHIS and involve a functionality that is decoupled from schema information. They are also highly efficient, enabling ER to scale up to entity collections with tens of millions of profiles.

In Section 1.1, we further explain the motivation behind this work, while in Section 1.2, we elaborate on the challenges imposed by HHIS. Section 1.3 provides an overview of the techniques introduced in this dissertation, Section 1.4 summarizes its contributions, and Section 1.5 presents its structure.

1.1 Motivation

To illustrate the difference between blocking for HOIS and for HHIS, consider the simple HOIS entity collection depicted in Figure 1.1 (a). Apparently, profile p_1 matches with p_3 and p_2 with p_4. Despite the slightly different attribute values among duplicate profiles, all entities share the same attribute names (i.e., schema). This allows for easily identifying those attribute name(s) that can produce blocks of high quality. In fact, the goal of blocking is to place every pair of matching entities in at least one common block (high effectiveness), while restricting the total number of comparisons at low levels (high efficiency). In this context, one of the possible solutions for the given entity collection is depicted in Figure 1.1 (b); blocks are extracted from the values of the attribute name “zip code”, with each

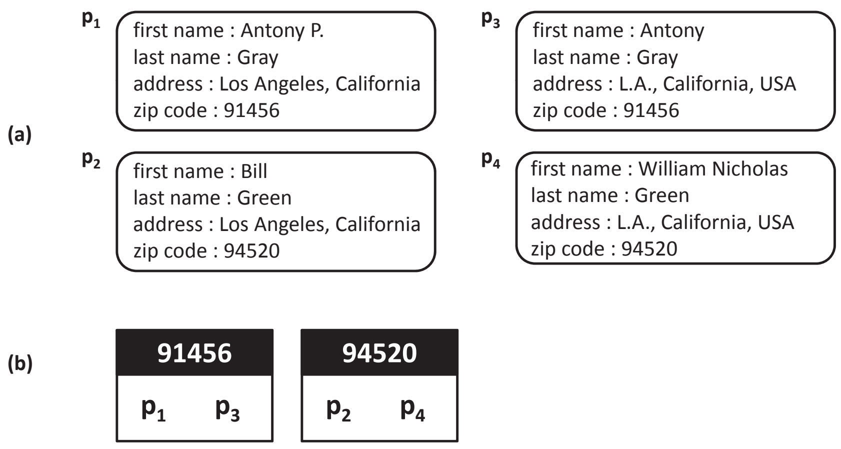


Figure 1.1: (a) Entity profiles stemming from a homogeneous information space (HOIS), and (b) Blocks created for them by traditional blocking techniques.

block corresponding to a distinct value. Both of the resulting blocks individually contain just one pair of matching entity profiles. Thus, assuming that we have an accurate entity matching method, all duplicate entities are detected with just 2 pairwise comparisons. The same entity matching approach would require 4 comparisons, when coupled with the naive (i.e., exhaustive) ER solution.

Consider now the HHIS entity collection that is depicted in Figure 1.2. Again, profile p_1 is matching with p_3 and p_2 with p_4. In this case, however, there are extreme levels of heterogeneity in the schema and the values of the entity profiles. In fact, the semantically equivalent attribute names appear in so many syntactically different forms (e.g., “Profession”, “work” and “job) that none of them is associated with more than one entity profile. The same applies to the attribute values, as well (e.g., “car dealer”, “car seller” and “auto seller”). Note also that the loose schema binding of HHIS abounds in tag-style values (e.g., “car seller” in p_4) and attribute names of different granularity; for instance, “zip code” corresponds to a subset of the practically equivalent attributes “Address”, “location” and “current location”. A further obstacle to schema-based blocking stems from the high levels of noise, which — among others — comes in the form of spelling mistakes (e.g., “Calefornia” in p_1).

In summary, traditional blocking approaches are inapplicable to HHIS, due to their strict requirement for a homogeneous binding schema with attributes of a-priori known characteristics. An alternative solution would be to transform a

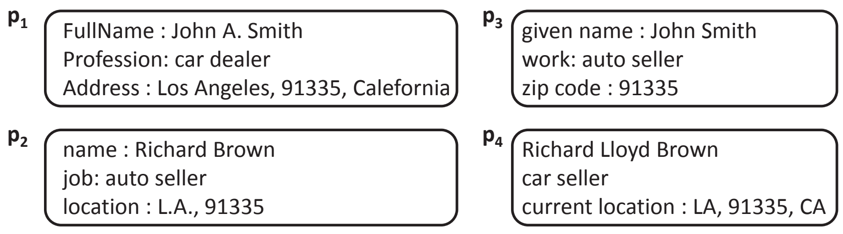


Figure 1.2: Entity profiles stemming from a highly heterogeneous information space (HHIS).

HHIS into a HOIS through a schema matching algorithm, and then apply traditional blocking techniques on the resulting canonical schema. However, the relevant techniques do not scale to the extreme levels of schema heterogeneity that HHIS involve (e.g., thousands of distinct attribute names), as the number of mappings they produce grows extremely fast with respect to the number of input attributes names [PINF11, NMMMBLP07, RB01]. We further elaborate on the intricacies of HHIS in the following section and explain how our approaches overcome them in Section 1.3.

1.2 Challenges

Any blocking technique that aims at achieving a good balance between efficiency and effectiveness over HHIS has to consider the following intrinsic characteristics:

- **Challenge 1 — Loose schema binding.** HHIS comprise structured and semi-structured data that are loosely bound to a rich diversity of schemata, ranging from locally-defined attribute names to pure tag-style annotations. The unprecedented level of heterogeneity pertains not only to the schemata describing the same entity types, but also to the separate profiles describing the same entity. For instance, Google Base[3] encompasses 100,000 distinct schemata corresponding to 10,000 entity types [MCD+07], whereas most bibliographic databases — even small ones like Cora[4] — abound in citations of varying format and quality that actually refer to the same paper. In the previous section, we explained that the major consequence of these settings with respect to blocking is the lack of schema information that could indicate the most suitable at-

[3] `http://www.google.com/base`
[4] `http://people.cs.umass.edu/ mccallum/data.html`

tribute name(s) for clustering matching entities into blocks. Even the advanced state-of-the-art schema matching approaches are inadequate for handling such extreme levels of heterogeneity [PINF11, NMMMBLP07, RB01]. Therefore, the loose schema binding calls for blocking approaches that are less dependent or even independent of schema information.

- **Challenge 2 — High levels of noise.** Web data are published through a free, unsupervised process that cannot filter information of low quality. As a result, they abound in noise, which ranges from spelling mistakes to missing information and inconsistent values. The deficient and/or false information in HHIS hamper the identification of matching entities and, thus, the creation of blocks. Blocking techniques usually transform every entity profile into a signature that is extracted from one or more selected attributes and subsequently place entities with identical signatures in the same block. In the example of Figure 1.1 (a), blocks were formed by representing every entity through a signature that merely consists of its value for the attribute "zip code". However, noise in signatures averts matching entities from sharing at least one block. Continuing our example, imagine there was an error in p_1's value for "zip code" (e.g., "9156" instead of "91456"); inevitably, p_1 would have no block in common with p_3. A possible solution to this issue would be to represent every entity with multiple signatures that are derived from different attributes. In the absence of schema information, though, this approach offers no viable solution. Therefore, blocking techniques for HHIS have to be inherently robust against any form of noise in entity profiles.

- **Challenge 3 — Huge and evolving volume.** Users contributing to HHIS are rather prolific, conveying an exponential growth in the content of Web 2.0 platforms, such as Wikipedia [AMC07]. Freebase alone contains more than 22 millions entities together with over 350 millions facts in about 100 domains [DZN12]. HHIS are also enriched by applications that automatically extract information from a variety of sources. In total, the Web of Data has increased its content from 4.7 billion triples in May, 2009 [BHBLBL09] to more than 30 billion triples by the end of 2012 [BdMNW12]. In the context of this unprecedented volume of data, the existing blocking techniques produce blocks of low efficiency. To achieve high efficiency, their signatures have to be quite distinctive, so that the average block size remains low. However, the more distinctive their signatures are, the higher is the likelihood of missed matches, due to the intrinsic noise in HHIS. Thus, distinctive signatures can only achieve

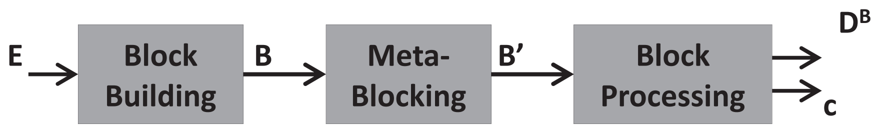


Figure 1.3: Our three-layered approach to blocking-based ER over HHIS.

high effectiveness through *redundancy*, i.e., the practice of placing every entity into multiple blocks, which also leads to low efficiency. In any case, the resulting block collections involve an excessively high computational cost, thus calling for novel techniques that process each block by identifying and purging the unnecessary comparisons. These methods are able to enhance the overall ER efficiency without affecting its effectiveness.

In the following, we introduce novel approaches to blocking-based ER over HHIS that are inherently capable of overcoming the above three challenges.

1.3 Summary of the Approach

Our approach to blocking goes beyond those presented in the literature in three ways:

- it is inherently crafted for dealing with the aforementioned challenges of HHIS,
- it breaks the blocking-based ER process over HHIS into three distinct steps that decouple effectiveness from efficiency, maximizing them independently, and
- it introduces a framework that facilitates practitioners in their effort to combine complementary blocking methods into highly performing ER solutions that can be easily tailored to the particular settings and requirements of each application.

In more detail, our framework consists of three layers, which are depicted in Figure 1.3. Each layer is responsible for a specific step of the blocking-based ER process and receives as input the output of the previous one. Its goal is to produce an output that improves the effectiveness or the efficiency (or both aspects) of the input.

The aim of the first layer, called Block Building, is to overcome Challenges 1 and 2 so as to cluster the input entities $\mathcal{E}$ into a block collection $\mathcal{B}$ that exhibits high levels of effectiveness at a reasonable cost in efficiency. The extreme heterogeneity (i.e., Challenge 1) is tackled through an *attribute-agnostic functionality* that completely disregards any schema information; blocks are exclusively built on the

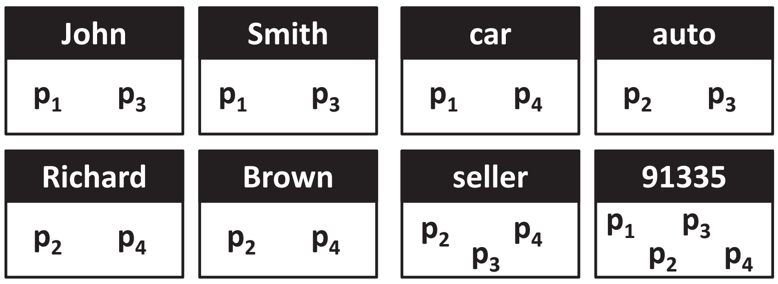


Figure 1.4: Blocks created for the entity collection of Figure 1.2 by a simple attribute-agnostic approach.

basis of attribute values. The high levels of noise (i.e., Challenge 2) are addressed through *redundancy*, which increases the likelihood that duplicate entities have at least one block in common. To illustrate these two characteristics, consider the blocks of Figure 1.4, which cluster the entities from Figure 1.2. We can notice that there is a distinct block for each token appearing in the attribute values of at least two entities and that it encompasses all entities containing the corresponding token in their profile. Thus, no schema information is used in the creation of blocks, and every entity is placed in multiple blocks. This example actually illustrates the simplest of our block building techniques, called *Token Blocking* (cf. Section 4.1).

The goal of the second layer, called Meta-Blocking, is to restructure the output of the first layer, $\mathcal{B}$, into a new block collection $\mathcal{B}'$ that maximizes efficiency, while retaining the original, high levels of effectiveness. Meta-Blocking actually aims at tackling the combined effect of Challenge 3 and the redundancy introduced by the underlying attribute-agnostic block building method, which together result in an excessively high number of pairwise comparisons. As an example, consider the block collection of Figure 1.4; in total, it contains 13 comparisons, although the naive ER approach would resolve the entities of Figure 1.2 with just 4 comparisons. The number of executed comparisons can be significantly restricted by discarding the repeated ones as well as those involving entities that are highly unlikely to be matching. Valuable evidence for this procedure is encapsulated in the *block assignments*[5] of $\mathcal{B}$; usually, the more blocks two entities have in common, the more likely they are to be matching. In this context, a new block collection $\mathcal{B}'$ can be derived from $\mathcal{B}$ by retaining those pairs of entities that co-occur frequently in the input blocks of $\mathcal{B}$. Continuing our example of Figure 1.4, if we retain the two pairs

[5]A block assignment is the association between a block and an entity.

of entities with the highest block overlap, we end up with two new blocks, $b_1 = \{p_1, p_3\}$ and $b_2 = \{p_2, p_4\}$, that need just 2 comparisons to identify all duplicates.

The third layer, called Block Processing, includes a variety of techniques that also aim at overcoming the combined effect of Challenge 3 and redundancy in order to maximize efficiency. Unlike meta-blocking techniques, they do so by examining individual blocks and comparisons so as to decide whether they will be processed and in which order. For instance, such a technique would discard the block "91335" of Figure 1.4 on the grounds that it is oversized (i.e., it contains the entire entity collection of Figure 1.2). Another example is a technique that eliminates all repeated comparisons, allowing the similarity of each pair of entities to be assessed just once, in the first block they share (cf. 6.3.1). Depending on the type of comparisons they target and the granularity of their functionality, block processing techniques can be conflicting or complementary. The former serve exactly the same goal and, thus, it suffices to apply one of them to the ER problem at hand; imagine, for instance, two methods that eliminate all repeated comparisons. In contrast, complementary block processing techniques target different types of comparisons and can be combined in an *ER workflow* of higher efficiency according to specific guidelines. The goal is actually to detect as many pairs of matching entities as possible, while restricting the computational cost c to the minimum possible number of executed comparisons.

On the whole, this thesis proposes a layered framework for blocking-based ER over HHIS that consists of three orthogonal, but complimentary tiers. Every layer comprises multiple techniques that allow for numerous combinations, called ER workflows. Their excellent performance in practice is verified through a thorough experimental study that involves three large-scale, real-world data sets. We have freely published their implementation (in Java) through Sourceforge.net[6] along with directions for obtaining our benchmark data.

1.4 Contributions

The novelties of our research work are organized into the following four areas:

[6]`https://sourceforge.net/projects/erframework/`

I. Block Building

The vast majority of relevant works in the literature focuses on schema-based block building techniques that are crafted for HOIS (cf. Section 2.2 for more details). In contrast, this dissertation presents novel block building techniques that rely on an attribute-agnostic, redundancy-bearing functionality in order to create blocks of high effectiveness in the context of HHIS. They are grouped in three families. The first one exclusively contains the basic technique of Token Blocking, which was illustrated in Figure 1.4. It achieves high robustness and effectiveness, at the cost of low efficiency (i.e., too many comparisons), due to the extreme levels of redundancy it employs. The other two families build upon Token Blocking with the aim of achieving equally high effectiveness at a significantly higher efficiency (i.e., lower redundancy). First, Agnostic Clustering techniques group together attributes with similar values and apply Token Blocking inside each cluster, independently of the others. The resulting blocks involve fewer comparisons, while missing a negligible number of duplicate entities. Second, the URI Semantics techniques include a series of atomic blocking schemes that are crafted for RDF data, exploiting the evidence contained in entity identifiers. They yield small blocks of high efficiency, but of limited effectiveness. Given that each atomic scheme considers a different aspect of entity profiles, their effectiveness can be substantially enhanced by combining them into composite blocking methods.

These block building techniques were originally introduced in the following of my publications:

[PINF11] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Peter Fankhauser. *Efficient Entity Resolution for Large Heterogeneous Information Spaces*. In Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM), February 2011, Hong Kong, China. Also presented at the 10th Hellenic Data Management Symposium (HDMS), June 2011, Athens, Greece.

[PIP+ar] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl. *A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces*. In IEEE Transactions on Knowledge and Data Engineering (TKDE) — to appear.

[PIN+12] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, Wolfgang Nejdl. *Beyond 100 Million Entities: Large-scale Blocking-*

based Resolution for Heterogeneous Data. In Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM), February 2012, Seattle, Washington, USA. Also presented at the 11th Hellenic Data Management Symposium (HDMS), June 2012, Chania, Crete, Greece.

Also relevant to this specific contribution are the following publications:

[PDKF10] George Papadakis, Gianluca Demartini, Philipp Kaerger, Peter Fankhauser. *The Missing Links: Discovering Hidden Same-as Links among a Billion of Triples.* In Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS), November 2010, Paris, France.

[PGN+11] George Papadakis, George Giannakopoulos, Claudia Niederée, Themis Palpanas, Wolfgang Nejdl. *Detecting and exploiting stability in evolving heterogeneous information spaces.* In Proceedings of the 11th ACM/IEEE Joint Conference on Digital Libraries (JCDL), June 2011, Ottawa, Canada.

[Pap11] George Papadakis. *Efficient entity resolution methods for heterogeneous information spaces.* In Proceedings of the IEEE ICDE Ph.D. Workshop, April 2011, Hanover, Germany.

II. Meta-Blocking

To the best of our knowledge, no prior work tried to exploit the information encapsulated in a block collection with the aim of restructuring it into a new one of higher efficiency and equivalent effectiveness. This is exactly the goal of meta-blocking, another contribution of this dissertation. We actually formalize this process as a generic task that applies to any redundant block collection so that a plethora of solutions can be developed for it. We also tackle it through a family of techniques that rely on the *blocking graph*. This data structure models the block assignments of the input block collection in an abstract way that decouples the functionality of our techniques from the block building method that produced it: the nodes correspond to entities, and the edges to pairwise comparisons, with their weight indicating an estimated likelihood that the adjacent entities are matching — based on patterns in the block assignments. In fact, we coin five generic, attribute-agnostic weighting schemes that are based exclusively on the blocks the adjacent entities have in common. Efficiency can be enhanced simply by pruning the edges with a low weight. To this end, we present two categories of attribute-agnostic pruning algorithms along with four pruning criteria that can be organized into a two-dimensional taxonomy. In total, they compose four techniques for meta-blocking that are extensively evaluated through a thorough experimental study.

The problem of Meta-Blocking and the techniques for solving it were originally introduced in the following publication:

[PKPNar] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. *Meta-Blocking: Taking Entity Resolution to the Next Level.* In IEEE Transactions on Knowledge and Data Engineering (TKDE) — to appear.

III. Block Processing

Iterative Blocking [WMK+09] pioneered the development of methods that process a given block collection in a way that enhances its efficiency and/or its effectiveness. However, no other blocking method followed in this direction. This dissertation introduces a series of intelligent block processing techniques that enhance efficiency at a negligible and controllable impact on effectiveness. Their goal is actually to discard the repeated and unnecessary comparisons that are contained in a set of blocks. To facilitate their understanding and use, we organize them into a two-dimensional taxonomy that categorizes them according to the type of comparisons they target and the granularity of their functionality (i.e., whether they operate on the coarse level of blocks or on the finer level of individual comparisons). Some of these methods are complementary, targeting different types of comparisons, and when combined, they form ER workflows of higher performance than the individual methods comprising them. To facilitate their composition, we also introduce practical guidelines that are based on our two-dimensional taxonomy.

These block processing techniques were originally introduced in the following of my publications:

[PIP+ar] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl. *A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces*. In IEEE Transactions on Knowledge and Data Engineering (TKDE) — to appear.

[PIN+12] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, Wolfgang Nejdl. *Beyond 100 Million Entities: Large-scale Blocking-based Resolution for Heterogeneous Data.* In Proceedings of the 5th ACM

International Conference on Web Search and Data Mining (WSDM), February 2012, Seattle, Washington, USA. Also presented at the 11th Hellenic Data Management Symposium (HDMS), June 2012, Chania, Crete, Greece.

[PINF11] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Peter Fankhauser. *Efficient Entity Resolution for Large Heterogeneous Information Spaces.* In Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM), February 2011, Hong Kong, China. Also presented at the 10th Hellenic Data Management Symposium (HDMS), June 2011, Athens, Greece.

[PIN+11a] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, Wolfgang Nejdl. *Eliminating the redundancy in blocking-based entity resolution methods.* In Proceedings of the 11th ACM/IEEE Joint Conference on Digital Libraries (JCDL), June 2011, Ottawa, Canada.

[PIN+11b] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, Wolfgang Nejdl. *To Compare or Not to Compare: Making Entity Resolution more Efficient.* In Proceedings of the 3rd International Workshop on Semantic Web Information Management (SWIM), June 2011, Athens, Greece (collocated with SIGMOD 2011).

[Pap11] George Papadakis. *Efficient entity resolution methods for heterogeneous information spaces.* In Proceedings of the IEEE ICDE Ph.D. Workshop, April 2011, Hanover, Germany.

IV. Metric Space

Another topic that has been neglected in the literature is the development of theoretical tools that facilitate the functionality of blocking methods. In this dissertation, we introduce a general metric space that consists of two orthogonal measures that quantitatively capture the trade-off between blocking effectiveness and efficiency. Their values can be efficiently computed, without requiring any analytical block examination. Instead, they merely consider the external characteristics of each block (i.e., the number of entities and comparisons it involves). The resulting metric space applies to all three layers of our framework and can be used in a number of ways: to a-priori assess the actual performance of a blocking technique, to a-priori identify the best performing among a set of blocking methods (based on application-specific quality requirements), and to guide the internal functionality of a blocking method.

This metric space was originally introduced in the following publication:

[PIN+12] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, Wolfgang Nejdl. *Beyond 100 Million Entities: Large-scale Blocking-based Resolution for Heterogeneous Data.* In Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM), February 2012, Seattle, Washington, USA. Also presented at the 11th Hellenic Data Management Symposium (HDMS), June 2012, Chania, Crete, Greece.

1.5 Structure of the Dissertation

The rest of this dissertation is organized as follows: Chapter 2 discusses the most important blocking techniques in the literature; it puts more emphasis on the block building ones, categorizing them on the basis of a novel, two-dimensional taxonomy. Chapter 3 introduces our metric space along with the notions that are necessary for describing our methodology. Chapter 4 presents our approaches to effective block building over HHIS, while Chapter 5 analyzes the task of meta-blocking, explaining how the abstraction of the blocking graph allows for a wide diversity of highly efficient methods. In Chapter 6, we introduce our block processing techniques along with a two-dimensional taxonomy that clarifies their functionality and facilitates their combination into highly performing ER workflows. Chapter 7 investigates the actual performance of all our techniques through a detailed experimental study that comprises three large-scale, real-world data sets. Finally, Chapter 8 concludes the dissertation and provides directions for future work.

Chapter 2

Related Work

This section provides a comprehensive overview of the state-of-the-art techniques for blocking-based Entity Resolution. In Section 2.1, we start with the task of ER in general, classifying existing methods in three main categories. We then elaborate on blocking techniques for ER, distinguishing them into three main categories, as well: those focusing on the creation of blocks (Section 2.2), those dealing with their processing (Section 2.3) and the hybrid ones (Section 2.4), which simultaneously address both aspects of blocking.

2.1 Entity Resolution Techniques

Entity Resolution (ER) constitutes a traditional problem with numerous applications that has been investigated since the very beginning of computer science [NK62]. It is the task of identifying sets of entity profiles that pertain to the same real-world object and comes in two different forms [Chr12b, EIV07, KL10]:

- *Clean-Clean ER*, also known as *Record Linkage*, is the process of detecting pairs of matching entities among two heterogeneous, individually clean (i.e., duplicate-free), but overlapping collections of entities. As an example, consider the task of merging individual collections of consumer products that stem from different on-line stores, thus having proprietary identifiers and slightly varying descriptions.
- *Dirty ER*, also known as *Deduplication*, receives as input a single entity collection and aims at detecting the matching profiles that are contained in it. As an example, consider the task of citation matching in the context of a bibliographic

database, such as Google Scholar[1].

A plethora of methods for tackling ER have been proposed over the years. They are distinguished in three main categories [RDG11], according to the type of information they incorporate in their functionality:

- *Non-relational approaches* decide whether two entities are matching or not, judging solely from the attribute values of their profiles. In the context of HOIS, there is usually an one-to-one mapping between the schemata describing the given entity profiles, thus allowing the comparison of attribute values in a pairwise manner. As an example, consider the task of matching authors using exclusively their personal information (e.g., birth-date, address, affiliation). Some of the attribute comparison methods are specialized in categorical and numerical data (e.g., age and time), but the emphasis lies on methods for string-valued attributes [Chr12a]. The main bulk of non-relational ER approaches actually aims at developing string similarity metrics that are capable of handling noise and missing values. An analytical survey of the string distance metrics can be found in [CRF03], while the phonetic encoding functions[2] are analyzed in [Chr12a].
- *Relational approaches* enhance the non-relational ones by considering the values of associated entities, as well. An illustrative example is the task of matching authors not only on the basis of their personal information, but of the similarity of their co-authors, as well. The additional information offered by the associated entities usually yields higher accuracy, thus outperforming the non-relational approaches. To this category belong the approaches that are presented in [ACG02, KMC05].
- *Collective approaches* go beyond the relational ones by resolving multiple types of entities at the same time. Their fundamental assumption is that the match decisions for one type of entities facilitate the resolution of the other types. As an example, consider the task of resolving authors, publications and venues at the same time; detecting that two authors refer to the same real-world person reinforces the evidence for disambiguating the related publications and venues. Collective approaches are further distinguished into those propagating the latest matches to the rest of the data iteratively [BG07, DHM05] and to those taking match decisions in a truly collective manner [RDG11, MW04,

[1]`http://scholar.google.com`

[2]The phonetic encoding functions, such as Phonex [LR96], estimate the similarity between two string values based on their pronunciation.

BG06, HSM08, PMM$^+$02, SD06, WGM12]. The latter are of higher performance and, unlike the former, they do not suffer from the burden of bootstrapping (i.e., the problem of finding a good starting point for detecting the first duplicates).

For a more detailed overview of the state-of-the-art ER approaches, the interested reader can refer to surveys [Win06, DH05, EIV07, GD05] and textbooks [Chr12a].

Regardless of their internal functionality, ER methods typically suffer from low efficiency, due to their quadratic time complexity (basically, they compare every entity with all others). To scale them to large data collections, approximate techniques are usually employed. These yield large savings in efficiency by sacrificing effectiveness to some extent. The most prominent among these approaches is *data blocking* [EIV07, Chr12b]. Its goal is to group similar entity profiles into blocks so that duplicate entities can be exclusively identified through the pairwise comparisons inside every block.

We split *blocking-based ER* into two orthogonal, but complementary procedures: (i) the creation of blocks, which deals with the effective clustering of entities into blocks, and (ii) their processing, which deals with the efficient examination of the resulting set of blocks. The existing blocking techniques examine these two tasks either in conjunction or independently. We call the methods falling in the former category *hybrid blocking techniques* and distinguish those of the latter category into *block building techniques*, which address the creation of blocks, and *block processing techniques*, which focus on executing the minimum necessary portion of the pairwise comparisons. Each one of these categories is further analyzed in one of the following sections.

2.2 Block Building Techniques

The goal of these methods is to cluster the similar, input entities into blocks such that the matching ones are placed in at least one common block with a high probability. They receive as *input* one or two entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, and return as *output* a collection of blocks $\mathcal{B}$. Internally, they transform every input entity into a compact representation comprising one or more blocking keys (*BKs*) that summarize the values of selected attributes. In the more restricted case, every block corresponds to a particular BK and contains all entities having this key in their representation [FS69]. More general approaches are not restricted to key equality, but place entities with similar BKs into the same block [MNU00, JLM03].

There is a large body of work in this field, especially for HOIS. In fact, the majority of the existing block building methods rely on an a-priori known schema in order to select the appropriate attributes for deriving BKs of high quality. This decision requires knowledge about the semantics of every attribute as well as the quality and the distribution of their values [Chr12b, Chr12a]. Inevitably, this requirement turns most of the existing techniques inapplicable to the heterogeneous settings of HHIS we are considering in this work.

The most significant block building techniques for HOIS are the following:

- *Standard Blocking* defined the basic blocking functionality, as it was the first approach of this kind that was presented in the literature [FS69]. It represents every entity by a single BK and places two entities into the same block on the condition that they share exactly the same key (i.e., every block contains all entities represented by a particular BK). This functionality achieves high efficiency, but often leads to limited effectiveness, as it cannot deal with noisy and missing values in the attribute(s) selected for the BKs. This drawback can be partly ameliorated by applying the core functionality of Standard Blocking multiple times — using different attributes (BKs) in every iteration [WYP10].
- *Bigrams Blocking* [BCC03] and its generalization, *Q-grams Blocking*[GIJ$^+$01], are inherently robust to noisy values and BKs. In contrast to Standard Blocking, they associate every entity with multiple blocks, based on the bi-/q-grams[3] that are extracted from every BK. In this way, they incorporate redundancy and increase the likelihood that two matching entities have at least one block in common, even in the context of noisy data. However, the resulting number of comparisons is excessively high and does not scale well to large entity collections [Chr12b, Chr12a].
- The *Suffix Array* approach [AO05] also employs redundancy in order to tackle noise in BKs. It actually extracts suffixes of certain length from the BKs, by removing one or more characters from their beginning. Every suffix then forms a block that contains all entities having it in their representation. The main drawback of this approach is that it cannot handle errors at the end of BKs, which are rather frequent [PZ84]. To overcome this shortcoming, blocks corresponding to highly similar suffixes can be merged [dVKCC09, dVKCC11]. Using Bloom filters, the efficiency of this procedure can be significantly en-

[3]A q-gram of a textual value v is a sub-string of length q.

hanced [dVKCC11].

- *StringMap* [JLM03] relies on a mapping procedure that transforms the BKs of all records to a Euclidean space of a predefined dimensionality. The fundamental property of this mapping is that the new space preserves the original similarities between the BKs. With the help of suitable data structures, such as R-Trees, similar BKs can be efficiently grouped into clusters. A new block is then created for every such cluster, containing all entities that are associated with one of its BKs. The main drawback of this approach is that it suffers from the curse of dimensionality: the dimensionality of the Euclidean space has to be high in order to achieve good performance, but the auxiliary data structures become less efficient under these settings [Chr12a]. This issue can be partly ameliorated through a double embedding scheme, which further maps the Euclidean space to another one of lower dimensionality and employs a binary KD-tree for clustering [Adl09].
- *Canopy Clustering* [MNU00] is suitable for entities that are represented by multiple BKs. It clusters them into (overlapping) blocks by comparing pairwise their BKs with a computationally cheap string-similarity metric. Usually, TF-IDF or the Jaccard coefficient are selected for this task. The main drawback of this approach is that its performance depends on the distribution of BKs as well as on two similarity thresholds.
- *Semantic Indexing* [NMMMBLP07] completely disregards BKs and creates blocks by considering exclusively the relationships between entities. At its core lies a *collaborative graph*, where every node corresponds to an entity and every edge connects two associated entities. For instance, the collaborative graph for a bibliographic data collection can be formed by mapping every author to a node and adding edges between co-authors. In this context, blocks are created in the following way: for each node n, a new block is formed, containing all nodes connected with n through a path, whose length does not exceed a predefined limit. This approach was experimentally verified to outperform both Standard Blocking and Sorted Neighborhood (cf. Section 2.4).

Recent, comparative analyses of most of these blocking approaches can be found in [Chr12b, Chr12a]. They experimentally demonstrate that there are large differences in efficiency and effectiveness not only among different techniques, but also for different configurations of the same technique. This actually indicates that their functionality depends heavily on a variety of (sometimes sensitive) parame-

	Redundancy-free	**Redundancy-bearing**		
		Redundancy-negative	**Redundancy-neutral**	**Redundancy-positive**
Schema-based	Standard Blocking	Canopy Clustering	(Sorted Neighborhood)	1. Bi-/Q-grams Blocking 2. Suffix Array
Schema-agnostic	-	-	Semantic Indexing	*1. Token Blocking* *2. Agnostic Clustering* *3. URI Semantics*

Figure 2.1: The two-dimensional taxonomy of block building methods. Methods in italics are introduced in Chapter 4, while methods in parentheses are analyzed in Section 2.4.

ters. This study also validates that the most critical factor for blocking is the selection of the blocking keys. It is worth stressing, though, that not all these methods address the issue of defining effective BKs. In particular, StringMap and Canopy Clustering take as granted that every entity is associated with multiple BKs and exclusively aim at clustering similar entities into blocks.

2.2.1 Classification of Block Building Techniques[4]

Block building techniques are rarely applied in isolation. Instead, they are typically combined with one or more block processing techniques so that the number of executed comparisons is minimized. The most crucial factor for these combinations is the positioning of the block building technique with respect to redundancy and to schema information. The former determines how a blocking scheme interprets redundancy and its implications (i.e., does a high number of common blocks correspond to similar entities or to dissimilar ones?); the latter specifies whether schema knowledge is required for the creation of blocks. This factor is also crucial for selecting the most suitable block building technique for the application at hand; schema-based techniques, for instance, are inapplicable to settings involving HHIS. Therefore, to facilitate the use of block building techniques as well as their combination with block processing ones, we categorized them into a two-dimensional taxonomy that comprises the orthogonal criteria of redundancy and schema information. The resulting categorization is outlined in Figure 2.1.

With respect to redundancy, blocking methods are broadly distinguished into

[4]Originally introduced in [PKPNar].

redundancy-free, which produce non-overlapping blocks, and *redundancy-bearing*, which result in overlapping blocks. Redundancy-bearing techniques are further categorized according to their interpretation of redundancy.

For the *redundancy-positive* ones, the number of blocks shared by a pair of entities is proportional to their similarity and, thus, the likelihood that they are matching. To illustrate this approach, consider the block collection of Figure 1.4. Every block corresponds to a distinct token that has been extracted from at least one attribute value — regardless of the associated attribute name(s). Thus, the more blocks two entities share, the more similar their profiles are. As depicted in Figure 2.1, this category includes methods that associate every entity with multiple BKs, such as Q-grams Blocking and Suffix Array.

In contrast, *redundancy-negative* blocking methods regard a high number of shared blocks as a strong indication that the corresponding entities unlikely to be matching; for them, highly similar entities share just one block. Canopy Clustering offers an illustrative example: starting with a pool of candidate matches, which initially contains the entire entity collection(s), it iteratively selects a random seed s_i and creates a cluster (i.e., canopy) around it. This cluster contains those entities from the pool that are more similar with s_i than a predefined threshold t_1. However, the highly matching entities, whose similarity with s_i exceeds another threshold $t_2(>t_1)$, are completely removed from the pool and, thus, cannot be included in the canopy of another seed s_j. Given that s_i has also been removed from the pool, it is highly unlikely to share multiple blocks with the entities that are highly matching with it.

In the middle of these two extremes lie *redundancy-neutral* blocking methods: they involve the same number of common blocks across all pairs of entities (e.g., Sorted Neighborhood) or they are completely oblivious to redundancy (e.g., Semantic Indexing). StringMap constitutes a special case, as its relation to redundancy depends on the technique used for clustering entities in the multidimensional Euclidean space.

The awareness of schema knowledge distinguishes block building methods into *schema-based* and *schema-agnostic* ones. The former define the BKs on the basis of schema information, while the latter completely decouple their functionality from this kind of evidence. As depicted in Figure 2.1, all methods proposed in the literature are schema-based, except for Semantic Indexing, which involves a schema-agnostic functionality. Note, though, that schema-based methods have two major drawbacks:

- They are inapplicable to HHIS, since they cannot extract blocking keys of high quality in the absence of schema information.
- They usually require the fine-tuning of multiple parameters [dVKCC09]. For example, the Suffix Array involves the minimum suffix length and the maximum block size, while StringMap has to configure the dimensionality of the Euclidean space and the data structure that is used for clustering. As mentioned above, though, the most critical parameters are the definition of reliable BKs and the selection of appropriate similarity metrics; in Section 2.2.2, we present two approaches that automatically learn the optimal configuration for these two factors.

The schema-agnostic methods we introduce in this work exhibit high robustness in the context of HHIS, despite their parameter-free functionality.

2.2.2 Parameter Tuning for Block Building

As mentioned above, a common drawback of most block building techniques is that their performance depends on the fine-tuning of many application- and data-specific parameters [Chr12b, dVKCC09]. Two are the most critical issues that have to be resolved:

- the attribute name(s) that provide the most reliable BKs, and
- the similarity metric(s) that decide(s) whether two BKs are to be placed in the same block.

To overcome these issues, automatic tuning methods that are based on machine learning algorithms have been proposed in the literature. In more detail, Bilenko et al. [BKM06] and Michelson et al. [MK06] considered blocking schemes of the form {*similarity metric, attribute name*} and proposed supervised learning techniques that identify the combinations of individual schemes with the highest performance over the golden standard. They only differ in their approach; the former learns blocking schemes by solving an optimization problem equivalent to the red-blue set cover problem, while the latter learns them through the sequential set covering algorithm. Both approaches yield high performance, but cannot scale to the large schema space of HHIS.

2.3 Block Processing Techniques

These methods aim at examining a block collection in a way that enhances its effectiveness or its efficiency or both aspects. They receive as *input* a set of blocks $\mathcal{B}$ and return as *output* the detected pairs of duplicates $\mathcal{D}^{\mathcal{B}}$, usually along with the corresponding computational cost — in terms of the number of executed comparisons. Such methods, however, have been overlooked by researchers, as the only relevant approach in the literature is *Iterative Blocking* [WMK+09]. It processes a given block collection iteratively, so that the match decisions taken in the current block affect the examination of the others. In fact, every time a new pair of entities is detected as duplicates, it is replaced by the merged profile in all blocks containing either of the individual entities. These blocks are then examined, even if they have already been investigated. In this way, the matching accuracy increases (and so does the effectiveness) and many repeated comparisons are spared. Note, though, that in this case the input block collection is not static, but is continuously updated.

2.4 Hybrid Blocking Techniques

These methods deal with the creation and the processing of blocks in an integrated way. Therefore, they receive as *input* one or two entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, and return as *output* the detected pairs of duplicates $\mathcal{D}^{\mathcal{B}}$, usually accompanied by the corresponding computational cost. Their fundamental assumption is that the interplay of block building with block processing leads to a higher overall performance. Their drawback, though, is that they lack the necessary flexibility for composing ER workflows of higher performance in combination with specialized, complementary methods.

Only a handful of hybrid blocking methods have been proposed in the literature. The *Sorted Neighborhood* approach [HS95, HS98] defines BKs that are suitable for sorting, such that similar entities are placed in neighboring positions. In fact, entities are sorted in alphabetical order of their BKs, and blocks are created dynamically, through a sliding window of fixed size that gradually passes over the entire entity collection. In every iteration, the window advances by one entity, adding it to the block and comparing it with all other entities. This approach has been generalized to accommodate Standard Blocking, as the latter is equivalent to advancing the window by w entities, where w is the size of the sliding window [DN09].

Apparently, the performance of Sorted Neighborhood depends heavily on the size of the sliding window; the larger its value is, the more duplicates are identified (i.e., higher effectiveness), but the more comparisons are executed and the lower the overall efficiency gets. In contrast, small windows lead to a high number of missed matches (i.e., duplicate entities having no block in common) and to low effectiveness. To address this issue, *Adaptive Sorted Neighborhood* [YLKG07] adjusts dynamically the size of the window, based on the string similarity of the BKs. Still, Sorted Neighborhood is a schema-based blocking method that is crafted for HOIS and, thus, is not applicable to HHIS.

In another line of research, *HARRA* [KL10] introduces a hybrid method that dynamically creates blocks through an LSH-based procedure and processes them iteratively. In more detail, HARRA hashes entities so that the similar ones are placed into the same buckets, which now operate as blocks. Inside every bucket, all pairwise comparisons are executed and pairs of matching entities are merged into new profiles that will be re-hashed. This procedure is repeated until one of the three possible stopping criteria is satisfied. As stressed by the authors, this approach can be applied to both Clean-Clean and Dirty ER, with slight modifications.

Chapter 3

Problem Formulation

3.1 Entity Resolution

At the core of entity resolution (ER) lie collections of entity profiles that individually describe real-world entities. A set of entity profiles is called *entity collection* and is symbolized by $\mathcal{E}$. Assuming infinite sets of attribute names $\mathcal{N}$, values $\mathcal{V}$, and identifiers $\mathcal{I}$, it can be formally defined as follows:

Definition 3.1. *An **entity collection** $\mathcal{E}_i$ is a tuple $\langle N_i, V_i, I_i, P_i \rangle$, where $N_i \subseteq \mathcal{N}$ is the set of attribute names appearing in it, $V_i \subseteq (\mathcal{V} \cup \mathcal{I})$ is the set of values used in it, $I_i \subseteq \mathcal{I}$ is the set of global identifiers[1] contained in it, and $P_i \subseteq I_i \times \wp(N_i \times V_i)$ is the set of entity profiles it comprises.* ∎

An *entity profile* is a uniquely identified collection of information in the form of name-value pairs. It can be formally defined as follows:

Definition 3.2. *An **entity profile** p_{id} is a tuple $\langle id, A_{p_{id}} \rangle$, where $id \in \mathcal{I}$ is a unique identifier, and $A_{p_{id}}$ is a set of name-value pairs $\langle n, v \rangle$, with $n \in \mathcal{N}$ and $v \in (\mathcal{V} \cup \mathcal{I})$.* ∎

The simplicity of our model allows for a high flexibility that can accommodate a wide variety of entity representations. For instance, the connection between two entities, p_i and p_j, can be represented simply by assigning the id of p_j as an attribute value of p_i, and vice versa. A tag-style value v_i can be simply denoted by leaving empty the attribute name in the corresponding name-value pair $\langle \_, v_i \rangle$ (see Figure 1.2), whereas a missing value for the attribute n_j is represented through the

[1]A global identifier is an id that uniquely identifies an entity profile.

unspecified attribute value in $\langle n_j, \_\rangle$. Nested attributes are also supported, as they merely need to be transformed into a flat set of name-value pairs. On the whole, our model is general enough to represent the entities of the HHIS we are considering, such as the Web and Dataspace applications [MCD+07].

Two profiles, p_i and p_j, are *duplicates* (or *matches*) — denoted by $\mathbf{p_i \equiv p_j}$ — if they represent the same real-world object. Given two entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, the goal of entity resolution is to identify the duplicate profiles they contain. Depending on the relation between $\mathcal{E}_1$ and $\mathcal{E}_2$, we distinguish the following types of ER:

- *Clean-Clean ER*, where both $\mathcal{E}_1$ and $\mathcal{E}_2$ are individually duplicate-free (i.e., clean), but possibly overlapping entity collections.
- *Dirty-Clean ER*, where $\mathcal{E}_1$ is a clean collection, while $\mathcal{E}_2$ may contain duplicates (i.e. it is "dirty").
- *Dirty-Dirty ER*, where both $\mathcal{E}_1$ and $\mathcal{E}_2$ are dirty.

In all cases, the output of ER comprises the pairs of duplicate entities that are shared by $\mathcal{E}_1$ and $\mathcal{E}_2$ — denoted by $\mathcal{D}^{\mathcal{E}_1 \cap \mathcal{E}_2}$. For simplicity, we consider the last two sub-problems to be equivalent to *Dirty ER*: the input comprises a single, dirty entity collection $\mathcal{E}$ that is formed by the union of the given collections (i.e., $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$), while the output encompasses the set of matching pairs of entities that are contained in $\mathcal{E}$ — represented by $\mathcal{D}^{\mathcal{E}}$. Thus, in the following, we exclusively consider two versions of ER: Clean-Clean ER and Dirty ER. In the context of homogeneous information spaces, such as databases, the former problem is usually called *record linkage* and the latter *deduplication* [Chr12b].

3.2 Blocking for Entity Resolution

The exhaustive ER solutions constitute quadratic procedures that cannot scale to large data sets. To apply ER to voluminous data collections, blocking is typically employed in order to restrict the computational cost to comparisons between similar entities. Its goal is to group such entities into clusters called *blocks* so that it suffices to perform comparisons solely among the entities of each block.

In practice, blocking transforms every profile into a (set of) *blocking key(s)* that is suitable for clustering. Therefore, the similarity between two profiles should be reflected in the similarity of their blocking key(s). Entities with the same (or similar) key(s) are grouped together into blocks. This procedure is encapsulated by a *blocking scheme*, which consists of two parts:

- A *transformation function* f_t maps an entity profile to its blocking key(s). As an example, consider a function that transforms the entity profiles of Figure 1.1 (a) into a representation consisting of their value for the attribute "zip code".
- A set of *constraint functions* $\mathcal{F}_c$ encapsulates the conditions for placing entities into blocks. Each function $f_c^i \in \mathcal{F}_c$ essentially captures a boolean condition that decides for every entity profile whether it should be placed in block b_i. Continuing the previous example, the constraint function f_c^{91456} places an entity of Figure 1.1 (a) in the block b_{91456} of Figure 1.1 (b) as long as its transformation function represents it by the value "91456". The individual constraint functions are independent from each other and allow for every entity to be placed into multiple blocks, thus introducing redundancy.

Based on these definitions, a blocking scheme that applies to the Clean-Clean ER problem can be formalized as follows:

Definition 3.3. *A **blocking scheme** $bs^{\mathcal{E}_1 \times \mathcal{E}_2}$ for two input entity collections*[2]*, $\mathcal{E}_1$ and $\mathcal{E}_2$, is defined by a transformation function $f_t : \mathcal{E}_1 \cup \mathcal{E}_2 \mapsto K$ and a set of constraint functions $\mathcal{F}_c : K \mapsto \{true, false\}$, where K is the space of all possible blocking keys for the given entity profiles.* ■

A blocking scheme $bs^{\mathcal{E}}$ that applies to Dirty ER is defined in analogy.

Applying a blocking scheme to the input entity collection(s) yields a set of blocks $\mathcal{B}$ that is called *block collection*[3]. Depending on the ER problem at hand, the individual blocks it comprises may be of two types:

1. *Unilateral blocks* are the product of blocking-based Dirty ER. All entities they contain are possible matches, as they stem from the same dirty entity collection.

2. *Bilateral blocks* are the product of blocking-based Clean-Clean ER. Internally, they are partitioned in two inner sub-blocks that individually contain non-matching entities (i.e., entities stemming from the same clean input collection). Hence, only entities belonging to different inner blocks are possible matches.

[2]Note that the input entity collections may be represented in a different format. However, we assume that a preprocessing step transforms both of them into the data model described in Definitions 3.1 and 3.2.

[3]In view of the unambiguous relation between a blocking scheme and a block collection, we will use these two terms interchangeably.

Formally, these two types of blocks are defined as follows:

Definition 3.4. *Given an entity collection $\mathcal{E}$ and a blocking scheme $bs^{\mathcal{E}}$, a* ***unilateral block*** $b_i \in \mathcal{B}^{\mathcal{E}}$ *is the maximal subset of $\mathcal{E}$ defined by the transformation function f_t and the constraint function f_c^i such that: $b_i \subseteq \mathcal{E} \wedge \forall p \in \mathcal{E} : f_c^i(f_t(p)) = true \Leftrightarrow p \in b_i$.* ∎

Definition 3.5. *Given two entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, and a blocking scheme $bs^{\mathcal{E}_1 \times \mathcal{E}_2}$, a* ***bilateral block*** $b_i^{1,2} \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$ *is the maximal subset of $\mathcal{E}_1 \times \mathcal{E}_2$ that is defined by the transformation function f_t and the constraint function f_c^i. Its* non-empty inner blocks, *b_i^1 and b_i^2, are defined by the following conditions:*

$$\begin{aligned} \forall i : b_i^1 \subseteq \mathcal{E}_1 \wedge \forall p \in \mathcal{E}_1 : f_c^i(f_t(p)) &= true \Leftrightarrow p \in b_i^1, \text{ and} \\ \forall i : b_i^2 \subseteq \mathcal{E}_2 \wedge \forall q \in \mathcal{E}_2 : f_c^i(f_t(q)) &= true \Leftrightarrow q \in b_i^2. \quad ∎ \end{aligned}$$

The difference in the internal structure of these block types calls for different processing, as well. In the case of unilateral blocks, all entities are compared with each other, whereas for bilateral blocks, the entities of the one inner block are only compared with those of the other.

3.2.1 Blocking Scheme Quality

The quality of a blocking scheme bs over a (pair of) entity collection(s) is expressed in terms of the following two measures:

- The efficiency of bs is directly related to the number of comparisons contained in the resulting block collection $\mathcal{B}$. This number — denoted by $||\mathcal{B}||$ — is called *aggregate cardinality* and is equal to $||\mathcal{B}|| = \sum_{b_i \in \mathcal{B}} ||b_i||$, where $||b_i||$ is the *individual cardinality* of b_i (i.e., total number of comparisons entailed by block b_i). For a unilateral block, we have $||b_i|| = |b_i| \cdot (|b_i| - 1)/2$, while for a bilateral one we have $||b_i|| = |b_i^1| \cdot |b_i^2|$. Efficiency is also characterized by the *minimum aggregate cardinality* of a block collection $\mathcal{B}$, which is denoted by $||\mathcal{B}||_{min}$. This measure expresses the minimum number of comparisons that will be executed in the ideal case that we process the blocks of $\mathcal{B}$ using an *oracle* for entity matching (i.e., the perfect matching method that always decides whether two entities are matching with 100% accuracy). Apparently, $||\mathcal{B}||_{min} \leq ||\mathcal{B}||$ and $||\mathcal{B}||_{min} = \sum_{b_i \in \mathcal{B}} ||b_i||_{min}$, where $||b_i||_{min}$ stands for the *minimum individual cardinality* of block b_i.

- The effectiveness of bs is directly related to the portion of detected matches. Assuming a perfect matching method, it is derived from the ratio between the number of duplicates sharing at least one block and the actual matches contained in the input entity collection(s). The former is denoted by $|\mathcal{D}^{\mathcal{B}}|$, and the latter by $\mathcal{D}^{\mathcal{E}_1 \cap \mathcal{E}_2}$ for Clean-Clean ER and $\mathcal{D}^{\mathcal{E}}$ for Dirty ER.

There is a clear trade-off between the effectiveness and the efficiency of a blocking scheme: the more comparisons it entails (i.e., higher $||\mathcal{B}||$), the higher its effectiveness is expected to be at the cost of lower efficiency, and vice versa. Thus, a blocking scheme is considered successful if it achieves a good balance between these two competing objectives. This trade-off is commonly captured by the following three measures [BKM06, dVKCC09, MK06, PINF11]:

- **Pair Completeness** (PC) expresses how many of the matching pairs of entities have at least one block in common and, thus, can be detected. Given a bilateral block collection $\mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$, its value is derived from the following formula:
$$PC(\mathcal{B}) = \frac{|\mathcal{D}^{\mathcal{B}}|}{|\mathcal{D}^{\mathcal{E}_1 \cap \mathcal{E}_2}|}.$$
For a unilateral block collection $\mathcal{B}^{\mathcal{E}}$, it is equal to:
$$PC(\mathcal{B}) = \frac{|\mathcal{D}^{\mathcal{B}}|}{|\mathcal{D}^{\mathcal{E}}|}.$$
Apparently, PC takes values in the interval $[0, 1]$, with higher values indicating higher *effectiveness* of the blocking scheme.

- **Pairs Quality** (PQ) estimates the portion of executed comparisons that correspond to real pairs of duplicate entities. Given a block collection $\mathcal{B}$, it is defined as:
$$PQ(\mathcal{B}) = \frac{|\mathcal{D}^{\mathcal{B}}|}{||\mathcal{B}||_{min}}.$$
PQ takes values in the interval $[0, 1]$, with higher values indicating a lower number of unnecessary comparisons and, thus, higher *efficiency* of the blocking scheme.

- **Reduction Ratio** (RR) measures the reduction in the number of executed comparisons for a block collection $\mathcal{B}$ with respect to a baseline block collection $\mathcal{B}'$. It is defined as:

$$RR(\mathcal{B}, \mathcal{B}') = 1 - \frac{||\mathcal{B}||_{min}}{||\mathcal{B}'||_{min}}.$$

It takes values in the interval $[0, 1]$ (provided that $||\mathcal{B}||_{min} \leq ||\mathcal{B}'||_{min}$), with higher values denoting higher *efficiency* of the blocking scheme.

In general, high *PC* values satisfy the application requirements with respect to the acceptable level of effectiveness over HHIS, while high *PQ* and *RR* values mean that the ER process can be efficiently applied to large data sets.

Based on the above definitions, the blocking-based ER problems we are tackling in this work can be formalized as follows:

Problem 3.1 (Blocking-based Clean-Clean ER)**.** *Given two duplicate-free, but overlapping entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$, cluster the entities of $\mathcal{E}_1$ and $\mathcal{E}_2$ into blocks and process them so that both $PC(\mathcal{B})$ and $PQ(\mathcal{B})$ are maximized.* ■

Blocking-based Dirty ER is defined in analogy. Note that the requirement for maximizing *PC* and *PQ* simultaneously necessitates that the efficiency enhancements stem from the careful removal of unnecessary comparisons between irrelevant entities.

3.2.2 Internal Functionality of Blocking-based ER

Internally, blocking-based ER can be split into three sub-problems that are tackled in sequence. They are formally defined in the following, in the order they should be executed:

Problem 3.2 (Block Building)**.** *Given two duplicate-free, but overlapping entity collections*[4]*, $\mathcal{E}_1$ and $\mathcal{E}_2$, define a blocking scheme bs that produces a block collection $\mathcal{B}$ of the highest possible effectiveness (i.e., PC), while having a computational cost that is significantly lower than that of the* naive block collection $\mathcal{B}_{naive}$ *(i.e., $RR(\mathcal{B}, \mathcal{B}_{naive}) \gg 0$)*[5]. ■

Problem 3.3 (Meta-Blocking)**.** *Given a block collection $\mathcal{B}$, restructure it into a new one $\mathcal{B}'$ that achieves significantly higher levels of efficiency (i.e., $PQ(\mathcal{B}') \gg PQ(\mathcal{B})$*

[4]For Dirty ER, the input comprises a single entity collection $\mathcal{E}$.

[5]The naive block collection ($\mathcal{B}_{naive}$) places all input entities in a single block. Thus, it compares every entity will all others, involving $|\mathcal{E}_1| \cdot |\mathcal{E}_2|$ comparisons for Clean-Clean ER and $|\mathcal{E}| \cdot (|\mathcal{E}| - 1)/2$ for Dirty ER.

and $RR(\mathcal{B}', \mathcal{B}) \gg 0$*), while maintaining the original effectiveness (i.e.,* $PC(\mathcal{B}') \geq PC(\mathcal{B})$*).* ∎

Problem 3.4 (Block Processing)**.** *Given a block collection* $\mathcal{B}$*, examine its elements so as to maximize* $PQ(\mathcal{B})$*, while maintaining the original levels of* $PC(\mathcal{B})$*.* ∎

Note that Meta-Blocking constitutes a novel task that, as yet, has not been studied in the literature. Its goal seems similar to that of Block Processing, but it involves a fundamentally different functionality: Block Processing aims at detecting practically all duplicates with the minimum number of pairwise comparisons, while Meta-Blocking aims at restructuring the input block collection based on the information encapsulated in its block assignments — regardless of the block building method that created it. Ultimately, its target is to enhance the performance of Block Processing by feeding it with a block collection that achieves a better balance between effectiveness and efficiency than the original set of blocks.

Problems 3.2, 3.3 and 3.4 essentially constitute optimization tasks. Their interplay in the context of the overall blocking-based ER problem is outlined in Figure 1.3. We address them individually in Chapters 4, 5 and 6, respectively, through a set of best effort strategies.

3.3 Metric Space for Blocking Techniques

The goal of (meta-)blocking is to achieve the optimal balance between *PC* and *PQ*. In the ideal case, this balance would be tuned by knowing the values of these measures *a-priori* so as to guide the processing of the relevant techniques. However, these measures can only be quantified *a-posteriori*, by analytically processing the input block collection in order to estimate the number of executed comparisons and detected duplicates. To overcome this problem, we propose to resort to a-priori approximations of the actual values of these measures.

To this end, we introduce the *BC-CC* metric space that is illustrated in Figure 3.1. It is formed by two orthogonal measures that provide a close approximation to *PC* and *PQ*, respectively. Its horizontal dimension is called *Blocking Cardinality* (*BC*) and quantifies the redundancy of $\mathcal{B}$ as the average number of block assignments per entity of the input collection(s); as we have experimentally shown, *BC* is highly correlated with *PC* for redundancy-positive blocking schemes (i.e., higher *BC* values lead to higher effectiveness) [PIN+12]. The vertical dimension is called *Comparisons Cardinality* (*CC*) and expresses the distribution of

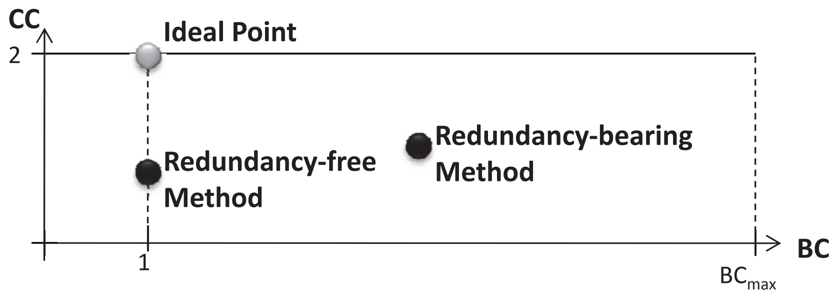


Figure 3.1: The *BC*-*CC* metric space and the mapping of the two main categories of blocking methods (black dots) in comparison with the ideal one (gray dot).

comparisons per block. Its value is directly related to *PQ* and *RR*, with higher *CC* values conveying higher efficiency.

More formally, *BC* is defined as follows:

Definition 3.6. *Given a* unilateral *block collection $\mathcal{B}^{\mathcal{E}}$, its* ***Blocking Cardinality*** *(BC) is defined as the average number of blocks $b_i \in \mathcal{B}^{\mathcal{E}}$ an entity profile $p \in \mathcal{E}$ is placed in:*

$$BC(\mathcal{B}) = \frac{\sum_{b_i \in \mathcal{B}^{\mathcal{E}}} |b_i|}{|\mathcal{E}|},$$

where $|b_i|$ and $|\mathcal{E}|$ denote the size of the block b_i and the input entity collection $\mathcal{E}$, respectively (i.e., the number of entity profiles they contain). ∎

Note that the value of *BC* depends not only on the blocking scheme at hand, but also on the data collection(s) it applies to. Thus, given a bilateral block collection $\mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$, we distinguish two different versions of *BC*: the Blocking Cardinality of the individual entity collections (BC_{ind}) and the Blocking Cardinality of their conjunction (BC_{ov})[6]. Their formal definitions are respectively the following:

Definition 3.7. *Given a* bilateral *block collection $\mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$, the* ***individual Blocking Cardinality*** *(BC_{ind}) of $\mathcal{E}_j$ is defined as the average number of inner blocks $b_i^j \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$ an entity profile $p \in \mathcal{E}_j$ is placed in:*

$$BC_{\text{ind}}(\mathcal{E}_j) = \frac{\sum_{b_i \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}} |b_i^j|}{|\mathcal{E}_j|},$$

where $j \in \{1, 2\}$. ∎

[6]Note that for Clean-Clean ER, the horizontal axis of the *BC*-*CC* metric space corresponds to the overall Blocking Cardinality BC_{ov} of $\mathcal{B}$.

Definition 3.8. *Given a* bilateral *block collection $\mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$, its* ***overall Blocking Cardinality*** *(BC_{ov}) is defined as the average number of blocks $b_i \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$ an entity profile $p \in (\mathcal{E}_1 \cup \mathcal{E}_2)$ is placed in:*

$$BC_{\text{ov}}(\mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}) = \frac{\sum_{b_i \in \mathcal{B}} |b_i|}{|\mathcal{E}_1| + |\mathcal{E}_2|},$$

where $|\mathcal{E}_1| + |\mathcal{E}_2|$ denotes the total size of the entity collections, $\mathcal{E}_1$ and $\mathcal{E}_2$. ■

To illustrate the functionality of *BC*, consider the blocks of Figure 1.4. They involve 19 block assignments that pertain to 4 distinct entity profiles, thus yielding a *BC* value equal to (19/4=)4.75. In general, *BC* takes values in the interval $[0, BC_{max}]$, where BC_{max} denotes the maximum reasonable *BC* value, which corresponds to the naïve method of placing all pairs of *comparable entities*[7] into a block of minimum size (i.e., $|b_i| = 2 \ \forall b_i \in \mathcal{B}$). For Clean-Clean ER over $\mathcal{E}_1$ and $\mathcal{E}_2$, it is equal to $BC_{max} = \frac{2 \cdot |\mathcal{E}_1| \cdot |\mathcal{E}_2|}{|\mathcal{E}_1| + |\mathcal{E}_2|}$, while for Dirty ER over $\mathcal{E}$ it is equal to $BC_{max} = |\mathcal{E}| - 1$. A *BC* value lower than 1 indicates a blocking method that fails to place each entity in at least one block. This is possible, for example, with blocking techniques that group entities based on the values of a specific attribute, thus ignoring the profiles that do not have it. A value equal to 1 denotes a technique that is close to a redundancy-free blocking method; it practically associates every entity with a single block, thus producing a set of (nearly) non-overlapping blocks. A value over 1 indicates redundancy-bearing methods (cf. Section 2.2.1), with higher *BC* values corresponding to higher levels of redundancy.

The vertical axis, *CC*, estimates the efficiency of a block collection through the number of block assignments that account for each comparison. For instance, the blocks of Figure 1.4 entail 15 comparisons that pertain to 19 entity-to-block associations and, thus, the corresponding value of *CC* amounts to (19/15=)1.27. The rationale behind *CC* is that a large set of individually small blocks is substantially more efficient than a small set of extremely large blocks; the former typically involves more block assignments, but contains fewer comparisons than the latter, thus resulting in higher *CC*. Therefore, the higher the value of *CC* is, the more efficient is the given block collection. Continuing our example, if we omit the two largest blocks of Figure 1.4 (b_{seller} and b_{91335}), we produce a more efficient block collection, whose *CC* value increases to 2.

Formally, *CC* is defined as follows:

[7]For Clean-Clean ER over $\mathcal{E}_1$ and $\mathcal{E}_2$, two entities, p_i and p_j, are comparable if $p_i \in \mathcal{E}_1$ and $p_j \in \mathcal{E}_2$. For Dirty ER, two entities, p_i and p_j, are comparable if they are different (i.e., $i \neq j$).

Definition 3.9. *Given a block collection $\mathcal{B}$, its **Comparison Cardinality** (CC) is defined as the ratio between the sum of block assignments and its aggregate cardinality:*

$$CC = \frac{\sum_{b_i \in \mathcal{B}} |b_i|}{||\mathcal{B}||},$$

where $|b_i|$ denotes the size of b_i. ■

CC takes values in the interval $[CC_{min}, CC_{max}]$, with higher values corresponding to fewer comparisons per block assignment and higher efficiency (i.e., smaller blocks, on average). Its maximum value CC_{max}=2 corresponds to the naïve solution of placing all pairs of comparable entities into a block of minimum size, while its maximum value CC_{min} corresponds to the naive block collection $\mathcal{B}_{naive}$. Thus, it is equal to $CC_{min} = \frac{|\mathcal{E}_1|+|\mathcal{E}_2|}{|\mathcal{E}_1|\cdot|\mathcal{E}_2|} (\ll CC_{max})$ for Clean-Clean ER and to $CC_{min} = \frac{|\mathcal{E}|}{|\mathcal{E}|\cdot(|\mathcal{E}|-1)/2} = \frac{2}{|\mathcal{E}|-1} \ll CC_{max}$ for Dirty ER.

On the whole, the *BC-CC* metric space is suitable for three tasks:

1. To a-priori estimate the values of *PC*, *PQ* and *RR*. This can be efficiently accomplished, as the *BC-CC* mapping of a block collection exclusively relies on the external features of its elements (i.e., size and cardinality) and can be computed in linear time (i.e., $O(|\mathcal{B}|)$).

2. To a-priori compare the performance of individual blocking schemes. In fact, the closer a blocking method is mapped to point (1,2) (refer to Figure 3.1), the better is its balance between *PC* and *RR* [PIN+12]. Indeed, point (1,2) represents the *Ideal Point*, since it corresponds to the *optimal blocking method*, i.e., the method that builds a block of minimum size for every pair of duplicates, thus involving neither repeated nor superfluous comparisons between non-matching entities. In our work, we use the *Ideal Point* as a reference point, as we will see in Chapter 5 — especially Sections 5.3.2 and 5.3.4.

3. It facilitates the development of blocking methods that enhance the effectiveness and/or the efficiency of a blocking scheme. An example for the latter case is the method presented in Section 6.2.3.

3.4 Summary

In this chapter, we formalized all concepts of blocking-based Entity Resolution that are relevant to our work. We also presented the formal definitions of the individual tasks that correspond to the three layers of our framework. They are all modeled as optimization problems, but in the following, we tackle them through a series of best effort strategies. Last but not least, we introduced the *BC-CC* metric space that facilitates the a-priori comparison of their relative performance as well as the a-priori estimation of the balance between effectiveness and efficiency they achieve.

Chapter 4

Block Building

The block building techniques receive as input one or two entity collections and aim at clustering their entities into blocks so that each pair of duplicates co-occur in at least one block, with a high probability. Most of these methods are designed for HOIS, relying on an a-priori known schema for their effectiveness. However, the intricacies of HHIS (cf. Chapter 1) render them inapplicable, calling for alternative techniques that inherently overcome the relevant challenges.

In this chapter, we introduce a series of novel block building techniques that rely on two fundamental principles in order to deal with the intricacies of HHIS:

- The *attribute-agnostic functionality* ensures that the creation of blocks completely disregards any schema information. The reason is that HHIS entail so high levels of noise and heterogeneity that attribute names cannot play a reliable role in the creation of blocks. Note that in the following, we use the terms attribute-agnostic and schema-agnostic interchangeably.
- The *redundancy-positive functionality* ensures that each entity is placed in multiple blocks so that the more blocks two entities have in common, the higher is the similarity of their profiles. This methodology constitutes a reliable means for reducing the likelihood of missed matches and is practically indispensable in the context of HHIS, due to the high levels of noise the latter entails. Note that all block collections incorporating this approach are mapped to the right of the x=1 axis on the *BC-CC* metric space.

Based on these principles, we have developed three groups of blocking methods. The most simple and general one, called Token Blocking, is presented in Section 4.1. It essentially creates a distinct block for each token shared by at least two input entities. Section 4.2 presents a refined version of this approach, called

Agnostic Clustering; it clusters attributes according to the similarity of their values and applies Token Blocking inside each cluster independently of the others; in this way, it creates blocks of higher efficiency and comparable levels of effectiveness. Finally, Section 4.3 presents a family of block building methods that are crafted for RDF data. They exploit the semantics in entity URIs, the links between entities as well as the tokens in the literal values of each entity in order to build block collections of higher performance than Token Blocking.

4.1 Token Blocking

This blocking scheme, which was originally introduced in [PINF11], is based on the following idea: each token t_i creates a distinct block b_i that contains all entities having t_i in the values of their profile — regardless of the associated attribute names. In this way, blocks are built in an attribute-agnostic manner, and every entity is placed in multiple blocks, ensuring a redundancy-positive functionality. There is only one restriction in this process: in the case of Dirty ER, a token t_i is **valid** (i.e., it creates a block) if it appears in at least two input entity profiles, while for Clean-Clean ER, it has to appear in entity profiles of both input sets. The latter requirement ensures that for each bilateral block, both inner blocks are non-empty; formally, this is expressed as $t_i \in (tokens(V_1) \cap tokens(V_2))$, where $tokens(V_j)$ represents the set of all tokens contained in the values V_j of the entity profiles in the input collection $\mathcal{E}_j$.

More formally, the transformation function f_t of this scheme converts an entity profile into the set of tokens contained in its attribute values:

$$f_t(p) = \left\{ t_i : \exists n_i, v_i : \langle n_i, v_i \rangle \in A_p \wedge t_i \in tokens(v_i) \right\},$$

where $tokens(v_i)$ is a function that returns the set of tokens comprising the value v_i. Its set of constraint functions $\mathcal{F}_c$ contains a function f_c^i for every **valid** token t_i; each f_c^i actually defines a block $b_i \in \mathcal{B}$ that contains all input entities having t_i in at least one of their attribute values. Thus, f_c^i encapsulates the following condition for placing an entity p in block b_i:

$$f_c^i(f_t(p)) = (t_i \in f_t(p)).$$

On the average case, the time complexity of this method for Clean-Clean ER is $O(BC_{\text{ov}} \cdot (|\mathcal{E}_1| + |\mathcal{E}_2|))$, while its space complexity is $O(|\bar{b_i}| \cdot (|tokens(V_1)| \cap |tokens(V_2)|))$, where $|\bar{b_i}|$ is the mean block size. For Dirty ER, the average time complexity is $O(BC \cdot |\mathcal{E}|)$ and the average space complexity is $O(|\bar{b_i}| \cdot |tokens(V)|)$.

Token Blocking has two major performance advantages:

- It scales to large-scale entity collections, as it can be efficiently implemented with the help of inverted indices.
- It is robust to noise and heterogeneity, because the likelihood of two matching entities sharing no block at all is very low. Indeed, this can only be the case when two matching entities have no token in common, a very unlikely situation for profiles describing the same real-world object.

However, the high levels of redundancy it entails have a negative impact on efficiency (i.e., high $||\mathcal{B}||$ and $||\mathcal{B}||_{min}$). Sections 4.2 and 4.3 present block building techniques that enhance its efficiency at a limited cost (if any) in effectiveness.

4.2 Agnostic Clustering Blocking

Token Blocking suffers from low efficiency, since it produces large blocks with a high portion of unnecessary comparisons. To enhance its efficiency at no cost in effectiveness, we could split its blocks into smaller ones, without separating their co-occurring duplicate entities. In this way, we can derive a new block collection that comprises a higher number of blocks, which are significantly smaller in size. The aggregate cardinality of the new collection will be lower and its *BC-CC* mapping will lie closer to the Ideal Point, thus indicating a better balance between effectiveness and efficiency.

Agnostic Clustering was introduced in [PIP$^+$ar] so as to serve exactly this goal. Its functionality exploits the attributes associated with each token, partitioning them into **non-overlapping** clusters according to the similarity of their values. The resulting groups are called *attribute clusters* (K) and are treated independently of each other: given a cluster $k \in K$, every valid token t_i of its values creates a distinct block, which contains all entities having t_i assigned to at least one attribute belonging to k. As a result, the partitioning of attributes into clusters leads to the partitioning of tokens and blocks, as well.

In more detail, Agnostic Clustering is equivalent to splitting each block b_i of Token Blocking according to the attribute clusters associated with t_i in the entity profiles of b_i. To illustrate this difference, imagine that token t_i is associated with n attributes, which belong to m attribute clusters ($m \leq n$). Token Blocking creates

Algorithm 4.1: Attribute Clustering Blocking.

```
    Input: Attribute sets: $N_1$, $N_2$, Attribute values: $V_1$, $V_2$
    Output: Set of attribute clusters: $K$
1   $links \leftarrow \{\}$;   $k_{glue} \leftarrow \{\}$;
2   foreach $n_{i,1} \in N_1$ do
3       $n_{j,2} \leftarrow getMostSimilarAttribute(n_{i,1}, N_2, V_2)$;
4       if $0 < sim(n_{i,1}.getValues(), n_{j,2}.getValues())$ then
5           $links.add(newLink(n_{i,1}, n_{j,2}))$;
6   foreach $n_{i,2} \in N_2$ do ... ; // same as with $N_1$
7   $links' \leftarrow computeTransitiveClosure(links)$;
8   $K \leftarrow getConnectedComponents(links')$;
9   foreach $k_i \in K$ do
10      if $|k_i| = 1$ then  $K.remove(k_i)$;  $k_{glue}.add(k_i)$;
11  $K.add(k_{glue})$;
12  return $K$;
```

a single block for t_i, with all entities having it in their values — regardless of the associated attributes. In contrast, Agnostic Clustering creates (at most) m distinct blocks — one for each attribute cluster; every block contains all entities having at least one attribute of the corresponding cluster associated with t_i. Given that the number of entities remains the same in both cases, the blocks of Agnostic Clustering are expected to be more in number and individually smaller, thus having a higher CC value than Token Blocking. In fact, the higher m is, the higher gets the value of CC. On the other hand, BC exhibits a slight increase, due to the tokens that are associated with multiple attribute clusters within the same entity profile (note that this implies that Agnostic Clustering involves a redundancy-positive functionality, just like Token Blocking).

The functionality of Agnostic Clustering for Clean-Clean ER is outlined in Algorithm 4.1. In essence, it works as follows: every attribute from N_1 is associated with the *most similar* attribute of N_2 (Lines 2-5), and vice versa (Line 6). The link between two attributes is stored in a list (Line 5) on the sole condition that the similarity of their values exceeds zero (Line 4), a score that actually implies dissimilarity. The transitive closure of the stored links is then computed (Line 7) to form the basis for partitioning attributes into clusters: each connected component of the transitive closure corresponds to a distinct attribute cluster (Line 8). The resulting set of attribute clusters is examined for *singleton clusters*, i.e., those containing a single attribute that was associated with no other. All these clusters are then merged into a new one, which is called *Glue Cluster* and is symbolized as k_{glue} (Line 10). In this way, we ensure that no attributes and, thus, no tokens are excluded from the block building procedure. The time complexity of the overall procedure is $O(|N_1|{\cdot}|N_2|)$, while its space complexity is $O(|N_1|+|N_2|)$, where $|N_1|$ and $|N_2|$ stand for the number of distinct attribute names in $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively.

For Dirty ER, the functionality of Agnostic Clustering remains almost the same as in Algorithm 4.1. The only difference is that the two loops in Lines 2-6 are replaced with a single one that compares every attribute with all others. Hence, the time complexity is quadratic and the space complexity is linear in the number of input attribute names (i.e., $O(|N|^2)$ and $O(|N|)$, respectively).

Relation to Schema Matching. Agnostic Clustering seemingly operates as a schema matching algorithm. As a matter of fact, though, there are three fundamental differences between these two approaches:

1. They serve different goals. Schema matching tries to partition the given attributes into clusters of semantically equivalent ones, whereas Agnostic Clustering aims at deriving attribute clusters that produce blocks with a comparison distribution that has a short tail (i.e., high *CC* values). The latter approach actually involves a schema-agnostic functionality, as it partitions the given attributes without considering any knowledge about their meaning or their name; instead, it merely takes into account the similarity of their values.

2. Agnostic Clustering associates singleton attributes with each other, a practice that is incompatible with the goal of schema matching.

3. Agnostic Clustering is inherently capable of supporting the unprecedented levels of heterogeneity in HHIS. In contrast, the state-of-the-art schema matching techniques are crafted for the limited attribute spaces of HOIS. The number of mappings they produce grows extremely fast with respect to the given attributes and, thus, they cannot scale to large schema spaces of HHIS [PINF11, NMMMBLP07, RB01].

As a result, the existing schema matching techniques cannot be employed in the place of Agnostic Clustering. In the following section, we examine the parameters that determine its functionality.

4.2.1 Representation Models & Similarity Metrics

The most critical part of Algorithm 4.1 is the function that estimates the similarity between two attributes (i.e., $getMostSimilarAttribute$). Internally, its functionality relies on two components:

- the model that collectively represents the set of values associated with each attribute, and
- the metric that assesses the similarity (i.e., common patterns) between the representation of two attributes.

The combination of a representation model and a similarity metric is called *clustering settings*. In the following, we elaborate on three such settings that have been established in the literature [GMP+12]. We analytically examine their relative performance in Section 7.2.

Term Vector in conjunction with Cosine Similarity

The term vector representation model transforms a set of values V into a Cartesian space, where each dimension corresponds to a distinct token contained in V. Thus, an attribute $n_k \in N$ is represented by a (sparse) vector $\bar{n_k}$, whose i-th coordinate denotes the $TF(t_i)\times IDF(t_i)$ weight of the corresponding token t_i [MRS08]. $TF(t_i)$ represents the *Term Frequency* of t_i, i.e., how many times t_i is associated with the specific attribute n_k, while $IDF(t_i)$ is equal to $\log(|N|/|N(t_i)|)$, where N stands for the input set of attributes and $N(t_i)$ for the subset of attributes associated with t_i ($N(t_i)\subseteq N$). For instance, the attribute in the pair $<n_k, v_k>=<name,$ "*home phone*"$>$ is represented by the vector $\bar{n_k}=\{TF(home)\times IDF(home), TF(phone)\times IDF(phone), 0,\ldots,0\}$.

In this context, the relevance between two attributes, n_1 and n_2, is quantified through the cosine similarity $CS(n_1, n_2)$ of the corresponding vectors:

$$CS(n_1, n_2) = \frac{\bar{n_1} \times \bar{n_2}}{||\bar{n_1}|| \times ||\bar{n_1}||}.$$

CS takes values in the interval [0, 1], with higher values indicating higher similarity between the given vectors/attributes.

The main drawback of these clustering settings is that they have to distinguish between *synonyms* and *homonyms* in order to yield high performance. The former term refers to different words having the same meaning (e.g., "buy" and "purchase"), while the latter refers to identical words occurring with different meaning;

as an example of homonyms, consider the word "left", which means either the past tense of leave or the opposite of right. Such ambiguities are typically resolved through language-specific, pre-processing techniques, like stemming, lemmatization and part-of-speech tagging [MRS08]. However, their effectiveness is degraded by the high levels of noise contained in HHIS (e.g., spelling mistakes). In addition, the entity profiles of HHIS can be written in any language(s), without necessarily conveying any information about it(them). For these reasons, in the following we employ these clustering settings independently of any pre-processing techniques.

Character N-grams in conjunction with Jaccard Similarity

This representation model overcomes both shortcomings of the above one: it is inherently tolerant to noise and requires no language-specific pre-processing techniques for higher performance (i.e., language-neutral functionality). In essence, it models each attribute as the set of n-grams (i.e., substrings of n consecutive characters) that appear in its values [MRS08]. Most commonly, the parameter n is set equal to 3, with the corresponding model called *character trigrams*. For instance, the attribute in the pair $<n_k, v_k>=<name$, "*home phone*"> is represented as $\{hom, ome, me\_, \_ph, pho, hon, one\}$.

In this context, the relevance between two attributes, n_1 and n_2, is defined as the Jaccard similarity $JS(n_1, n_2)$ of their trigrams:

$$JS(n_1, n_2) = \frac{|trigrams(n_1) \cap trigrams(n_2)|}{|trigrams(n_1) \cup trigrams(n_2)|},$$

where function $trigrams(n_k)$ produces the trigrams representation of the attribute n_k. JS takes values in the interval $[0, 1]$, with higher values indicating higher similarity.

Character N-gram Graphs in conjunction with Value Similarity

This representation model enhances the previous one by connecting with edges those n-grams that co-occur within a sliding window of n characters (i.e., neighboring n-grams) [GKVS08]. The edges are actually weighted in proportion to the frequency of co-occurrence and encapsulate contextual information that ensures higher effectiveness. Similar to the above clustering settings, n is usually set equal to 3, with the corresponding model called *character trigram graphs*. To illustrate its difference from the character trigrams model, consider Figure 4.1,

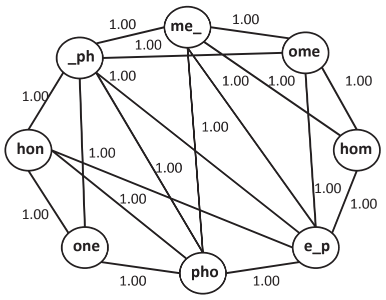


Figure 4.1: The trigram graph for value “home_phone”.

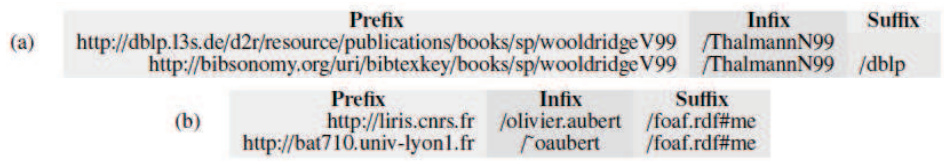
which depicts the corresponding graph representation for the attribute n_k in the pair $<n_k, v_k>=<name$, “*home phone*”>; apparently, the trigram graph conveys much more information than the plain bag representation of trigrams.

The n-gram graphs represent each attribute value by an individual graph, which we call *value graph*. The set of values corresponding to an attribute is also represented by a single graph. We call it *attribute graph* and derive it from the merge of the individual value graphs. Thus, it comprises the union of the nodes and the edges of the value graphs, with the weight of every edge converging to the mean weight of the corresponding edges in the value graphs (for more details, see [GP10]).

The relevance of two attributes n_1 and n_2 can be estimated through any graph metric that assesses the similarity of the corresponding attribute graphs, G^1 and G^2. For higher effectiveness, we employ a metric that considers both the number of common edges and their relative weights. It is called *value similarity* (VS) and essentially expresses the portion of common edges sharing exactly the same weight [GKVS08]. More formally, each common edge $e \in (G^1 \cap G^2)$ contributes $\mathrm{VR}(e)/\max(|G^1|, |G^2|)$ to VS, where VR(e) is the value ratio, i.e., a symmetric, scaling factor that is defined as $\mathrm{VR}(e)=\min(W^1(e), W^2(e))/\max(W^1(e), W^2(e))$, where $W^k(e)$ is the weight of edge e in graph G^k ($k \in \{1, 2\}$). $\mathrm{VR}(e)$ takes values in the interval $[0, 1]$, with 0 corresponding to non-matching edges that do not contribute to VS (i.e., $\mathrm{VR(e)}=0\ \forall e \notin (G^1 \cap G^2)$). Plugging all these measures together, we have:

$$VS(G^1, G^2) = \frac{\sum\limits_{e \in (G^1 \cap G^2)} \frac{\min(W^1(e), W^2(e))}{\max(W^1(e), W^2(e))}}{\max(|G^1|, |G^2|)}.$$

VS is defined in the interval $[0, 1]$, with higher values indicating higher similarity. Its maximum value VS_{max}=1 actually corresponds to a pair of graphs with identical



(a)

Prefix	Infix	Suffix
http://dblp.l3s.de/d2r/resource/publications/books/sp/wooldridgeV99	/ThalmannN99	
http://bibsonomy.org/uri/bibtexkey/books/sp/wooldridgeV99	/ThalmannN99	/dblp

(b)

Prefix	Infix	Suffix
http://liris.cnrs.fr	/olivier.aubert	/foaf.rdf#me
http://bat710.univ-lyon1.fr	/~oaubert	/foaf.rdf#me

Figure 4.2: Examples of matching pairs of URIs, split in the PI(S) scheme.

nodes, edges and weights.

Similar to the previous clustering settings, the advantage of the current ones is their language-agnostic, noise-tolerant functionality. However, they are able to capture more reliable similarity patterns than the plain *n*-grams model, due to the contextual information that is contained in their weighted edges.

4.3 URI Semantics Blocking[1]

The blocking schemes of this category are crafted for RDF data, relying primarily on evidence drawn from the semantics of entity identifiers. In fact, URIs often contain clues about the corresponding entity, even though the *W3C* explicitly discourages users from incorporating semantics into them [JW04]. These semantics come in the form of substrings that are suitable for matching entities and for clustering them into blocks. Indeed, we experimentally verified that around two thirds of the 182 million URIs of the 2009 Billion Triple Challenge data set[2] follow the *Prefix-Infix(-Suffix)* scheme — PI(S) for short — that is depicted in Figure 4.2 [PDKF10]. Matching the entities on the basis of equivalent infixes actually yields a Precision well over 90% and a Recall exceeding 70%.

Each component of the PI(S) form plays a special role: the *Prefix* part contains information about the source (i.e., domain) of the URI, the *Infix* part is a sort of a local identifier, and the optional *Suffix* part contains either details about the format (e.g., `.rdf` and `.n3`), or a named anchor. Apparently, the infixes of URIs contain the most discriminative information for our purpose, since they are more *source-independent* than the prefixes and the suffixes [PDKF10]. As an example, consider the duplicate entities of Pair (a) in Figure 4.2; despite the high heterogeneity in the prefixes of their URIs, the infix remains the same, while the suffix (e.g., `dblp`) is optional and can be ignored when matching URIs. It is possible, therefore, to create

[1]Originally introduced in [PIN+12].
[2]`http://km.aifb.kit.edu/projects/btc-2009`

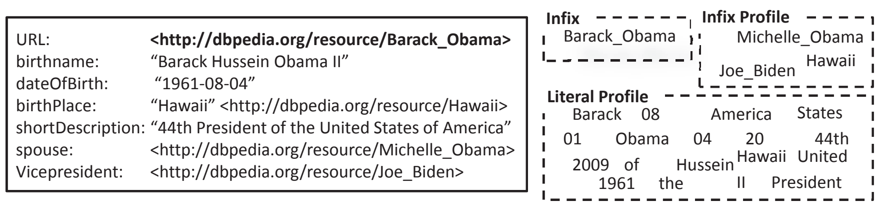

URL: <http://dbpedia.org/resource/Barack_Obama>
birthname: "Barack Hussein Obama II"
dateOfBirth: "1961-08-04"
birthPlace: "Hawaii" <http://dbpedia.org/resource/Hawaii>
shortDescription: "44th President of the United States of America"
spouse: <http://dbpedia.org/resource/Michelle_Obama>
Vicepresident: <http://dbpedia.org/resource/Joe_Biden>
Infix
Barack_Obama
Infix Profile
Michelle_Obama
Joe_Biden
Hawaii
Literal Profile
Barack 08 America States 01 Obama 04 20 44th 2009 of Hussein Hawaii United 1961 the II President

Figure 4.3: Illustration of the description items of an entity profile that are used by our blocking schemes.

blocks on the basis of equivalent infixes that are extracted from entity identifiers.

However, there are certain limitations to this approach:

- It is possible that an entity identifier contains no semantics. This is particularly true in the case of blank nodes[3] and of arbitrary or numerical URIs.
- Some matching entities are likely to have non-identical infixes, due to noise or divergent information. As an example, consider the matching entities of Pair (b) in Figure 4.2, where the two duplicate profiles have dissimilar infixes (`~oaubert` and `olivier.aubert`).

These situations can be resolved through blocking methods that rely on evidence drawn from different parts of entity profiles. We consider the following two methods:

- We investigate the possibility of creating blocks by exploiting the relationships between entities; this approach seems similar to Semantic Indexing [NMMMBLP07], but is fundamentally different in that it exclusively relies on the semantics contained in the identifiers of directly affiliated resources.
- We explore the creation of blocks on the basis of *literal values*, i.e., objects in RDF statements that are neither URIs nor blank nodes.

The common characteristic of the above blocking methods is their attribute-agnostic functionality, since they completely disregard any knowledge about attribute names. Instead, they exclusively consider the rest of the description items that are contained in an entity profile. As depicted in Figure 4.3, they actually partition an entity profile p_{id} into three parts that are suitable for schema-agnostic blocking:

[3]Blank nodes constitute anonymous nodes in RDF graphs and are typically used whenever there is no available information about the corresponding resource. Consequently, their identifiers do not carry any semantics.

- The *Infix* of p_{id} is extracted from its identifier id.
- The *Infix Profile* of p_{id} ($IP_{p_{id}}$) is the set of the infixes of all URIs contained in p_{id} as attribute values, with the exception of its own identifier (i.e., id). More formally:

  $$IP_{p_{id}} = \{infix(id_i) : id_i \in \mathcal{I} \wedge \exists n_i :< n_i, id_i > \in A_{p_{id}} \wedge id_i \neq id\},$$

  where $infix(id_i)$ is a function that extracts the infix from the given entity identifier id_i[4]. Thus, the Infix Profile of p_{id} captures the semantics encapsulated in the URIs of all entities that are directly associated with p_{id}.
- The *Literal Profile* of p_{id} ($LP_{p_{id}}$) comprises the set of all tokens of the literal values contained in it. More formally:

  $$LP_{p_{id}} = \{t_i : \exists n_i, v_i :< n_i, v_i > \in A_{p_{id}} \wedge t_i \in tokens(v_i) \wedge v_i \notin \mathcal{I}\},$$

  where $tokens(v)$ is the function employed by Token Blocking in order to tokenize a string value v on all its special characters (i.e., characters that are neither letters nor digits).

We call the blocking methods that solely rely on one category of the aforementioned information **atomic blocking schemes**. In contrast, the **composite blocking schemes** consider evidence from two or more of the above information sources. We elaborate on the former category in Section 4.3.1 and on the latter in Section 4.3.2. Note that, before applying these blocking schemes to a collection of RDF statements, we first generate the entity profiles that are contained in it according to Definition 3.2: all triples with a common subject s comprise an entity profile p_s that has s as its identifier, while the respective predicates and objects correspond to the set A_{p_s} of its name-value pairs. Note that the same procedure applies to blank nodes, as well, provided that the identifier assigned to each of them is consistent across all triples pertaining to the corresponding entity.

4.3.1 Atomic Blocking Schemes

In this section, we formalize the three atomic schemes that stem from the aforementioned entity parts.

[4]For an efficient approach to splitting a collection of URIs into the PI(S) form, see [PDKF10].

Infix Blocking

This method builds blocks on the equality of infixes, i.e., every block is associated with a specific infix and contains all entities that share it. Its transformation function converts an entity profile p_{id} into the infix of its id:

$$f_t(p_{id}) = infix(id).$$

Its constraint function f_c^i places an entity in block b_i if its infix is identical to i:

$$f_c^i(p_{id}) = (i \equiv infix(id)), \text{ where } i \text{ is one of the infixes occurring in the data set.}$$

Apparently, this is a redundancy-free blocking scheme (the only one presented in this dissertation), as every entity is placed in just one block, together with all other entities sharing the same infix. For example, the entity of Figure 4.3 is placed solely in the block that corresponds to the infix "Barack_Obama". The resulting blocks are non-overlapping (i.e., $BC \leq 1$), of small size and high efficiency (CC), because of the distinctive information of infixes. The drawback, though, is that their effectiveness as well as their robustness are rather limited: Infix Blocking does not apply to entities with a blank node or a random URI as their id, since the corresponding identifiers have a local scope and do not provide a meaningful infix for blocking and entity matching. It also fails to match duplicate entities with different infixes, due to noise.

Infix Profile Blocking

This is the second blocking scheme that creates blocks on the equality of infixes. It differs, though, from the previous one in that it exclusively considers infixes from the Infix Profiles of the given entities. Its transformation function represents every entity profile p_{id} by its Infix Profile, $IP_{p_{id}}$:

$$f_t(p_{id}) = IP_{p_{id}}.$$

Its constraint function f_c^i associates an entity with block b_i if the infix i is contained in its Infix Profile:

$$f_c^i(p_{id}) = (i \in IP_{p_{id}}), \text{ where } i \text{ is one of the infixes occurring in the data set.}$$

The cardinality of $IP_{p_{id}}$ is typically larger than one, since every entity is usually associated with many others. Therefore, most of the entities have a non-empty Infix Profile and are placed in multiple blocks, which are now overlapping (i.e., $1<BC$ if the majority of entities have a non-empty Infix Profile). For example, the entity of Figure 4.3 is contained in the three blocks that correspond to the infixes “Michelle_Obama”, “Hawaii”, and “Joe_Biden”. This redundancy-positive functionality leads to larger blocks, on average, and to lower CC values than Infix Blocking. Therefore, its mapping on the BC-CC space lies to the right of Infix Blocking on the X-axis and lower than it on the Y-axis (assuming that both schemes are applied to the same entity collections). Thus, it exhibits lower efficiency to the benefit of higher effectiveness and robustness.

In more detail, this blocking scheme has the potential to cover duplicates that Infix Blocking misses; even though blank nodes and numerical URIs have no infix, their infix profile can be non-empty. The same applies to matching entities with non-identical infixes, since their Infix Profiles are likely to share at least one infix of their related entities. However, the coverage of this strategy is also limited, since it does not apply to entity profiles that exclusively contain literal values. In addition, entities with numerical/arbitrary URIs tend to be related to other numerical/arbitrary URIs of the same domain, thus lacking an exploitable Infix Profile, unless they are re-used in other domains.

Literal Profile Blocking

As its name suggests, this blocking scheme forms blocks on the equality of the tokens that are extracted from the Literal Profiles of the given entities. As a result, each block corresponds to a single token and each entity is associated with multiple blocks. For instance, the entity depicted in Figure 4.3 will be placed in 18 blocks, one for each token in its Literal Profile — provided that they are all valid (cf. Section 4.1). The underlying transformation function converts every entity profile p_{id} into its Literal Profile, $LP_{p_{id}}$:

$$f_t(p_{id}) = LP_{p_{id}}.$$

Its constraint function f_c^i places an entity in block b_i if the token t_i is contained in its Literal Profile:

$$f_c^i(p_{id}) = (t_i \in LP_{p_{id}}), \text{ where } t_i \text{ is a token.}$$

This blocking scheme is similar to Token Blocking, with the only difference that it completely disregards all URIs contained in entity profiles as values (these URIs are exclusively handled by Infix Profile Blocking). This means that it only fails to cover profiles that lack any valid token in their literal values, thus achieving higher effectiveness than the other atomic schemes in most of the cases. Its efficiency, though, is expected to be lower, due to the higher degree of redundancy it involves: each block corresponds to a single token, with the number of entities sharing a token being typically higher than those sharing an infix (infixes normally consist of several tokens concatenated together, and, thus, are less common than individual tokens). This results in blocks that are larger, on average, and yield lower values of CC.

On the whole, its mapping on the *BC-CC* space lies to the right of Infix Blocking on the X-axis and lower than it on the Y-axis (assuming that both schemes are applied to the same entity collection).

4.3.2 Composite Blocking Schemes

In isolation, the above atomic schemes are of limited robustness, as their effectiveness depends on the characteristics of the entity collection(s) at hand. To remedy this situation, we propose their combination into composite blocking schemes. Given that the individual atomic schemes rely on different aspects of entity profiles, they are *complementary* and lead to higher effectiveness when combined. This can be accomplished simply by merging the individual block collections they produce. Let K_1 and K_2 symbolize the set of blocking keys of the block collections, $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively; their union $\mathcal{B}_1 \cup \mathcal{B}_2$ forms a new block collection $\mathcal{B}'$ in the following way: blocks that correspond to a common key $k_i \in (K_1 \cap K_2)$ are replaced by their merge, while blocks that correspond to a unique key $k_j \in (K_1 \triangle K_2)$ are included in $\mathcal{B}'$ without any modifications. As an example, consider the block collections $\mathcal{B}_1 = \{b_{obama} = \{p_1, p_2\}, b_{michelle} = \{p_3, p_4\}\}$ and $\mathcal{B}_2 = \{b_{michelle} = \{p_3, p_5\}, b_{hawai} = \{p_4, p_5\}\}$; their union is the block collection $\mathcal{B}' = \{b_{obama} = \{p_1, p_2\}, b_{michelle} = \{p_3, p_4, p_5\}$, $b_{hawai} = \{p_4, p_5\}\}$. Note that this merge has the potential to turn *singleton blocks* (i.e., blocks that contain a single entity) into valid ones, by merging those stemming from different blocking schemes, but corresponding to the same key. Continuing our example, imagine that $\mathcal{B}_1$ and $\mathcal{B}_2$ contained the singleton blocks $b_{barack} = \{p_3\}$, $b_{barack} = \{p_5\}$, respectively; in this case, their union $\mathcal{B}'$ would comprise the additional valid block $b_{barack} = \{p_3, p_5\}$.

The composite schemes consider multiple entity parts, thus achieving higher

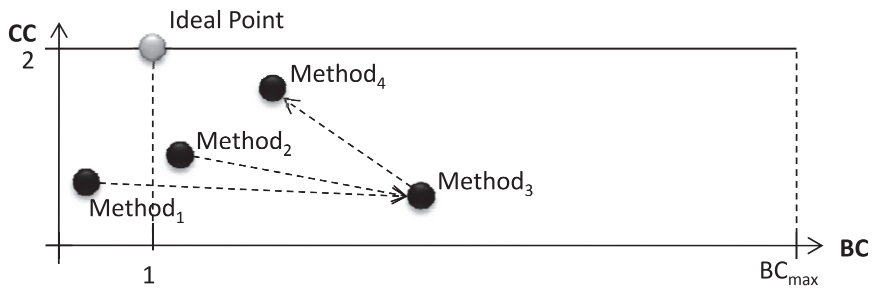


Figure 4.4: The effect of merging two individual blocking schemes ($Method_1$ and $Method_2$) into a composite one ($Method_3$), which is more robust and effective, on the BC-CC space. Although this practice leads to more comparisons, the higher robustness it conveys allows for enhancing efficiency through meta-blocking (cf. Chapter 5) and block processing techniques (cf. Chapter 6). Their effect is to move the mapping $Method_3$ to $Method_4$, which is closer to the Ideal Point.

robustness and effectiveness. This comes, however, at the cost of lower efficiency, since the resulting block collections entail more blocks that are usually larger in size, due to the merge of the blocks corresponding to common keys (i.e., keys that stem from different atomic blocking schemes, but correspond to the same token or infix). In Figure 4.4, we illustrate the effect of merging atomic blocking schemes into composite ones on the BC-CC space. Combining $Method_1$ with $Method_2$ leads to $Method_3$ that has a higher BC value (i.e., higher redundancy) and a lower CC value (i.e., restricted efficiency), due to the larger, on average, blocks it creates.

In the following paragraphs, we present all four possible combinations of the above atomic schemes and explain the rationale behind them. Note that all of them involve a redundancy-positive functionality.

Complete Infix Blocking

This blocking scheme is derived from the combination of Infix Blocking with Infix Profile Blocking. Thus, its transformation function extracts from an entity profile p_{id} the union of its infix and its Infix Profile:

$$f_t(p_{id}) = infix(id) \cup IP_{p_{id}}.$$

Blocks are built on the equivalence of infixes, with each block corresponding to a single infix and each entity potentially placed in multiple blocks. More formally, the constraint function for block b_i is defined as follows:

$$f_c^i(p_{id}) = (i \equiv infix(id)) \vee (i \in IP_{p_{id}}), \text{ where } i \text{ is an infix.}$$

Compared to Infix Blocking, this scheme covers profiles with a synthetic identifier (i.e., a blank node or a random URI) and is able to match entities with non-identical infixes. Compared to Infix Profile Blocking, it applies to entities with empty infix profiles which, nevertheless, have an infix. Most importantly, though, this combination takes advantage of infixes that originally resulted in singleton blocks, due to their scarcity in the individual key sets (i.e., they were associated with just one profile). As verified in Section 7.2, this results in a considerably higher levels of redundancy and, thus, in enhanced robustness and effectiveness. In fact, the only case that this scheme is not applicable is for entities that lack any meaningful URI in their profile; these are entities that have an arbitrary, synthetic URI as their id and are solely associated with identifiers of the same kind as well as with literal values.

Infix-Literal Profile Blocking

This scheme results from the combination of Infix Blocking with Literal Profile Blocking. Hence, its transformation function represents an entity p_{id} by the union of its infix and its Literal Profile:

$$f_t(p_{id}) = infix(id) \cup LP_{p_{id}}.$$

Its constraint functions build blocks on the equality of infixes or tokens:

$$f_c^i(p_{id}) = (i \equiv infix(id)) \vee (i \in LP_{p_{id}}), \text{ where } i \text{ is an infix and/or a token.}$$

Note that these two atomic schemes are likely to share a considerable portion of blocking keys. The reason is that some infixes merely consist of a single token that is also used as blocking key by Literal Profile Blocking. As an example, consider the infix "Hawaii" of the entity in Figure 4.3, which appears as token in a literal value, as well. The individual blocks that are common among the atomic schemes are merged into new blocks of larger size, thus inducing lower efficiency (i.e., lower CC values). However, robustness and effectiveness are substantially enhanced, since the only profiles that are not covered by this composite scheme are those having a blank node or a random URI as id, while containing no literal values in their set of attribute values. Apparently, this case is highly unlikely.

Infix Profile-Literal Profile Blocking

This scheme is derived from the combination of Infix Profile Blocking with Literal Profile Blocking. Thus, its transformation function models an entity profile p_{id} as the union of its Infix Profile and its Literal Profile:

$$f_t(p_{id}) = IP_{p_{id}} \cup LP_{p_{id}}.$$

Its constraint functions define blocks on the equality of infixes or tokens:

$$f_c^i(p_{id}) = (i \in IP_{p_{id}}) \vee (i \in LP_{p_{id}}), \text{ where } i \text{ is an infix and/or a token.}$$

Similar to the above blocking scheme, it involves higher redundancy than the individual blocking schemes it comprises (i.e., higher *BC* value), thus exhibiting higher effectiveness coupled with higher robustness. The only profiles that lie out of its scope are those containing solely blank nodes and random URIs in their set of attribute values.

Total Description Blocking

This blocking scheme is formed by the combination of all three atomic schemes of Section 4.3.1. Hence, it represents an entity profile p_{id} by the infixes of all URIs contained in it and the tokens of all its literal values:

$$f_t(p_{id}) = infix(id) \cup IP_{p_{id}} \cup LP_{p_{id}}.$$

Blocks are defined on the equality of tokens or infixes:

$$f_c^i(p_{id}) = (i \equiv infix(id)) \vee (i \in IP_{p_{id}}) \vee (i \in LP_{p_{id}}), \text{ where } i \text{ is an infix and/or a token.}$$

The interplay of all atomic blocking methods leads to the highest robustness among all schemes of URI Semantics Blocking. It fails only in the highly unlikely case where an entity profile has a non-meaningful identifier as its id and exclusively contains identifiers of this kind in its attribute values (i.e., it contains no valid tokens). This results not only in the highest effectiveness among all methods, but also in the highest *BC* value. It actually entails the largest block collection, with its blocks having the largest average size. Their aggregate cardinality is considerably higher, thus yielding lower efficiency and lower *CC* values.

4.4 Summary

We started this chapter by introducing Token Blocking as the basic schema-agnostic, redundancy-positive technique for building blocks. We then presented an enhancement, called Agnostic Clustering, which yields blocks of equally high effectiveness, but of higher efficiency. We explained that its clustering algorithm can be coupled with established representation models and string similarity metrics, thus allowing for a versatile functionality. Considering the special characteristics of RDF data, we enhanced Token Blocking with new approaches that are inherently crafted for them. To this end, we introduced a tripartite categorization of the evidence contained in an entity profile and explained how every category lays the basis for an atomic block building technique. We argued that these blocking schemes are complementary and that their combinations lead to higher robustness and effectiveness, while significantly outperforming Token Blocking with respect to efficiency. The relative performance of all these blocking schemes is systematically analyzed and evaluated in Section 7.2.

Chapter 5

Meta-Blocking

Meta-blocking has been formalized in Definition 3.3 as the task of restructuring a block collection in order to improve its efficiency at a limited, if any, cost in effectiveness. In fact, the *PC* of the new block collection can be higher than that of the input one, provided that meta-blocking infers new entity connections from the original ones. In this dissertation, though, we consider this inference problem to be orthogonal to the problem of improving the efficiency of a blocking scheme without affecting its effectiveness. We actually focus on enhancing the output of our block building techniques. Thus, our meta-blocking techniques rely on the intrinsic characteristic of the resulting, redundancy-positive block collections: the similarity of two entities is proportional to the number of blocks they have in common.

Based on this principle, our techniques aim at identifying the most similar pairs of entities so as to place them in the restructured blocks of the output. At their core lies the *blocking graph*, a data structure that models the block assignments of the input block collection in an abstract way that decouples meta-blocking from the underlying block building method. Its nodes correspond to the clustered entities and its edges connect every pair of *co-occurring entities* (i.e., entities that share at least one block). As an example, consider the blocking graph of Figure 5.1 (a), which corresponds to the blocks of Figure 1.4. The edges of a blocking graph are naturally undirected and weighted according to a scheme that determines the trade-off between the computational cost and the gain of comparing the adjacent entities (i.e., the benefit for the recall of the ER process, in case the entities are matching). In the example of Figure 5.1 (a), we present the simplest scheme, which sets the weight of each edge equal to the number of blocks the adjacent entities have in common.

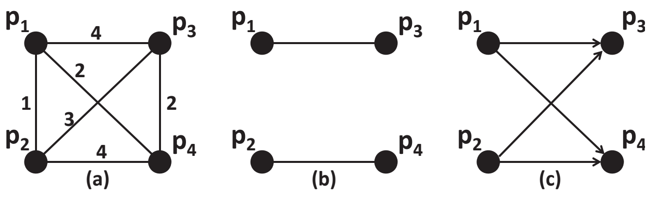


Figure 5.1: (a) The blocking graph of the block collection in Figure 1.4, (b) the pruned blocking graph, and (c) an alternative pruned blocking graph, discussed in Section 5.4.

Formally, the blocking graph for a unilateral block collection is defined as follows:

Definition 5.1. *Given a* unilateral *block collection $\mathcal{B}^{\mathcal{E}}$, the* ***undirected blocking graph*** *derived from it is a graph $\mathcal{G}_{\mathcal{B}} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$, where $V_{\mathcal{B}}$ is the set of its nodes, $E_{\mathcal{B}}$ is the set its undirected edges, and WS is the weighting scheme that determines the weight of every edge in the interval $[0,1]$. $V_{\mathcal{B}}$ contains all entities of $\mathcal{E}$ that are placed in at least one block of $\mathcal{B}^{\mathcal{E}}$ (i.e., $\forall v_i \in V_{\mathcal{B}} : \exists p_i \in \mathcal{E} \wedge b_j \in \mathcal{B}^{\mathcal{E}} \wedge p_i \in b_j$), while $E_{\mathcal{B}}$ contains undirected edges between all pairs of co-occurring entities (i.e., $\forall e_{i,j} = \langle p_i, p_j \rangle \in E^{\mathcal{B}} : p_i \neq p_j \wedge \exists b_k \in \mathcal{B}^{\mathcal{E}} \wedge p_i \in b_k \wedge p_j \in b_k$).* ■

The blocking graph over a *bilateral* block collection $\mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2}$ is defined in analogy. The only difference is that it results in a *bipartite graph*, since its set of nodes $V_{\mathcal{B}}$ is separated into two disjoint sets, $V^1_{\mathcal{B}}$ and $V^2_{\mathcal{B}}$, which comprise the entities stemming from the entity collections $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively (i.e., $V^1_{\mathcal{B}} \subseteq \mathcal{E}_1$ and $V^2_{\mathcal{B}} \subseteq \mathcal{E}_2$). More formally: $\forall v^k_i \in V^k_{\mathcal{B}} : \exists p_i \in \mathcal{E}_k \wedge b^{1,2}_j \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2} \wedge p_i \in b^k_j$, where $k \in \{1, 2\}$. Thus, the set of edges $E_{\mathcal{B}}$ contains only connections between entities stemming from different entity collections: $\forall e_{i,j} = \langle p_i, p_j \rangle \in E_{\mathcal{B}} : \exists b^{1,2}_k \in \mathcal{B}^{\mathcal{E}_1 \times \mathcal{E}_2} \wedge p_i \in b^1_k \wedge p_j \in b^2_k$.

Note that for reasons explained in Section 5.3, the edges of a blocking graph can be directed, as well. An edge pointing from entity p_i to p_j is represented by $e_{\bar{i},j}$ to distinguish it from the undirected edge $e_{i,j}$ that connects the same entities. A blocking graph with directed edges is called *directed blocking graph* and is symbolized as $\bar{\mathcal{G}}_{\mathcal{B}}$.

The purpose of the blocking graph is to facilitate efficiency improvements over the input block collection. An immediate contribution to this goal is the *elimination of redundant comparisons* without any impact on effectiveness (i.e., *PC*). Redundant comparisons are easily identified during the creation of the blocking graph, since the corresponding entities have already been connected with an edge. In such cases, we simply skip connecting them with an additional edge so that each pair of co-occurring entities is connected with at most one edge — regardless of the total number of comparisons between them in the blocks of $\mathcal{B}$. Consequently, every pair of adjacent entities is examined only once, thus improving efficiency without any impact on effectiveness, as the set of co-occurring entity pairs remains unchanged.

Additional efficiency enhancements can be achieved through the *pruning* of the blocking graph: edges between *non-matching* entities can be removed from the graph, discarding unnecessary comparisons without any impact on PC. This process is carried out according to a *pruning algorithm* and theoretically can result in a graph that exclusively contains edges between matching entities. Continuing our example, the blocking graph of Figure 5.1 (b) can be derived from that of Figure 5.1 (a) by discarding edges with a weight lower than 2, or by retaining the two edges with the highest weight. In any case, the remaining edges determine a new set of blocks that ideally places every set of matching entities in a separate block. The graph of Figure 5.1 (b), for instance, results in two blocks, $b_1 = \{p_1, p_3\}$ and $b_2 = \{p_2, p_4\}$, which achieve the same recall as the blocks of Figure 1.4, while involving just 2 comparisons.

In practice, though, we can only approximate this ideal case by exploiting the evidence that is encapsulated in the given block collection. How entities are assigned to blocks provides reliable indications for the similarity of co-occurring entities, which can be quantified by assigning a weight to the edge connecting them. In the context of redundancy-positive blocking methods, the more blocks two entities share, the more similar they are and the higher the weight of their adjacent edge should be. In this way, the pruning of the blocking graph becomes the process of removing edges with low weights on the grounds that they (are likely to) link dissimilar entities. In more detail, the weight $e_{i,j}.weight$ of an edge $e_{i,j}$ expresses the utility of the comparison between the profiles p_i and p_j; that is, it quantifies the trade-off between the cost $c_{i,j}$ of comparing the adjacent entities and the gain $g_{i,j}$ of executing this comparison (i.e., $e_{i,j}.weight = g_{i,j}/c_{i,j}$). The cost $c_{i,j}$ pertains to the number of comparisons required by the corresponding edge and is always equal to 1 (remember that, by definition, every edge in the blocking graph captures one comparison). Thus, the edge weight is always equal to the gain of the corresponding comparison; its value should be 0 if the compared entities are not matching and 1 if they are duplicates (i.e., $e_{i,j}.weight = 0 \leftrightarrow p_i \not\equiv p_j$ and $e_{i,j}.weight = 1 \leftrightarrow p_i \equiv p_j$).

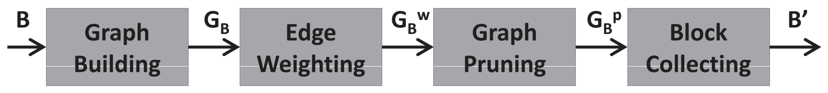


Figure 5.2: The internal functionality of our approach to meta-blocking.

However, it is not possible to estimate the real value of $g_{i,j}$ (and correspondingly of $e_{i,j}.weight$) without actually executing the comparison between p_i and p_j. For this reason, we use a *weighting scheme* that a-priori approximates the weight of each edge by considering the features of the blocking graph (e.g., the number of blocks shared by the adjacent entities and/or the corresponding individual cardinalities). In Section 5.2, we will present five such weighting schemes for redundancy-positive blocking methods, whereas Section 5.3 introduces four techniques for discarding edges with low weights through a *pruning criterion*, which bounds either the number or the weight of the retained edges.

Overall, our approach to meta-blocking, which was originally introduced in [PKPNar], involves four successive steps that are illustrated in Figure 5.2:

1. *Graph Building* receives a block collection $\mathcal{B}$ and derives the blocking graph $\mathcal{G}_{\mathcal{B}}$ from its block assignments. We elaborate on this process in Section 5.1.

2. *Edge Weighting* takes as input a blocking graph $\mathcal{G}_{\mathcal{B}}$ and turns it into the *weighted blocking graph* ($\mathcal{G}^w_{\mathcal{B}}$) by determining the weights of its edges. We introduce several weighting schemes for this procedure in Section 5.2.

3. *Graph Pruning* receives as input the weighted blocking graph and derives the *pruned blocking graph* ($\mathcal{G}^p_{\mathcal{B}}$) from it, by removing some of its edges. We delve into the pruning algorithms and the pruning criteria involved in this procedure in Section 5.3.

4. *Block Collecting* is given as input the pruned blocking graph $\mathcal{G}^p_{\mathcal{B}}$ and extracts from it a new block collection $\mathcal{B}'$, which actually constitutes the final output of the entire meta-blocking process. Note that the type of input blocks does not need to coincide with that of the output blocks (as we will see in Section 5.3, a set of unilateral blocks can be transformed into a set of bilateral ones, and vice versa). We analyze this step in Section 5.4.

The weighting scheme, the pruning algorithm and the pruning criterion can entail a *schema-dependent*, a *schema-agnostic*, or a *hybrid* functionality. In the

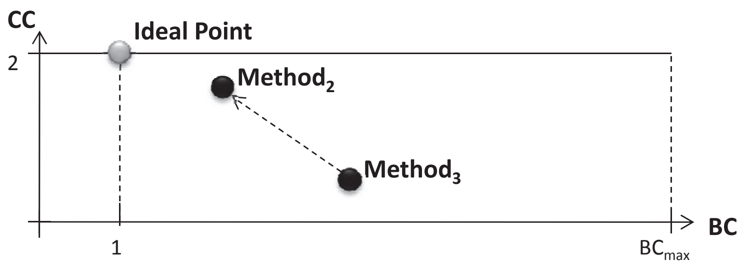


Figure 5.3: Illustration of the effect of meta-blocking and of block processing on the *BC-CC* mapping of a blocking collection.

following, we exclusively consider schema-agnostic techniques, since they are applicable to any *blocking settings*, i.e., any combination of a blocking scheme and a (pair of) entity collection(s). Their effect on the *BC-CC* mapping of the original block collection is depicted in Figure 5.3; they reduce *BC* and increase *CC* so that the new block collection moves closer to the Ideal Point (1,2). This is similar to the effect of block processing techniques, but our meta-blocking methods do not aim at substituting them. Instead, their goal is to improve the output of block building in order to facilitate the performance of block processing. This is why meta-blocking intervenes between these two procedures, as depicted in Figure 1.3.

5.1 Building the Blocking Graph

The process of extracting the blocking graph from a bilateral block collection $\mathcal{B}$ is outlined in Algorithm 5.1 (for unilateral blocks, the corresponding algorithm is simpler, and we omit it for brevity). For each block in $\mathcal{B}$, it considers every pair of co-occurring entities it contains (Lines 2-5); for bilateral blocks, this process requires that the considered entities belong to different inner blocks (i.e., $p_i \in b_i^1$ and $p_j \in b_i^2$). For each pair, we add the corresponding nodes to the initially empty blocking graph (Lines 4 and 6) and connect them with an edge (Line 7). The edge weights are specified after the structure of the blocking graph has been settled, because — as explained in the next section — it is possible for a blocking scheme to rely on it (Line 8). To restrict them to the interval $[0, 1]$, regardless of the input weighting scheme (cf. Definition 5.1), we normalize them by dividing with the maximum one (Line 9). The time complexity of this procedure, which is independent of the underlying blocking scheme, is equal to the aggregate cardinality of $\mathcal{B}$

Algorithm 5.1: Building the Blocking Graph.

```
   Input: (i) $\mathcal{B}$ a block collection,
          (ii) $WS$ a weighting scheme
   Output: $\mathcal{G}_{\mathcal{B}}$ the corresponding blocking graph
1  $V_{\mathcal{B}} \leftarrow \{\}$; $E_{\mathcal{B}} \leftarrow \{\}$;
2  foreach $b_i \in \mathcal{B}$ do
3      foreach $p_i \in b_i^1$ do
4          $V_{\mathcal{B}} \leftarrow V_{\mathcal{B}} \cup \{v_i\}$ ;
5          foreach $p_j \in b_i^2$ do
6              $V_{\mathcal{B}} \leftarrow V_{\mathcal{B}} \cup \{v_j\}$ ;
7              $E_{\mathcal{B}} \leftarrow E_{\mathcal{B}} \cup \{e_{i,j}\}$ ;
8  setWeights($WS$, $\mathcal{B}$, $V_{\mathcal{B}}$, $E_{\mathcal{B}}$);
9  normalizeWeights($E_{\mathcal{B}}$);
10 return $\mathcal{G}_{\mathcal{B}} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$;
```

(i.e., $O(||\mathcal{B}||)$).

Graph Materialization

The blocking graph constitutes a conceptual model that aims at facilitating the interpretation and the development of our meta-blocking techniques. In the context of large entity collections with millions of entities (nodes) and billions of comparisons (edges), its materialization poses significant technical challenges. For this reason, it can be *indirectly* implemented in two ways:

- through inverted indices, which associate each entity with the list of the blocks containing it, and
- with the help of bit arrays, which represent each entity as a vector with zero values in all places, but those corresponding to the blocks containing it; the latter places are valued 1.

Both approaches scale well in the context of HHIS and accommodate all the weighting schemes of Section 5.2.

Efficiency of Construction

Theoretically, the construction of the blocking graph has the same complexity as the naive block processing method that iterates over all pairs of co-occurring entities. In practice, though, meta-blocking exhibits a significantly lower running time, because it exclusively involves operations with integers, when implemented on the basis of inverted indices or bit arrays. Hence, the computation of edge weights is much faster than the comparison of entity profiles, which invariably relies on string similarity metrics that have a non-trivial complexity of their own. As an example, consider edit distance, one of the simplest string comparison techniques, whose complexity even for an optimized implementation is $O(n^2/\log n)$, when n is the length for both of the compared strings [MP80]. We analytically examine the time requirements of our meta-blocking approaches in Section 7.5.8.

5.2 Edge Weighting

We now introduce five schema-agnostic weighting schemes that rely exclusively on evidence drawn from the input block collection. We will use the following notations: $\mathcal{B}_i \subseteq \mathcal{B}$ denotes the set of blocks containing the entity p_i, $\mathcal{B}_{i,j} \subseteq \mathcal{B}$ is the set of blocks shared by the entities p_i and p_j (i.e., $\mathcal{B}_{i,j} = \mathcal{B}_i \cap \mathcal{B}_j$), and $|v_i|$ symbolizes the degree of node v_i (i.e., the number of edges connected to it). In the following, we analytically describe our weighting schemes and explain the rationale behind them.

- *Aggregate Reciprocal Comparisons Scheme* (*ARCS*). This scheme is based on the premise that the more entities a block contains, the less likely they are to be matching. The reason is that the information forming a large block is not distinctive enough to group highly similar entities. For instance, stop words usually cluster together a large part of the input entity collection in the context of Token Blocking. The *ARCS* scheme defines the similarity of two entities sharing a block as the reciprocal of the comparisons it entails (i.e., its individual cardinality). Hence, the aggregate similarity of two co-occurring entities, p_i and p_j, is defined as the sum of the reciprocal individual cardinalities of their common blocks. More formally, the weight of the edge $e_{i,j}$ connecting them is defined as follows:
$$e_{i,j}.weight = \sum_{b_k \in \mathcal{B}_{i,j}} \frac{1}{||b_k||}.$$

- *Common blocks Scheme* (*CBS*). A strong indication for the similarity of two entities is provided by the number of blocks they have in common; the more blocks they share, the more likely they are to match. Hence, the weight of an edge connecting entities p_i and p_j is set equal to:

$$e_{i,j}.weight = |\mathcal{B}_{i,j}|.$$

- *Enhanced Common blocks Scheme* (*ECBS*). This scheme improves on *CBS* by adding contextual information to its weights. Instead of merely considering the number of common blocks, it takes into account the total number of blocks that are associated with each one of the co-occurring entities. Inspired from the IDF metric of Information Retrieval, the fewer blocks an entity is placed in, the higher should be the weights of the edges associated with it. Thus, it sets the weight of each edge equal to:

$$e_{i,j}.weight = |\mathcal{B}_{i,j}| \cdot \log \frac{|\mathcal{B}|}{|\mathcal{B}_i|} \cdot \log \frac{|\mathcal{B}|}{|\mathcal{B}_j|}.$$

- *Jaccard Scheme* (*JS*). Similar to *ECBS*, this scheme aims at enhancing *CBS* by considering the total number of blocks associated with the co-occurring entities. To this end, it sets the weight of edge $e_{i,j}$ equal to the Jaccard similarity of the lists of blocks associated with its adjacent entities, p_i and p_j:

$$e_{i,j}.weight = \frac{|\mathcal{B}_{i,j}|}{|\mathcal{B}_i| + |\mathcal{B}_j| - |\mathcal{B}_{i,j}|}.$$

  In essence, this weight reveals the percentage of common blocks shared by the adjacent entities. It takes values in the interval $[0, 1]$, with 0 indicating the absence of common blocks and 1 corresponding to identical block lists.

- *Enhanced Jaccard Scheme* (*EJS*). This scheme improves on *JS* by adding contextual information to the Jaccard similarity of the associated block lists. Namely, it considers the total number of edges (i.e., comparisons) associated with each one of the adjacent nodes, so that the fewer edges are connected with a node, the higher should their individual weights be (a rationale similar to IDF). Thus, we have:

$$e_{i,j}.weight = \frac{|\mathcal{B}_{i,j}|}{|\mathcal{B}_i| + |\mathcal{B}_j| - |\mathcal{B}_{i,j}|} \cdot \log \frac{|E_{\mathcal{B}}|}{|v_i|} \cdot \log \frac{|E_{\mathcal{B}}|}{|v_j|}.$$

Note that all the above weighting schemes rely on the fundamental principle of redundancy-positive blocking methods that the similarity of block assignments provides a good representation of matching probability: the more blocks two entities have in common, the more similar their profiles are expected to be. We experimentally analyze their effect on the performance of meta-blocking in Section 7.5.4.

5.3 Pruning the Blocking Graph

This process is based on two essential components:

- the *pruning algorithm*, which specifies the procedure that will be followed in the processing of the blocking graph, and
- the *pruning criterion*, which determines the edges to be retained.

The combination of a pruning algorithm with a pruning criterion forms a *pruning scheme*. In the following, we introduce a series of pruning schemes that involve schema-agnostic pruning algorithms and criteria, thus being applicable to any blocking graph.

Pruning algorithms

In general, the pruning algorithms can be categorized in two classes:

- The **edge-centric algorithms** iterate over the *edges* of a blocking graph in order to select the *globally* best comparisons, by filtering out those that do not satisfy the pruning criterion.
- The **node-centric algorithms** iterate over the *nodes* of a blocking graph with the aim of selecting the *locally* best comparisons for each entity (i.e., the adjacent entities with the largest edge weights).

We analytically examine the relative performance of these two types of pruning algorithms in Section 7.5.2.

Pruning criteria

In general, they can be categorized in a two-dimensional taxonomy that is formed by the orthogonal, but complementary dimensions of functionality and scope.

The **functionality** of pruning criteria distinguishes them into:

- *weight thresholds*, which specify the minimum weight for the edges to be retained, and
- *cardinality thresholds*, which determine the maximum number of retained edges.

The **scope** of pruning criteria distinguishes them into:

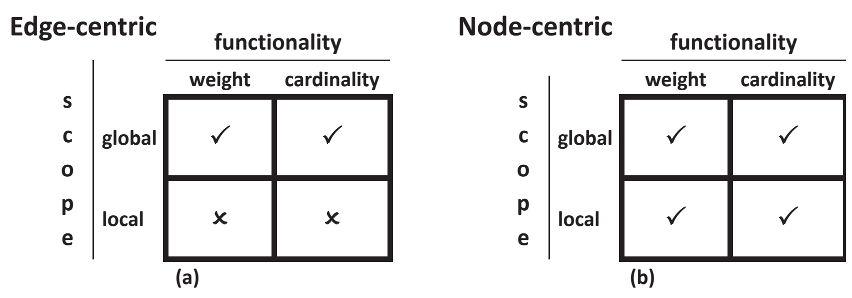


Figure 5.4: All possible combinations of pruning algorithms with pruning criteria.

- *global thresholds*, which define conditions that are applicable to the entire blocking graph (i.e., all the edges of the graph), and
- *local thresholds*, which specify conditions that apply only to a subset of it (i.e., the adjacent edges of a specific node).

Cardinality thresholds should be preferred in applications that have predefined temporal resources (i.e., available processing time), because they allow for a-priori determining the number of executed comparisons. In contrast, weight thresholds are convenient for applications that put more emphasis on controlling effectiveness, since the harshness of their pruning is analogous to their value. Both classes, though, are suitable for *incremental ER* (a.k.a., Pay-As-You-Go ER) [WMGMar], where the goal is to execute most of the matching comparisons in the first iterations, decreasing their number gradually, as ER progresses. For weight (cardinality) thresholds, this can be simply achieved by decreasing (increasing) its value in every iteration.

Pruning Schemes. The composition of pruning schemes is determined by the scope of pruning thresholds. In Figure 5.4, we illustrate all possible combinations of pruning algorithms with pruning criteria. Starting with the edge-centric algorithms, we observe that they can only be combined with global criteria — regardless of their functionality. The reason is that it is impossible to employ a local threshold, when trying to select the top weighted edges across the entire blocking graph. The combination of edge-centric algorithms with global weight thresholds (i.e., WEP) is analyzed in Section 5.3.1 and their coupling with global cardinality thresholds (i.e., CEP) in Section 5.3.2.

By definition, the node-centric algorithms are compatible with local thresholds — regardless of their functionality. However, they can be combined with global thresholds, as well. Their combination with a global weight threshold is actu-

Algorithm 5.2: Weight Edge Pruning.

```
Input: (i) $\mathcal{G}_{\mathcal{B}}^{in}$ the blocking graph, and
       (ii) $w_{min}$ the global weight criterion.
Output: $\mathcal{G}_{\mathcal{B}}^{out}$ the undirected pruned blocking graph
1 foreach $e_{i,j} \in E_{\mathcal{B}}$ do
2     if $e_{i,j}.weight < w_{min}$ then
3         $E_{\mathcal{B}} \leftarrow E_{\mathcal{B}}$ - $\{e_{i,j}\}$ ;
4 return $\mathcal{G}_{\mathcal{B}}^{out} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$;
```

ally identical to WEP, as they both retain the edges that are weighted higher than the given threshold. Their coupling with a global cardinality threshold retains the same number of adjacent edges among all nodes (e.g., the 2 top-weighted edges per node). In contrast, their combination with a local cardinality threshold derives the number of retained edges for each node from its degree (e.g., $|v_i|/10$ of the top weighted edges for every node v_i); this approach is substantially different from CEP, which keeps the top weighted edges across the entire blocking graph. The pruning schemes that combine node-centric algorithms with local weight thresholds (i.e., WNP) are examined in Section 5.3.3, while those coupling them with cardinality thresholds — of any scope — (i.e., CNP) are examined in Section 5.3.4.

Before elaborating on the functionality of the pruning schemes, it should be stressed that the node-centric algorithms yield a directed, pruned blocking graph, unlike the edge-centric algorithms, which produce an undirected one.

5.3.1 Weight Edge Pruning (WEP)

This scheme consists of the edge-centric algorithm coupled with a global weight threshold (i.e., the minimum edge weight). Its functionality is outlined in Algorithm 5.2. It iterates over all edges (Line 1) and discards (Line 3) those having a weight lower than the input threshold (Line 2). The remaining edges form the pruned blocking graph of the output. The time complexity of this algorithm is equal to the aggregate cardinality of the original block collection (i.e., $O(||\mathcal{B}||)$).

The most critical part of this algorithm is the selection of the minimum edge weight w_{min}. Its precise value depends on the underlying weighting scheme and the resulting distribution of edge weights, in particular. In general, though, the matching entities are expected to be connected with edges of higher weights than the non-matching ones. Thus, the goal is to identify the break-even point that

Algorithm 5.3: Cardinality Edge Pruning.

```
Input: (i) $\mathcal{G}_{\mathcal{B}}^{in}$ the blocking graph, and
       (ii) $K$ the global cardinality criterion
       blocking graph.
Output: $\mathcal{G}_{\mathcal{B}}^{out}$ the undirected pruned blocking graph
1 $SortedStack \leftarrow \{\}$;
2 foreach $e_{i,j} \in E_{\mathcal{B}}$ do
3     $SortedStack$.push($e_{i,j}$);
4     if $K < SortedStack.size()$ then
5         $SortedStack$.pop();
6 foreach $e_{i,j} \in E_{\mathcal{B}}$ do
7     if $SortedStack.contains(e_{i,j}) = false$ then
8         $E_{\mathcal{B}} \leftarrow E_{\mathcal{B}}$ - $\{e_{i,j}\}$ ;
9 return $\mathcal{G}_{\mathcal{B}}^{out} = \{V_{\mathcal{B}}, E_{\mathcal{B}}, WS\}$;
```

distinguishes the former type of edges from the latter one. Experimental evidence with real-world data sets suggests that the average edge weight provides an efficient (i.e., requires just one iteration over all edges) as well as reliable (i.e., low impact on effectiveness) estimation of this break-even point — regardless of the underlying weighting scheme (see Section 7.5.1 for more details).

5.3.2 Cardinality Edge Pruning (CEP) or Top-K Edges

This scheme combines the edge-centric pruning algorithm with a global cardinality threshold K that specifies the total number of edges retained in the pruned graph. Its goal is to retain the K edges with the maximum weight. Its functionality is outlined in Algorithm 5.3. At its core lies a *sorted stack* that stores the edges in descending order of weight so that the edge with the lowest one is efficiently removed (i.e., pop), when the maximum capacity K is exceeded. The algorithm iterates over all edges of the input blocking graph twice: the first iteration (Lines 2-5) identifies the top-K edges and stores them in the sorted stack, while the second iteration (Line 6-8) removes from the graph those edges that are not contained in the sorted stack. Hence, its time complexity is equal to the aggregate cardinality of the original block collection (i.e., $O(||\mathcal{B}||)$).

To specify the optimal value for K, we employ a technique that relies on the BC-CC mapping of the initial blocking graph and its pruned version (cf. Section 3.3). The goal is to map the latter closer to the Ideal Point (1,2) than the

Algorithm 5.4: Weight Node Pruning.

```
   Input: (i) $\mathcal{G}_{\mathcal{B}}^{in}$ the blocking graph,
          (ii) $wt$ the local weight criterion
   Output: $\mathcal{G}_{\mathcal{B}}^{out}$ the directed pruned blocking graph
1  $E_{\mathcal{B}}^{out} \leftarrow \{\}$;
2  foreach $v_i \in V_{\mathcal{B}}$ do
3      $\mathcal{G}_{v_i} \leftarrow$ getNeighborhood($v_i$, $\mathcal{G}_{\mathcal{B}}$);
4      $t_{v_i} \leftarrow wt(\mathcal{G}_{v_i})$;
5      foreach $e_{i,j} \in E_{v_i}$ do
6          if $t_{v_i} \leq e_{i,j}.weight$ then
7              $E_{\mathcal{B}}^{out} \leftarrow E_{\mathcal{B}}^{out} \cup \{\bar{e_{i,j}}\}$ ;
8  return $\mathcal{G}_{\mathcal{B}}^{out} = \{V_{\mathcal{B}}, E_{\mathcal{B}}^{out}, WS\}$;
```

former. Given that the pruned graph results in a bilateral block collection with K blocks of size 2 (cf. Section 5.4), its CC takes the maximum value (i.e., CC_{out}=2)[1], while its BC is equal to $BC_{out} = 2K/|\mathcal{E}|$. Thus, CC_{out} is greater than or equal to CC_{in} of the input blocking graph in all cases and, for an improved BC-CC mapping, it suffices to have:

$$BC_{out} \leq BC_{in} \Leftrightarrow \frac{2K}{|\mathcal{E}|} \leq BC_{in} \Leftrightarrow K \leq \lfloor \frac{BC_{in} \cdot |\mathcal{E}|}{2} \rfloor,$$

where BC_{in} stands for the BC value of the input blocking graph. Therefore, the maximum meaningful value for K is specified with respect to the level of redundancy of the input block collection. In cases where $CC_{in} \ll CC_{out}$, we can set $K = \lfloor BC_{in} \cdot |\mathcal{E}|/2 \rfloor$ in order to ensure higher redundancy and, thus, higher PC. Although this approach maintains the same levels of redundancy (i.e., the same number of block assignments), efficiency is significantly improved; unlike the input block collection, which contains blocks of various sizes, the output one exclusively comprises blocks of minimum size (i.e., two entities per block). This means that CEP minimizes the number of pairwise comparisons for a specific level of redundancy.

[1] CC expresses the ratio of block assignments over comparisons (Definition 3.9). Given that the output of CEP involves only blocks of size 2, there are two block assignments for every comparison, thus leading to $CC_{out} = CC_{max} = 2$.

5.3.3 Weight Node Pruning (WNP)

This scheme combines the node-centric pruning algorithm with a local weight threshold. In essence, it applies WEP to the neighborhood of each node v_i, i.e., the sub-graph $\mathcal{G}_{v_i}$ that comprises the nodes of $\mathcal{G}_{\mathcal{B}}$ connected with v_i along with the edges connecting them; the former are denoted by V_{v_i} and the latter by by E_{v_i}. Its functionality, though, differs from WEP in two aspects:

- it employs a different threshold for every neighborhood, and
- it replaces the retained, undirected edges with directed ones that point from v_i to a neighboring node.

Algorithm 5.4 presents the pseudo-code for this procedure. It iterates over all nodes of the input blocking graph (Line 2) and extracts the corresponding neighborhood $\mathcal{G}_{v_i}$ (Line 3). Then, it specifies the minimum edge weight for $\mathcal{G}_{v_i}$ according to the given threshold criterion (Line 4). It iterates over all edges of E_{v_i} (Line 5) and for each undirected edge exceeding the specified local threshold, it adds one directed edge to the pruned graph (Lines 6-7). In the worst case, the input blocking graph is a complete one, thus accounting for a time complexity of $O(|V_{\mathcal{B}}|\cdot|E_{\mathcal{B}}|)$. In practice, though, the underlying blocking scheme ensures that not all nodes are connected with each other, thus yielding a significantly lower time complexity.

To specify the optimal threshold for each neighborhood, we rely on the same rationale as WEP: all weighting schemes of Section 5.2 assign high values to edges connecting matching entities and low values to edges connecting non-matching nodes. Regardless of the selected scheme, the corresponding break-even point can be approximated by the mean weight of the edges in each neighborhood $\mathcal{G}_{v_i}$.

5.3.4 Cardinality Node Pruning (CNP) or k-Nearest Entities

At the core of this scheme lies the node-centric pruning algorithm in conjunction with a local cardinality threshold. Its goal is to select for each node v_i, the k neighboring nodes that are connected with the top edge weights (i.e., k-nearest entities). To this end, it applies the CEP algorithm to the neighborhood $\mathcal{G}_{v_i}$ of v_i. This process is outlined in Algorithm 5.5. It iterates over all entities of the input blocking graph (Line 2), extracting their neighborhood (Line 4) and setting the maximum number of entities retained in each case (Line 5). Subsequently, it iterates over the edges of the current neighborhood and places them into the sorted stack (Line 6-9). For each of the selected undirected edges, a new, directed one is added to the pruned blocking graph of the output (Lines 10-12). In the worst case, the time

Algorithm 5.5: Cardinality Node Pruning.

```
Input: (i) $\mathcal{G}^{in}_{\mathcal{B}}$ the blocking graph,
       (ii) $ct$ the local cardinality criterion
Output: $\mathcal{G}^{out}_{\mathcal{B}}$ the directed pruned blocking graph
1  $E^{out}_{\mathcal{B}} \leftarrow \{\}$;
2  foreach $v_i \in V_{\mathcal{B}}$ do
3      $SortedStack_{v_i} \leftarrow \{\}$;
4      $\mathcal{G}_{v_i} \leftarrow$ getNeighborhood($v_i$, $\mathcal{G}_{\mathcal{B}}$);
5      $k \leftarrow ct(\mathcal{G}_{v_i})$;
6      foreach $e_{i,j} \in E_{v_i}$ do
7          $SortedStack_{v_i}$.push($e_{i,j}$);
8          if $k < SortedStack_{v_i}.size()$ then
9              $SortedStack_{v_i}$.pop();
10     foreach $e_{i,j} \in E_{v_i}$ do
11         if $SortedStack.contains(e_{i,j}) = true$ then
12             $E^{out}_{\mathcal{B}} \leftarrow E^{out}_{\mathcal{B}} \cup \{\bar{e_{i,j}}\}$ ;
13 return $\mathcal{G}^{out}_{\mathcal{B}} = \{V_{\mathcal{B}}, E^{out}_{\mathcal{B}}, WS\}$;
```

complexity of this algorithm is the same as that of *WNP* (i.e., $O(|V_{\mathcal{B}}|\cdot|E_{\mathcal{B}}|)$), but in practice, it is significantly lower.

In general, the cardinality threshold for each neighborhood can depend on its size (e.g., $k_i = \lceil 0.1 \cdot |E_{v_i}|\rceil$). For simplicity, though, we assume in the following that k takes the same value for each neighborhood. To identify its optimal value, we rely on the *BC*-*CC* mapping of the input and the output blocking graph. Again, the goal is to ensure that the latter is closer to (1,2) than the former. Given that the block collection contains bilateral blocks with inner block sizes of 1 and k (cf. Section 5.4), the *CC* of the pruned graph is equal to $CC_{out} = \frac{k+1}{k}$, while its *BC* is equal to $BC_{out} = k + 1$. Thus, k can be specified with respect to the *CC* and the *BC* of the input block collection: $1/(1 - CC_{in}) \leq k \leq BC_{in} - 1$. In cases where $CC_{in} \ll 1$, we can safely set $k = \lfloor BC_{in} - 1 \rfloor$, ensuring significantly higher efficiency ($CC_{out} > 1$) at equal levels of redundancy and, thus, *PC*.

5.4 Collecting the new blocks

The last step of our meta-blocking approach transforms the pruned blocking graph into the new block collection that is returned as output. This process depends on the type of the blocking graph.

For the undirected pruned blocking graphs, which are produced by the edge-centric pruning algorithms, block collecting is straightforward: every retained edge lays the basis for creating a bilateral block of minimum size that contains the adjacent entities. As a result, the new block collection is redundancy-free (i.e., non-overlapping blocks). For example, the pruned blocking graph of Figure 5.1(b) is transformed in the blocks $b_1 = \{\{p_1\}, \{p_3\}\}$ and $b_2 = \{\{p_2\}, \{p_4\}\}$.

For the directed pruned blocking graphs, which are derived from the node-centric pruning algorithms, block collecting creates a bilateral block for each node v_i. Its inner blocks have the following property: one of them contains the entity that is mapped to v_i, while the other contains the entities connected with v_i through the retained, outgoing edges. For instance, the pruned blocking graph of Figure 5.1(c)[2] is transformed into the blocks $b_1 = \{\{p_1\}, \{p_3, p_4\}\}$ and $b_2 = \{\{p_2\}, \{p_3, p_4\}\}$. In this way, the new block collection is redundancy-bearing, since it is possible for two retained edges with different direction to connect the same entities. This means that its efficiency can be further enhanced with redundancy-reduction techniques.

5.5 Summary

In this chapter, we introduced meta-blocking as a generic task that can be applied on top of any redundancy-positive block building method in order to increase its efficiency at a minor cost in effectiveness. We also described a family of techniques that rely on the blocking graph in order to identify and discard comparisons between entities that are highly unlikely to match. We proposed five schema-agnostic weighting schemes for extracting the blocking graph from the input block collection, based on the block overlap between the adjacent entities of every edge. We distinguished the pruning algorithms that can be applied on top of them into two orthogonal categories and defined a two-dimensional taxonomy of the relevant pruning criteria. We also identified all the generic pruning settings that can be derived from the four possible combinations of pruning algorithms with pruning criteria. Their relative performance is analytically examined through a series of experiments in Section 7.5.

[2]For clarity we have excluded the outgoing edges of nodes p_3 and p_4.

Chapter 6

Block Processing

The methods of this category receive as input a block collection $\mathcal{B}$ and examine analytically its elements in order to identify the set of matching entities it contains. Their goal is actually to detect most of the existing pairs of duplicates at the minimum computational cost (i.e., number of executed comparisons). This means that their effect on the *BC*-*CC* mapping of a block collection is similar to that of meta-blocking: as depicted in Figure 5.3, *CC* increases, maximizing efficiency, while *BC* remains relatively stable, maintaining the original levels of effectiveness. On the whole, the mapping moves closer to the Ideal Point (1,2).

To facilitate the understanding and use of our block processing techniques, we introduce a two-dimensional taxonomy in Section 6.1. We then introduce the individual block processing strategies, grouped according to their granularity of functionality: we elaborate on the Block-refinement techniques in Section 6.2 and on the Comparison-refinement ones in Section 6.3. Note that all strategies apply transparently to both types of block collections, thus handling Dirty and Clean-Clean ER uniformly, without any significant modifications. Some of these techniques are complementary and can be combined into comprehensive ER solutions. Section 6.4 presents the rules that guide this process on the basis of the two-dimensional taxonomy of Section 6.1.

6.1 Classification of Block Processing Techniques[1]

Block processing techniques aim at reducing the number of executed pairwise comparisons without any significant impact on the detected number of duplicates. This

[1]Originally introduced in [PIP$^+$ar].

is usually accomplished by skipping the execution of individual blocks or comparisons. To grasp their functionality, we first need to clarify the type of comparisons they target. Given a block $b_k \in \mathcal{B}$, the comparison between $p_i \in b_k$ and $p_j \in b_k$ belongs to one of the following types:

- *Matching comparison*, if $p_i \equiv p_j$.
- *Repeated comparison*, if p_i and p_j have already been compared in a previously examined block.
- *Superfluous comparison*, if p_i or p_j or both of them have been matched to some other entity profile in the context of Clean-Clean ER and, thus, they cannot be duplicates. For Dirty ER, the comparison between p_i and p_j is superfluous if they involved entities have already been identified as matches.
- *Non-matching comparison*, if $p_i \not\equiv p_j$ and the comparison is neither repeated nor superfluous.

In this context, a block processing technique could aim at one or more of the following targets:

- to eliminate all repeated comparisons,
- to discard all superfluous comparisons, and
- to restrict the execution of non-matching comparisons.

Apparently, the first two targets can be achieved without any impact on the identified matches (i.e., effectiveness). This does not apply, though, to the third target: there is no safe way of determining whether two entities are duplicates or not, without actually comparing their profiles. Therefore, the methods that try to identify non-matching comparisons constitute approximate techniques with an inherent risk to miss part of the duplicates (i.e., they incur lower levels of effectiveness).

In practice, each block processing technique usually targets a specific type of comparisons. Thus, we distinguish them in the following categories:

- *Repeat methods* aim at discarding repeated comparisons without affecting effectiveness.
- *Superfluity methods* try to skip superfluous comparisons without any impact on effectiveness.
- *Non-match methods* target non-matching comparisons at a limited and controllable cost in effectiveness.
- *Scheduling methods* enhance efficiency in a indirect way, specifying the processing order that boosts the effect of other categories.

Granularity	Comparison's Type			
	Repeat Method	**Superfluity Method**	**Non-match method**	**Scheduling method**
Block-refinement	-	-	1. Block Purging 2. Block Pruning	1. Block Scheduling 2. (Sorted Neighborhood)
Comparison-refinement	1. Iterative Blocking 2. Comparison Propagation	Duplicate Propagation	Comparison Pruning	Comparison Scheduling

Figure 6.1: The two-dimensional taxonomy of block processing techniques.

In general, the block processing techniques that belong to different categories are complementary and can be combined into an ER workflow. Instead, techniques that target the same type of comparison typically have conflicting functionalities (i.e., they serve exactly the same need). An additional parameter that facilitates the distinction between complementary and conflicting approaches is their *granularity of functionality*, which categorizes them in two classes:

- *Block-refinement methods* operate at the coarse level of entire blocks, deciding whether a whole block will be processed or not.
- *Comparison-refinement methods* operate at the finer level of individual comparisons, deciding whether a single comparison will be executed or not.

In principle, block-refinement methods exhibit limited accuracy when discarding comparisons, but involve low time and space complexity. They offer the best choice for applications with limited resources and for those involving entity comparisons of low computational cost (e.g., in the case of relatively small entity profiles). In contrast, comparison-refinement techniques are more precise in the identification of unnecessary comparisons, but their higher accuracy comes at the cost of higher time and space complexity. They are suitable for applications that emphasize on effectiveness and have ample resources at their disposal as well as for applications that involve time-consuming entity comparisons (e.g., in the case of large entity profiles).

On the whole, the comparisons' type and the granularity of functionality define a two-dimensional taxonomy of efficiency methods that facilitates the combination of blocking methods into comprehensive ER workflows. Its outline is illustrated in Figure 6.1, along with a complete list of the existing techniques and those introduced in the following sections.

Algorithm 6.1: Computing the Purging Threshold.

```
Input: Set of blocks: $\mathcal{B}$
Output: Purging threshold: $maxICardinality$
1  $\mathcal{B}' \leftarrow orderByICardinality(\mathcal{B})$;
2  $blockAssignments \leftarrow 0$;   $index \leftarrow 0$;
3  $totalComparisons \leftarrow 0$;   $lastICardinality \leftarrow 1$;   $stats[] \leftarrow \{\}$;
4  foreach $b_i \in \mathcal{B}'$ do
5      if $lastICardinality < ||b_i||$ then
6          $stats[index].iCardinality = lastICardinality$;
7          $stats[index].cc = \frac{blockAssignments}{totalComparisons}$;
8          $index{+}{+}$;
9          $lastICardinality = ||b_i||$;
10     $blockAssignments \mathrel{+}= |b_i|$;   $totalComparisons \mathrel{+}= ||b_i||$;
11 $stats[index].iCardinality = lastICardinality$;
12 $stats[index].cc = \frac{blockAssignments}{totalComparisons}$;
13 $maxICardinality = lastICardinality$;
14 for $i \leftarrow stats.size()\text{-}1$ to $1$ do
15     if $stats[i].cc{=}stats[i-1].cc$ then
16         $maxICardinality{=}stats[i].iCardinality$;
17         break;
18 return $maxICardinality$;
```

6.2 Block-refinement Methods

6.2.1 Block Purging

The goal of this technique, which was originally introduced in [PINF11], is to discard non-matching comparisons by removing *oversized blocks*. These are blocks that contain an excessively high number of comparisons, although they are highly unlikely to contain *non-redundant duplicates* (i.e., pairs of matching entities that have no other, smaller block in common). Such blocks have a negative impact on efficiency (i.e., they decrease *PQ* and *RR*), although they have a negligible contribution to *PC*. Therefore, the gist of Block Purging is to specify a conservative upper limit on the individual cardinality of the processed blocks so that the oversized ones are discarded without any significant impact on *PC*. This limit is called *purging threshold*.

At the core of our approach lies the following observation: assuming that the blocks are sorted in descending order of individual cardinality, the value of *CC* increases when moving from the top block to the ones in the lower ranking positions. The reason is that its denominator (i.e., aggregate cardinality) decreases faster than its numerator (i.e., number of block assignments). The purging threshold is specified as the first individual cardinality that has the same CC value with the next (smaller) one; discarding blocks with fewer comparisons can only reduce BC (and, thus, PC), while having a negligible effect — if any — on PQ.

The outline of this approach is presented in Alg. 6.1. Line 1 orders the given block collection $\mathcal{B}$ in ascending order of individual cardinality, thus making it possible to calculate the CC for each distinct cardinality with a single pass (Lines 4-10). Lines 11-12 ensure that the last block is also considered in the computation of the statistics. Starting from the largest individual cardinality, the CC values of consecutive ones are then compared (Lines 14-17). The procedure stops as soon as the value of CC remains stable (Lines 15-17).

Apparently, the time complexity of this algorithm is dominated by the initial sorting and is equivalent to $O(|\mathcal{B}|{\cdot}log|\mathcal{B}|)$. Its space complexity is dominated by the array that stores the statistics for each individual cardinality and is equal to $O(|\mathcal{B}|)$.

6.2.2 Block Scheduling

This technique, which was coined in [PINF11], aims at sorting the input block collection $\mathcal{B}$ so that its processing makes the most of Block Prunning (cf. Section 6.2.3) and Duplicate Propagation (cf. Section 6.3.2). The rationale behind it is that the earlier a pair of matching entities is detected, the more superfluous comparisons will be saved in the subsequently processed blocks. This is particularly true for Clean-Clean ER, where the identified duplicate entities do not match with any other entity and all comparisons involving them can be safely discarded. However, this is not true for Dirty ER. In this case, the effect of Block Scheduling is reinforced when combined with Block Pruning, rather than with Duplicate Propagation. In the latter case, Block Scheduling merely discards multiple comparisons between the same, matching entities, a practice that is necessary only in the absence of Comparison Propagation (see below, Section 6.3.1).

In more detail, Block Scheduling associates every block $b_i \in \mathcal{B}$ with a *block utility* value, $u(b_i)$, that expresses the trade-off between the cost of processing it, $cost(b_i)$, and the corresponding gain, $gain(b_i)$. The former corresponds to the number of comparisons entailed in b_i (i.e., $cost(b_i) = ||b_i||$), while the latter pertains to the number of superfluous comparisons that are spared in the subsequently examined blocks — due to the propagation of detected duplicates. The actual value of the block utility $u(b_i)$ for a bilateral block b_i (i.e., Clean-Clean ER) has been estimated through a probabilistic analysis to be equal to [PINF11]:

$$u(b_i) = \frac{gain(b_i)}{cost(b_i)} \approx \frac{1}{max(|b_{i,1}|, |b_{i,2}|)}.$$

For a unilateral block b_i (i.e., Dirty ER), the block utility can be simply set equal to:

$$u(b_i) = \frac{1}{||b_i||} \text{ or, equivalently, } u(b_i) = \frac{1}{|b_i|}.$$

To incorporate this measure in the processing of blocks, we employ a *ranking function* $r : \mathcal{B} \mapsto \Re$ that defines a partial order on $\mathcal{B}$, sorting its elements in descending order according to the following implication: $u(b_i) \leq u(b_j) \Rightarrow r(b_i) \geq r(b_j)$. Therefore, its time complexity is equal to $O(|\mathcal{B}| \cdot log|\mathcal{B}|)$, while its space complexity is $O(|\mathcal{B}|)$.

6.2.3 Block Pruning

This method, which was originally presented in [PINF11], constitutes a coarse-grained approach that builds upon Block Scheduling in order to save non-matching comparisons. Instead of examining the resulting block collection in its entirety, it terminates the ER process prematurely, at a point that ensures a good trade-off between *PC* and *PQ*/*RR*.

In more detail, the block processing order defined by Block Scheduling ensures that the blocks placed at the top ranking positions offer high expected gain at a low cost. That is, they contain a relatively high number of duplicates, while involving a low number of comparisons. Block Scheduling actually ensures that the lower the ranking position of a block is, the fewer duplicates it contains and the more non-matching comparisons it involves. Therefore, blocks placed at the lowest ranking positions are unlikely to contain new, yet unidentified duplicates. This means that there is a break-even point where the possibility of finding additional matches is no longer worth the cost; blocks lying after this point can be excluded from the ER process to enhance its efficiency at a negligible cost in the missed matches (i.e., small decrease in *PC*).

Block Pruning aims at approximating this point in order to discard blocks dominated by non-matching comparisons. To this end, it keeps track of the evolution of *duplicate overhead* (*dh*), a measure that assesses the (average) number of comparisons that were performed in order to detect the latest match(es). Its value after processing the k-th block **containing duplicates** is defined as:

$$dh_k = \frac{|\mathcal{C}_{k-1}|}{|\mathcal{D}_k|},$$

where $|\mathcal{C}_{k-1}|$ represents the number of comparisons performed after processing the $k-1$-th block **with duplicates**, and $|\mathcal{D}_k|$ stands for the number of **new** matches identified within the latest block (i.e., $|\mathcal{D}_k| \geq 1$).

Duplicate overhead takes low values (close to 1) for the blocks placed at the top ranking positions, since every new pair of duplicates requires a small number of comparisons. The value of dh increases for duplicates discovered in blocks of lower ranking positions. As soon as it exceeds the predefined threshold of *maximum duplicate overhead* (dh_{max}), the entire ER process is terminated. This indicates that the cost of detecting new duplicates is excessively high and the few remaining matches are not worth it. Although this threshold can be adapted to the performance requirements of the application at hand, a value that provides a good estimation of the break-even point was experimentally derived from the following formula [PINF11]:

$$dh_{max} = 10^{log(||\mathcal{B}||/2)}. \tag{6.1}$$

The intuition behind Formula 6.1 is that the number of comparisons required for detecting the latest match is considered too large, when it reaches half the order of magnitude of all possible comparisons in the considered blocks.

Given that Block Pruning requires a single pass over the ordered input block collection, its time complexity is equal to $O(|\mathcal{B}|)$, where $|\mathcal{B}|$ is the number of blocks remaining after Block Purging.

6.3 Comparison-refinement Methods

6.3.1 Comparison Propagation

This method, which was coined in [PIN$^+$11a], constitutes a general technique for discarding all repeated comparisons from unilateral and bilateral block collections, without any impact on PC. In essence, it propagates all executed comparisons in an indirect way, thus avoiding the need to explicitly store them. Its functionality relies on two pillars:

- The process of *Block Enumeration* is a preparatory step that assigns a unique index to each block, indicating its processing order. As a result, b_i symbolizes

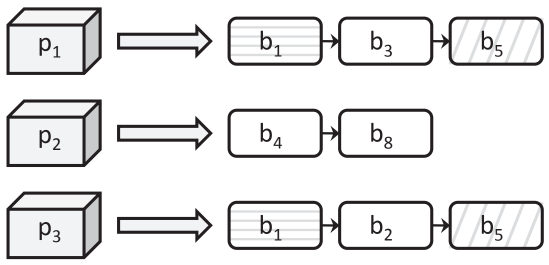


Figure 6.2: The Entity Index employed by Comparison Propagation.

the block placed in the *i*-th position of the processing list.

- The data structure of *Entity Index* (*EI*) points from the input entities to the blocks containing them. It is actually a *hash table*, whose *keys* correspond to entity ids, while each *value* lists the indices of the blocks that contain the corresponding entity. As an example, consider the *EI* of Figure 6.2, where we can see that entity p_1 is placed in blocks b_1, b_3 and b_5.

A comparison between p_i and p_j is recognized as repeated if the *Least Common Block Index* condition (*LeCoBI* for short) does not hold. This condition ensures that the current block is the first to contain both entities, returning `true` only if their lowest common block index is equal to the current block's index. Otherwise, if the least common index is lower than the current one, the entities have already been compared in another block, and the comparison should be discarded as redundant. As an example, consider the entities p_1 and p_3 in Figure 6.2. They share two blocks (b_1 and b_5) and, thus, their least common block index is 1. This means that the *LeCoBI* condition is satisfied in b_1, but not in b_5, thus saving the repeated comparison of p_1 and p_3 in the latter block.

The examination of the *LeCoBI* condition is linear with respect to the total number of blocks associated with a pair of entities. This is achieved by iterating once and in parallel over the two lists of block indices, after sorting them individually in ascending order. For higher efficiency, this sorting is executed only once, during the construction of the *EI*.

The time complexity for building the data structure of *EI* is linear with respect to the number of given blocks and the entities contained in them; in the average case, it is equal to $O(BC_{ov} \cdot |\mathcal{B}|)$. Its space complexity is linear with respect to the size of the input entity collections, depending, of course, on the overall level of redundancy. On average, it is equal to $O(BC_{ov} \cdot (|\mathcal{E}_1| + |\mathcal{E}_2|))$ for Clean-Clean ER and to $O(BC \cdot |\mathcal{E}|)$ for Dirty ER.

6.3.2 Duplicate Propagation

This method, which was originally presented in [PINF11], discards superfluous comparisons at no cost in PC. At its core lies a central data structure, called *Duplicate Index* (DI), which at any time contains all pairs of duplicate entities that have been identified so far. In the case of Dirty ER, these pairs are explicitly stored in DI[2], whereas for Clean-Clean ER, they are stored indirectly. In the latter case, it actually suffices to store the profile ids of all the entities that have already been matched to another one. Before comparing p_i and p_j, Duplicate Propagation checks DI in order to deem whether the comparison is superfluous. For Dirty ER, this requires that the match $p_i \equiv p_j$ is already contained in DI. For Clean-Clean ER, the comparison is superfluous if DI contains either of the entities and, thus, it is executed only if neither p_i nor p_j is contained in DI.

Apparently, the time complexity of this technique is constant (i.e., $O(c)$), as it merely requires a couple of look-ups in a hash-table. For high performance, though, it has to be coupled with a scheduling method of higher computational cost. Its space complexity depends on the size of DI and is equal to number of duplicates contained in the given block collection: $O(|D^{\mathcal{E}_1 \cap \mathcal{E}_2}|)$ for Clean-Clean ER and $O(|D^{\mathcal{E}}|)$ for Dirty ER.

6.3.3 Comparison Pruning

This technique, which was originally introduced in [PIN+11b], aims at discarding non-matching comparisons from redundancy-positive block collections at a controllable cost in effectiveness (i.e., PC). It can be conceived as an improved version of Block Pruning that operates on the level of individual comparisons, instead of considering entire blocks. It actually prunes a comparison if the involved entities are deemed highly unlikely to be a match in view of the overlap among their associated blocks.

In more detail, the overlap of two entities, p_i and p_j, is called *Entities Similarity* and is symbolized by $ES(p_i, p_j)$. Its value is defined from the Jaccard similarity of

[2]Ostensibly, this approach does not scale to the voluminous data collections of HHIS. In practice, though, the pairs of matching entities comprise a tiny portion of all pairwise comparisons.

the lists of block indices that are associated with them:

$$\begin{aligned} ES(p_i, p_j) &= \frac{|indices(p_i) \cap indices(p_j)|}{|indices(p_i) \cup indices(p_j)|} \\ &= \frac{|indices(p_i) \cap indices(p_j)|}{|indices(p_i)| + |indices(p_j)| - |indices(p_i) \cap indices(p_j)|} \end{aligned} \quad (6.2)$$

where $indices(p_k)$ denotes the set of block indices associated with the entity profile p_k. Formula 6.2 indicates that we only need to estimate the number of indices that are shared by p_i and p_j in order to compute $ES(p_i, p_j)$. As explained above, this process is facilitated by EI and is linear with respect to the total number of indices (i.e., it suffices to iterate over the two lists of indices just once and in parallel, due to their sorting in ascending order).

A pair of entities, p_i and p_j, is considered similar enough to justify the comparison of their profiles if $ES(p_i, p_j)$ exceeds the predefined threshold of the minimum allowed similarity value (ES_{min}). The actual value of this threshold depends on the redundancy of the individual entity collection(s). For a bilateral block collection (i.e., Clean-Clean ER), it is derived from the following formula:

$$ES_{min} = \frac{a \cdot min(BC_{\text{ind}}(\mathcal{E}_1), BC_{\text{ind}}(\mathcal{E}_2))}{BC_{\text{ind}}(\mathcal{E}_1) + BC_{\text{ind}}(\mathcal{E}_2) - a \cdot min(BC_{\text{ind}}(\mathcal{E}_1), BC_{\text{ind}}(\mathcal{E}_2))} \quad (6.3)$$

For a unilateral block collection (i.e., Dirty ER), ES_{min} is defined as:

$$ES_{min} = \frac{a \cdot BC}{(2 - a) \cdot BC} \quad (6.4)$$

In both cases, a takes values in the interval $(0, 1]$. Intuitively, these thresholds demand that two entities are analytically compared if their common blocks amount to $a \cdot 100\%$ of the (minimum individual) Blocking Cardinality. The performance of Comparison Pruning was experimentally found to be robust to the fluctuation of a, with higher values corresponding to stricter similarity conditions, and vice versa [PIN+11b].

Given that Comparison Pruning relies on the same data structures and operations as Comparison Propagation, it shares the same space and time complexity with it.

6.3.4 Comparison Scheduling

This technique, which was originally presented in [PIP+ar], aims at reducing the superfluous comparisons that are executed in order to increase efficiency at no cost in *PC*. Similar to Block Scheduling, it achieves its goal indirectly, by boosting the effect of Duplicate Propagation. However, it is more effective than Block Scheduling, due to the finer granularity of its functionality: instead of handling entire blocks, it considers individual comparisons, ordering them in such a way that the matching ones are executed first. Thus, more superfluous comparisons are saved in the subsequently processed blocks.

To this end, it first gathers the set of *valid comparisons* (C_v), which encompasses all pairwise comparisons of $\mathcal{B}$ that remain after filtering the initial set of blocks with a combination of the aforementioned efficiency methods (typically, Comparison Propagation and Comparison Pruning). Then, it associates each pairwise comparison $c_{i,j}$ between entities p_i and p_j with a *comparison utility* value, $u(c_{i,j})$. Similar to the block utility value, this measure is defined as:

$$u(c_{i,j}) = \frac{gain(c_{i,j})}{cost(c_{i,j})}.$$

The denominator corresponds to the cost of executing $c_{i,j}$ and is unary for all comparisons (i.e., $cost(c_{i,j}) = 1$). Thus, $u(c_{i,j}) = gain(c_{i,j})$, where $gain(c_{i,j})$ represents the likelihood that p_i and p_j are matching.

Several approaches are possible for estimating $gain(c_{i,j})$. In this work, we consider a best effort scoring mechanism that is derived from the following measures:

- The *Entities Similarity* $ES(p_i, p_j)$ is the same measure as that employed by Comparison Pruning, i.e., the portion of blocks shared by entities p_i and p_j. The higher its value is, the more likely are p_i and p_j to be matching. Hence, $u(c_{i,j})$ is proportional to $ES(p_i, p_j)$.
- The *Inverse Comparison Frequency* (ICF) of each entity. Inspired from the IDF metric of Information Retrieval, the $ICF(p_i)$ for an entity p_i is computed by dividing the size of C_v by that of its subset $C_v(p_i)$, which contains only those comparisons involving entity p_i (i.e., $C_v(p_i) = \{c_{i,k} \in C_v\}$). More formally:

$$ICF(p_i) = \log \frac{|C_v|}{|C_v(p_i)|}. \tag{6.5}$$

The rationale behind this metric is that the more *valid* comparisons are associated with an entity p_k, the less likely p_k is to match with one of its co-occurring entities. For this reason, the more comparisons entail p_k, the higher is the value of the denominator in Formula 6.5 and the lower is the value of $ICF(p_k)$. This means that the utility of comparison $c_{i,j}$ is proportional to both $ICF(p_i)$ and $ICF(p_j)$.

On the whole, the utility of a comparison $c_{i,j}$ is equal to[3]:

$$u(c_{i,j}) = ES(p_i, p_j) \cdot ICF(p_i) \cdot ICF(p_j).$$

To incorporate Comparison Scheduling in the ER process, we employ a *ranking function* $r : C_v \mapsto \mathfrak{R}$ that defines a partial order on C_v, sorting its elements in descending order according to the following implication: $u(c_{i,j}) \leq u(c_{k,l}) \Rightarrow r(c_{i,j}) \geq r(c_{k,l})$. Its time complexity is equal to $O(|C_v| \cdot log|C_v|)$, while its space complexity is $O(|C_v|)$.

6.4 Building ER Workflows[4]

Some of the above block processing techniques are complementary and can be combined with block building and meta-blocking methods in order to form a complete blocking-based ER solution, called *ER workflow*. Its composition typically depends on two factors:

- the *resources* that are available for handling the time and space requirements of the selected efficiency methods, and
- the *performance requirements* of the underlying application with respect to effectiveness and efficiency.

In this section, we introduce a general procedure for composing ER workflows, which covers a variety of performance and resource requirements for both Clean-Clean and Dirty ER over HHIS. In essence, this procedure explains how to combine all methods presented in Chapters 4 to 6 and consists of the six steps that are outlined in Figure 6.3. They are all optional, with the exception of the first one (i.e., the creation of blocks). We elaborate on each step in the following.

The first step selects the most suitable block building method for the application at hand. Given that the main techniques presented in Chapter 4 are conflicting, it

[3]Linear combinations of these three measures are also possible, but the resulting performance has been experimentally verified to be lower than that of the mere multiplication [PIP⁺ar].

[4]Originally introduced in [PIP⁺ar].

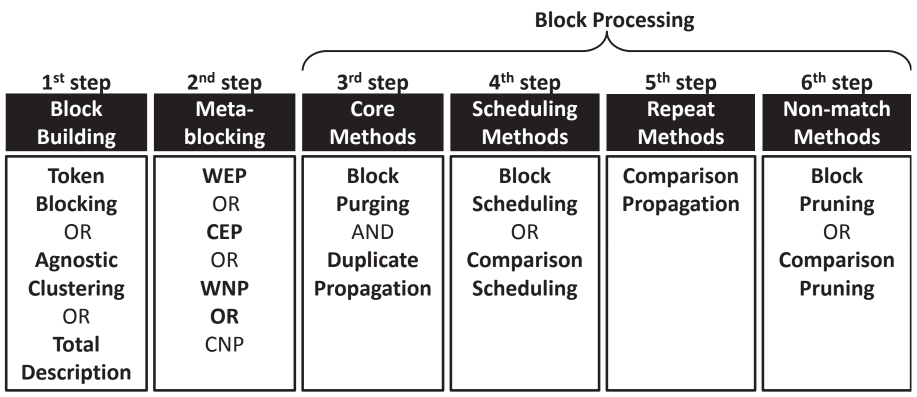


Figure 6.3: Procedure for creating an ER workflow.

suffices to include only one of those depicted in the left-most column of Figure 6.3.

All these block building techniques are redundancy-positive and, thus, can be combined with any meta-blocking method in order to achieve a better balance between effectiveness and efficiency. The methods of Chapter 5, though, are conflicting and, hence, only one of the four approaches can be incorporated in an ER workflow. Note that the choice of the meta-blocking approach is crucial, as it may determine the block processing techniques that will be added in the ER workflow, as well. For example, WEP and CEP do not need to be combined with Comparison Propagation and Comparison Pruning, as they transform the input block collection into a redundancy-free one. Note also that the high space and time complexity of meta-blocking render it incompatible with ER applications of limited resources (i.e., workflows that exclusively involve block-refinement methods). Instead, it is more appropriate for ER workflows that are dominated by comparison-refinement efficiency techniques.

The third step is to include the two core efficiency methods, i.e., Block Purging and Duplicate Propagation. They are indispensable for practically all ER workflows, since they significantly improve efficiency at a negligible cost in PC, while consuming minimal resources.

In the fourth step, we opt for the scheduling method that boosts the performance of Duplicate Propagation and Block Pruning (where applicable), by determining the processing order of the blocks or comparisons. Two are the valid options: Block Scheduling and Comparison Scheduling. The former is better integrated with block-refinement efficiency techniques, whereas the latter operates exclusively in conjunction with comparison-refinement ones. Hence, the scheduling method constitutes another critical part of an ER workflow, determining the overall granularity of its functionality and, consequently, its complexity and performance.

The fifth step incorporates the technique that eliminates all repeated comparisons, i.e., Comparison Propagation. As mentioned above, it is unnecessary for ER workflows involving WEP or CEP. In addition, it should be skipped in the case of ER workflows that can only afford minimal space requirements, due to the high space complexity it involves.

The last step determines the technique that — in addition to Block Purging — deals with non-matching comparisons. As mentioned above, though, the options can be restricted by the selected meta-blocking and scheduling methods; WEP and CEP can only be combined with Block Pruning, whereas workflows involving WNP or CNP can choose between Block Pruning and Comparison Pruning. Similarly, Block Scheduling is compatible with either of the two options, whereas workflows involving Comparison Scheduling can only opt for Comparison Pruning. In general, it is a good practice to add Comparison Propagation in workflows already encompassing Comparison Pruning, since both techniques share exactly the same space and time complexity and their functionality can be integrated in a single procedure.

Following these guidelines, an ER workflow can be simply created by specifying the methods that are included in it. Regardless of its composition, its methods are applied consecutively, in the order they are added, so that the output of the current one constitutes the input of the next one. There are only two exceptions to this rule (i.e., the execution order of a method deviates from the order it is added in the workflow):

- Block Purging is added at the third step, but should be executed right after block building. In this way, it substantially reduces the space and time complexity of the selected meta-blocking and block processing approaches.
- Comparison Scheduling is added at the fourth step, but is the last method to be executed in the workflows that involve it.

Duplicate Propagation constitutes a special case, since it is integrated into the entity comparison process.

6.5 Summary

At the core of this chapter lies the two-dimensional taxonomy of block processing techniques that was introduced in Section 6.1. Based on it, we elaborated on three block-refinement methods, explaining the type of comparisons they target and the low space and time requirements they involve. Next, we presented four comparison-refinement techniques, one for every type of comparisons, illuminating the way they improve on block refinement techniques. Having clarified the scope of each block processing technique, we provided guidelines for combining the complementary ones into ER workflows of higher performance. The procedure we introduced takes into account the available resources and the performance requirements of the underlying application. We experimentally investigate the relative performance of three major ER workflows in Section 7.6.

Chapter 7

Experimental Evaluation

The goal of this chapter is to experimentally evaluate the approaches presented in Chapters 4, 5 and 6 for the Problems 3.2, 3.3 and 3.4, respectively. We begin our analysis with the presentation of the data collections it comprises in Section 7.1. We then elaborate on the performance of our techniques independently for every problem: the block building methods are examined in Section 7.2, the meta-blocking ones in Section 7.5 and the block processing ones in Section 7.6. A separate section (i.e., Section 7.3) is devoted to Block Purging, because it conveys high efficiency enhancements at a very low computational cost and, thus, it should always be applied on top of block building (cf. Section 6.4). We also elaborate on the predictive accuracy of our *BC-CC* metric space in Section 7.4. We conclude our experimental study with a discussion of its outcomes in Section 7.7.

Note that all approaches and experiments were fully implemented in Java, version 1.6. We have publicly released their implementation through SourceForge.net[1]). For the implementation of the blocking functionality (i.e., inverted indices), we used the open source library of Lucene[2], version 2.9. The functionality of the n-gram graphs was provided by the open source library of JInsect[3]. For the implementation of the unconstrained EM clustering algorithm, we employed the open source library of Weka[4], version 3.6. All experiments were performed on a server with Intel Xeon E5472 3.0 GHz, 32GB of RAM memory, running Linux (SUSE SLES 10).

[1]http://sourceforge.net/projects/erframework
[2]http://lucene.apache.org/
[3]http://sourceforge.net/projects/jinsect
[4]http://www.cs.waikato.ac.nz/ml/weka/

	D_{movies}		$D_{infoboxes}$		D_{BTC09}
	DBPedia	**IMDB**	**DBPedia$_1$**	**DBPedia$_2$**	
Entities	27,615	23,182	1.19×10^6	2.16×10^6	1.82×10^8
Name-Value Pairs	186,013	816,012	1.75×10^7	3.67×10^7	1.15×10^9
Avg. Profile Size	6.74	35.20	14.66	16.94	6.31
Duplicates	22,405		892,586		$11{,}533/5.99\times10^6$
Comparisons	6.40×10^8		2.58×10^{12}		1.66×10^{16}

Table 7.1: Overview of the data sets used in our experimental study.

7.1 Data sets

To thoroughly test our blocking techniques, we employ three large-scale, real-world data sets: D_{movies} and $D_{infoboxes}$ for Clean-Clean ER[5] as well as D_{BTC09} for Dirty ER. Their technical characteristics are summarized in Table 7.1.

D_{movies} is a collection of several thousands of movies shared among IMDB and DBPedia. We derived its ground-truth from the *"imdbid"* attribute in the profiles of the DBPedia movies.

$D_{infoboxes}$ consists of two different versions of the DBPedia Infobox Data Set[6]. They have been collected by extracting all name-value pairs from the infoboxes of the articles in Wikipedia's english version. $DBPedia_1$ is a snapshot of Wikipedia Infoboxes in October 2007, whereas $DBPedia_2$ dates from October 2009. Although it may seem trivial to resolve two versions of the same data set, this is not true in our case. During the two years that intervene between $DBPedia_1$ and $DBPedia_2$, Wikipedia infoboxes were so heavily modified that there is only a small overlap between their profiles, even for duplicate entities. As shown in Table 7.2, just 40% of all name-value pairs and 50% of the attribute names are shared among the entities that are common in both versions. Regarding the ground-truth, we considered as matches those entities that had exactly the same URL. Inevitably, a small part of the actual matches has been ignored, but this pertains only to the entities that had their URL changed (e.g., due to disambiguation reasons).

D_{BTC09} constitutes the largest, real-world data set ever used for (Dirty) ER. It comprises more than 182 million entities that are described by 1.15 billion RDF statements. The data were crawled from several thousand Semantic Web sources, each having unique characteristics for the format and the quality of its informa-

[5]Both data sets have been publicly released (together with the implementation of our methods) so as to encourage researchers to use them as a benchmark. See `http://www.l3s.de/ papadakis/erFramework.html` for more details.

[6]`http://wiki.dbpedia.org/Datasets`

	Attribute Names	Name-Value Pairs
$DBPedia_1$	30,757	17,453,516
$DBPedia_2$	52,554	36,653,387
Common	27,253	10,361,467
Distinct	56,058	43,745,435

Table 7.2: Overlap in the profiles of duplicates in $D_{infoboxes}$.

tion[7]. As a result, D_{BTC09} constitutes a sizeable and representative HHIS, which is suitable for deriving safe conclusions about the generality of our blocking techniques. As ground-truth, we employed the two golden standards that were used in [PDKF10]. The first one was derived from the explicit `owl:sameAs` statements and is denoted by $SameAs$. It encompasses 5.99 pairwise million matches of the form $p_i \equiv p_j$ that, in total, involve 8.67 million distinct entities. The second ground-truth set is symbolized as IFP and was inferred from the implicit equivalence relationships of the InverseFunctionalProperties[8]. It contains $11,553$ pairwise matches among $17,990$ distinct entities.

The reason for considering two sources of ground-truth is the bias that may be lurking in the explicit equivalence relationships. In fact, it is possible that some of them stem from machine-generated same-as statements, which typically follow specific URI patterns. This is the case, for instance, with the transformation of Wikipedia URIs into DBPedia ones. Therefore, to have a better understanding of the general performance and the robustness of our algorithm, we need an additional data set that involves a higher variety of equivalence relationships, coming from a rich diversity of sources. The ground-truth set of implicit equivalence relationships (i.e., IFP) serves this need perfectly.

7.2 Evaluation of Block Building Approaches

This section consists of two parts: Section 7.2.1 compares the main variations of AC Blocking with Token Blocking over our Clean-Clean ER datasets, while Section 7.2.2 compares the URI Semantics blocking methods with Token Blocking over D_{BTC09}. In both experimental studies, we focus on the balance between effectiveness (i.e., PC) and efficiency (i.e., PQ) that is achieved by the resulting block

[7]See `http://km.aifb.kit.edu/projects/btc-2009` for more information

[8]The inverse functional properties (IFPs) provide a reliable means of discovering *implicit* equivalence relationships in the Semantic Web: *any two resources that share the same value for an IFP are, actually, identical.* As an example, consider the attribute name n_1="e-mail"; two profiles sharing the same value for n_1 most likely correspond to the same person.

collections. In addition, we consider the most important technical characteristics that determine their actual performance. These characteristics are expressed in terms of the following metrics:

- *Disk Space* occupied on the hard drive.
- *Number of Blocks* generated by the technique.
- *Method Coverage* denotes the portion of the given entities that qualify for the respective blocking method (i.e., how many entities are transformed into a non-empty set of blocking keys).
- *Block Coverage* expresses the portion of entities that are placed in at least one block[9].

The first two metrics are related to the efficiency aspects of a method (e.g., storage efficiency), with higher values corresponding to lower efficiency. In contrast, the last two measures are indicative of the robustness (and the effectiveness) of a blocking scheme, with higher values corresponding to more robust methods. Comparing Method Coverage with Block Coverage, we can actually deduce the portion of entities that share no blocking keys with any other entity, even though they qualify for the blocking scheme. This parameter is particularly crucial for the applicability of the URI Semantics blocking methods that are based on the infixes of entity profiles.

7.2.1 Clean-Clean ER

In this section, we compare Agnostic Clustering (AC) with Token Blocking over D_{movies} and $D_{infoboxes}$. We actually consider AC in conjunction with all representation models of Section 4.2.1, so as to identify the best performing combination. In short, the following variations of AC are examined:

- *Term Vector AC* relies on the combination of the term vector model with the cosine similarity,
- *Trigrams AC* is based on the combination of character trigrams with Jaccard similarity, and
- *Trigram Graphs AC* results from the combination of trigram graphs with the value similarity metric.

We also intend to evaluate the performance of Algorithm 4.1 with respect to the established clustering techniques. However, only clustering methods with an *un-*

[9]Remember that, by definition, every block has to contain at least 2 entities.

		$\mathbf{D_{movies}}$ (min.)	$\mathbf{D_{infoboxes}}$ (hrs.)
EM	Term Vector	1.49	116
	Trigrams	1.70	>200
AC	Term Vector	0.06	17
	Trigrams	0.09	66

Table 7.3: Execution time for the attribute clustering algorithms.

constrained functionality are applicable to our settings. In other words, only those methods that do not require as input the number of returned clusters are directly comparable with our algorithm. In this context, we selected as baseline a variation of the Expectation Maximization (*EM*) algorithm [DLR77] that specifies the number of clusters through an unsupervised procedure based on cross-validation[10]. EM can be combined with the character n-grams and the term vector models. For $n = 3$, the former combination is termed *Trigrams EM*, while the latter is called *Term Vector EM*. Note, though, that EM is incompatible with the n-gram graphs, since this representation model is only suitable for pairwise comparisons (i.e., it does not produce features in a vector format).

We begin our analysis by probing into the time efficiency of the two clustering algorithms. To this end, we recorded the execution time of EM and AC in combination with the term vector and the character trigrams over D_{movies} and $D_{infoboxes}$. The outcomes are presented in Table 7.3. We can notice that AC is substantially faster than EM, requiring around 1/20 and 1/6 of its running time in the case of D_{movies} and $D_{infoboxes}$, respectively. Also noteworthy is that Trigrams EM was unable to process $D_{infoboxes}$ within a time frame of 200 hours. Consequently, we consider these particular clustering settings to be inapplicable to $D_{infoboxes}$. Apparently, the higher running time of EM can only be compensated by block collections of significantly better effectiveness and efficiency. Hence, we now examine the relative performance of blocks produced by AC and EM.

Table 7.4 presents the performance of all methods on the D_{movies} data set. We can see that all variations of the clustering algorithms produce a limited number of attribute clusters, since D_{movies} contains just 11 attributes. As a result, there are minor differences in the behavior of the blocking methods (e.g., they all occupy the same disk space). Nevertheless, we can identify the following pattern: the higher the number of clusters is, the more blocks are produced and the less comparisons they entail, in total. This results in higher efficiency and moves the

[10]See `http://weka.sourceforge.net/doc/weka/clusterers/EM.html` more details.

	Clusters	**Disk Space**	**Blocks**	**Compar.** ($\times 10^8$)	**BC**$_{ov}$	**CC** ($\times 10^{-2}$)	**PC** (%)	**PQ** ($\times 10^{-5}$)
Token Blocking	1	28MB	40,825	3.05	34.30	0.57	99.92	7.35
Term Vector EM	4	52MB	33,777	2.81	32.85	0.59	97.94	7.81
Trigrams EM	2	52MB	18,707	0.48	10.86	1.18	76.55	35.72
Term Vector AC	3	52MB	43,270	2.90	33.16	0.58	99.80	7.72
Trigrams AC	3	52MB	43,271	2.91	34.08	0.59	99.82	7.70
Trigram Graphs AC	4	52MB	44,158	2.13	32.96	0.08	99.55	1.05

Table 7.4: Performance of block building techniques over D_{movies}.

BC-*CC* mapping of the blocking methods closer to the Ideal Point (i.e., their BC_{ov} decreases, while their *CC* increases). This effect has a direct impact on their actual performance, increasing *PQ* to a considerable extent, while *PC* is reduced by less than 2%. The only exception to this pattern is Trigrams EM, which involves the least number of comparisons, but fails to place in a common block almost 1 out of 4 pairs of duplicates. Thus, it constitutes the only clustering approach with inferior performance to Token Blocking. All other blocking techniques offer a better balance between *PC* and *PQ*, with Term Vector AC exhibiting the best trade-off.

Table 7.5 offers stronger evidence for the differences in the performance of the individual blocking methods. The reason is that the high number of attribute names of $D_{infoboxes}$ allows for higher variation in the attribute clusters. It is noteworthy, though, that the performance pattern of D_{movies} applies in this data set, as well: the higher the number of attribute clusters is, the higher is the resulting number of blocks and the less comparisons they entail, in total. This effect leads to substantially higher *CC* values (almost by an order of magnitude) and, thus, to higher *PQ* values. On the other hand, effectiveness (i.e., *PC*) remains practically stable. Unlike D_{movies}, however, the increase in the number of attribute clusters results in a substantial increase in the values of BC_{ov} and the space occupied on the disk, due to the significantly higher number of blocks. It is also worth noting that Term Vector EM exhibits the worst performance: it involves more comparisons than all other methods for practically the same *PC* with them. In contrast, most variations of AC provide a better balance between *PC* and *PQ* than Token Blocking, with Trigram Graphs AC exhibiting the best trade-off.

On the whole, we can argue that AC Blocking substantially improves on Token Blocking, offering higher efficiency for the same levels of effectiveness. It also outperforms EM-based blocking methods in many aspects: it is applicable to large entity collections, it can be combined with the n-gram graphs representation model, and it produces blocks of higher quality. For the last point, it should be stressed that

	Clusters	Disk Space	Blocks ($\times10^6$)	Compar. ($\times10^{12}$)	BC_{ov}	CC ($\times10^{-4}$)	PC (%)	PQ ($\times10^{-7}$)
Token Blocking	1	2.1GB	1.21	6.18	29.51	0.16	99.997	1.44
Term Vector EM	2	4.9GB	1.35	6.38	31.86	0.17	99.996	1.40
Term Vector AC	3,717	4.4GB	1.22	6.18	29.42	0.16	99.997	1.44
Trigrams AC	24,927	5.0GB	4.48	1.05	41.76	1.34	99.982	8.52
Trigram Graphs AC	26,762	5.0GB	4.80	1.03	43.22	1.41	99.992	8.70

Table 7.5: Performance of block building techniques over $D_{infoboxes}$.

	Method Coverage	Block Coverage	Disk Space (GB)	Blocks ($\times10^6$)	Comp. ($\times10^{15}$)
Token Blocking	100.00%	100.00%	114	31.16	25.91
Infix	67.20%	31.32%	13	15.34	0.006
Infix Profile	66.17%	64.84%	26	23.59	1.07
Literal Profile	55.63%	54.95%	59	17.83	2.67
Complete Infix	98.91%	97.25%	33	68.87	1.10
Infix-Literal Pr.	91.88%	74.18%	65	27.90	3.25
Infix Pr.-Literal Pr.	69.29%	68.13%	75	38.44	3.90
Total Description	99.66%	98.90%	83	81.08	3.81

Table 7.6: Technical characteristics of the block collections produced by Token Blocking and the URI Semantics blocking schemes.

EM involves a “blind” functionality: unlike our Attribute Clustering algorithm, EM does not guarantee that every cluster contains attribute names from both input entity collections. Instead, it is possible that a cluster exclusively contains attributes stemming from the same source, thus rendering their values useless for blocking. Regarding the relative performance of the representation models, there are minor differences for D_{movies}, but for $D_{infoboxes}$, the best performance clearly corresponds to the n-gram graphs. The reason is that their noise-tolerant and language-agnostic functionality turns them more suitable than the other models for tackling the intricacies of HHIS.

7.2.2 Dirty ER

In this section, we examine the performance of the URI Semantics blocking methods over D_{BTC09}. As a baseline, we employ Token Blocking, the only block building technique that is also applicable to the highly heterogeneous settings of D_{BTC09}. We begin our analysis with Table 7.6, which presents the technical characteristics of the resulting block collections.

Regarding Method Coverage, we can see that the atomic blocking schemes individually cover less than 2/3 of all entities, while their combinations have substantially higher coverage — well above 90% in most of the cases. Nevertheless, the individual coverage of Infix and Infix Profile blocking is larger than one would expect. To explain this phenomenon, we investigated the extent to which blank nodes are used as ids, not only for uniquely identifying entities, but also for expressing the associations between them. We found out that, among all data sources of D_{BTC09}, less than a third of their entities (32.61%) have a blank node as their id and a mere 4.99% of their statements have a blank node as their object. Consequently, blank nodes are completely outweighed by real URIs, thus having a restricted impact on the applicability of our method.

The values of Block Coverage follow the same pattern as those of Method Coverage: they are slightly lower than 66% for atomic blocking schemes, but significantly higher for the composite ones. In fact, the value of Block Coverage is around 2% lower than the corresponding value of Method Coverage in almost all the cases. This means that our blocking schemes place almost all entities that contain the required description item(s) in at least one block. The only exception to this rule is Infix Blocking, whose Block Coverage is less than half its Method Coverage. This discrepancy should be attributed to the *singleton infixes*, i.e., infixes that appear in just one entity identifier, thus forming no block. Another reason are the arbitrary entity identifiers, i.e., the random or numerical URIs that lack an infix.

It is interesting to examine why this pattern does not apply to Infix Profile, as well. One of the main reasons is the connectivity of the resources contained in D_{BTC09}. In fact, it was reported in [PDKF10] that 12.55% of the resources appear only as subjects, 30.49% appear solely as objects, while the rest (56.96%) appear both as subjects and objects. This evidence suggests that there are strong connections between the nodes of the underlying RDF graph, and, thus, sources with synthetic URIs are highly likely to be connected to other domains that do not necessarily follow the same methodology for generating identifiers. Therefore, they pose no serious threat to the robustness of Infix Profile and the composite blocking schemes built on it.

Continuing the analysis of Table 7.6, we observe that the results on the efficiency characteristics are quite intuitive: the composite blocking schemes involve more blocks, which entail more comparisons and occupy more disk space than the atomic ones. Therefore, the efficiency of the former is lower than that of the latter. Among all techniques, Token Blocking is the least efficient method, as it

	BC	**CC** ($\times 10^{-7}$)	**PC$_{IFP}$** (%)	**PC$_{SA}$** (%)	**PQ$_{IFP}$** ($\times 10^{-12}$)	**PQ$_{SA}$** ($\times 10^{-9}$)
Token Blocking	19.23	1.35	99.32	92.26	0.44	0.21
Infix	0.32	94.20	59.31	49.60	1,102.12	476.13
Infix Profile	2.30	3.92	84.19	43.78	9.10	2.45
Literal Profile	10.90	7.43	94.49	24.51	4.10	0.55
Complete Infix	3.07	5.10	85.16	87.14	9.00	4.76
Infix-Literal Pr.	11.34	6.36	96.43	65.67	3.44	1.21
Infix Pr.-Literal Pr.	13.11	6.61	96.64	52.09	2.87	0.80
Total Description	13.82	6.13	97.98	91.13	2.98	1.43

Table 7.7: Performance of block building techniques over D_{BTC09}.

requires more than 25% additional disk space and entails the largest number of comparisons. This is a side-effect of the low-granularity of its functionality, which results in rather frequent blocking keys (i.e., tokens); for example, most of the URIs contain the token "http". For the same reason, Literal Profile exhibits the lowest efficiency among all atomic schemes. It differs, though, from Token Blocking in two aspects: (i) it involves a lower number of distinct blocking keys, and (ii) it can be combined with the other atomic blocking schemes to enhance its performance.

The actual performance of the URI Semantics blocking schemes is reported in Table 7.7. We notice that PC is quite low for the atomic blocking schemes, but substantially higher for the composite ones — independently of the ground-truth set. In fact, the PC of composite methods is larger than that of the individual schemes comprising them in all cases. This pattern provides strong evidence in favor of merging complementary blocking schemes. The combination of all three atomic schemes (i.e., Total Description) actually achieves the highest PC among all URI Semantics blocking methods, being lower than that of Token Blocking by just 1% — for both benchmark sets. Note also that there is a clear association between the values of PC and BC, with high BC values conveying high PC ones. This interesting association is analytically discussed in Section 7.4.

Regarding efficiency, we observe two different trends for each ground-truth set: for $SameAs$, the combination of atomic blocking schemes improves at least the PQ of the less efficient one(s), whereas for IFP, the composite schemes have a lower PQ than all atomic schemes comprising them. The former pattern implies that the number of detected matches increases faster than the number of comparisons (and vise versa for the latter pattern). Indeed, we observe that there is a significant increase in PC_{SA} for composite schemes, whereas the relative increase in PS_{IFP}

	Purged Blocks	**Comp.** ($\times 10^8$)	**BC$_{\text{ov}}$**	**CC** ($\times 10^{-2}$)	**PC** (%)	**PQ** ($\times 10^{-4}$)
Token Blocking	42	1.11	30.23	1.38	99.91	2.01
Term Vector EM	38	1.10	29.09	1.34	97.65	1.99
Trigrams EM	35	0.09	8.11	4.61	75.60	18.82
Term Vector AC	76	0.81	27.72	1.75	99.78	2.78
Trigrams AC	74	0.81	28.65	1.79	99.80	2.75
Trigram Graphs AC	52	0.73	29.12	2.02	99.54	3.05

Table 7.8: Performance of Block Purging over D_{movies}.

is substantially smaller. CC exhibits the same behavior with PQ_{SA}, taking higher values for composite schemes. Note also that PQ_{SA} is significantly higher than PQ_{IFP} across all blocking schemes, because the $SameAs$ ground-truth set involves a higher number of duplicates than IFP.

Overall, we can conclude that the atomic blocking schemes exhibit the highest levels of efficiency, but suffer from deficient robustness and effectiveness. The composite blocking methods improve on both of these weaknesses, without significant sacrifices with respect to efficiency. Total Description actually achieves similar levels of effectiveness and robustness with Token Blocking even though it involves just 85% of its comparisons. Two are its advantages over Token Blocking that account for the better balance between effectiveness and efficiency it achieves:

- Token Blocking extracts all tokens from the URIs that appear as attribute values in entity profiles, whereas Total Description exclusively considers their infix. In this way, it exclusively retains the most distinctive blocking keys, saving the unnecessary comparisons that stem from repeated tokens in entity URIs.
- Token Blocking completely disregards the URIs that are used as entity identifiers, whereas Total Description extracts their infixes, as well. In this way, it enhances its robustness without any significant impact on efficiency, due to the discriminative information of infixes.

7.3 Evaluation of Block Purging

This section examines the effect of Block Purging (i.e., Algorithm 6.1) on all block building techniques across the three data sets of our experimental study. We begin with the application of Block Purging over blocking-based Clean-Clean ER in Section 7.3.1 and continue with its application over blocking-based Dirty ER in

	Purged Blocks	**Comp.** ($\times 10^{10}$)	**BC$_{ov}$**	**CC** ($\times 10^{-3}$)	**PC** (%)	**PQ** ($\times 10^{-5}$)
Token Blocking	396	5.68	16.24	0.96	99.91	1.57
Term Vector EM	564	4.34	17.24	1.32	99.90	2.05
Term Vector AC	396	5.68	16.24	0.96	99.91	1.57
Trigrams AC	1,064	3.06	27.50	3.02	99.98	2.92
Trigram Graphs AC	1,358	2.42	28.12	3.90	99.99	3.69

Table 7.9: Performance of Block Purging over $D_{infoboxes}$.

Section 7.3.2.

7.3.1 Clean-Clean ER

The performance of Block Purging over D_{movies} and $D_{infoboxes}$ is presented in Tables 7.8 and 7.9, respectively. Comparing it with the original performance of the block building techniques in Tables 7.4 and 7.5, respectively, we notice that BC_{ov} decreases for all of them, thus getting closer to the x=1 axis. On the other hand, CC increases to a great extent, getting closer to its maximum value (i.e., CC_{max}=2). All approaches move, therefore, closer to the Ideal Point, thus indicating an improvement in their balance between effectiveness and efficiency across both data sets. Indeed, Block Purging decreases PC by less than 1% in all cases, whereas the overall number of comparisons is reduced by 68% in D_{movies} and by two orders of magnitude in $D_{infoboxes}$, on average.

This behavior implies a high accuracy in detecting the oversized blocks. In fact, Block Purging performs a conservative, but valuable cleansing across both data sets, removing almost the same portion of blocks from all blocking methods: in D_{movies} it discards between 0.10% and 0.17% of all blocks and in $D_{infoboxes}$ around 0.03% of them. Given that this results in a similar impact on the performance of all methods, we can conclude that the original block collections involve similar power-law distributions of comparisons: few blocks are oversized, containing the largest part of the comparisons, while the vast majority of blocks entails a handful of entities.

Note also that, among all blocking methods, Trigram Graphs AC maintains the best balance between PC and PQ after Block Purging. It requires by far the lowest number of pairwise comparisons across both data sets, while exhibiting the highest PQ across all blocking methods with PC > 99%. Especially for $D_{infoboxes}$, it achieves the highest PC, while requiring 20% less comparisons than the next

	Comp. ($\times 10^{11}$)	**BC**	**CC** ($\times 10^{-4}$)	**PC$_{IFP}$** (%)	**PC$_{SA}$** (%)	**PQ$_{IFP}$** ($\times 10^{-8}$)	**PQ$_{SA}$** ($\times 10^{-5}$)
Token Blocking	15.49	4.31	5.07	96.79	60.52	0.73	0.23
Infix	0.004	0.28	1,144.70	58.85	49.54	1,516.59	659.53
Infix Profile	1.62	1.42	15.97	76.53	41.36	5.49	1.53
Literal Profile	9.37	2.90	5.63	94.34	18.29	1.17	0.12
Complete Infix	1.69	2.14	23.21	72.64	86.60	4.99	3.07
Infix-Literal Pr.	8.79	3.07	6.38	94.48	63.75	1.25	0.43
Infix Pr.-Literal Pr.	12.00	4.29	6.52	93.57	50.03	0.90	0.25
Total Description	11.64	4.96	7.77	95.37	89.35	0.95	0.46

Table 7.10: Performance of Block Purging over D_{BTC09}.

best approach.

7.3.2 Dirty ER

The performance of Block Purging over D_{BTC09} is reported in Table 7.10. Comparing it with the original performance of the block building methods in Table 7.7, we can observe that the required number of comparisons drops by four orders of magnitude for all of them. At the core of this substantial improvement lies the restriction imposed by block purging on the maximum block sizes: from several million entities, they were reduced to several thousand entities.

Note also that our experimental results verify the behavior of Block Purging on the *BC*-*CC* space, as demonstrated in Figure 4.4: the *BC* values move to the left of the X-axis, while the *CC* values move higher on the Y-axis. As a result, every underlying blocking method achieves a better balance between *PC* and *PQ*: the latter is enhanced by four orders of magnitude, while there is a negligible decrease in the former across both ground-truth sets. In fact, the values of *PC* are lower than in Table 7.7 by just 1% or 2%, in most of the cases. The only exceptions to this rule, which exhibit a sharp decrease in *PC*, are Infix Profile and Complete Infix with respect to IFP and the Token Blocking with respect to $SameAs$. This is a strong indication of limited robustness for the corresponding methods, as their initially high effectiveness depends extensively on their oversized blocks. In complete contrast, Total Description is the only one that retains very high *PC* (i.e., around 90%) for both ground-truth sets, thus constituting the most robust of the proposed blocking schemes.

7.4 BC-CC Mapping vs Real Performance

In this section, we experimentally verify that BC and CC provide a highly accurate, a-priori estimation of a blocking method's real performance. Our analysis relies on the *Pearson correlation coefficient* $\rho(\mathbf{X}, \mathbf{Y})$, a well established measure for estimating the linear dependency between two variables, X and Y. It takes values in the interval $[-1, 1]$, with higher absolute values corresponding to a stronger correlation between X and Y. A value of $|\rho(X, Y)| = 1$ actually indicates a completely linear relationship of the form $X = \alpha \cdot Y + \beta$, where $\alpha, \beta \in \mathcal{R}$ and $0 < \alpha$ if $\rho_{X,Y} = 1$, while $\alpha < 0$ if $\rho_{X,Y} = -1$. In this context, our goal is to prove that the values of BC exhibit a highly positive correlation with those of PC and that the same applies to CC and PQ.

Our analysis consists of two parts. The first one measures $\rho(BC, PC)$ and $\rho(CC, PQ)$ across different blocking methods that are used for the same task. In this way, we can examine whether our metrics are suitable for a-priori distinguishing the best performing technique for a given task. The second part measures $\rho(BC, PC)$ and $\rho(CC, PQ)$ inside the same blocking method for various values of the purging threshold (cf. Section 6.2.1). In this way, we can investigate the usefulness of incorporating BC and CC into our Block Purging approach (i.e., Algorithm 6.1). In combination, these two analyses evaluate holistically the main applications of our BC-CC metric space.

For the first analysis, we consider all techniques applied to Block Building and Block Purging over all data sets. We actually derive the correlations $\rho(BC, PC)$ and $\rho(CC, PQ)$ from the performance reported in Tables 7.4, 7.5 and 7.7 for Block Building and in Tables 7.8, 7.9 and 7.10 for Block Purging. The outcomes of this analysis are presented in Table 7.11. We observe that there is a positive correlation between BC and PC in all cases, but for Block Building over $D_{infoboxes}$. This exception is caused by the practically identical PC shared by all block building methods over $D_{infoboxes}$, despite the fact that their BC varies greatly. This rather rare behavior, though, is totally rectified by Block Purging: the variation it yields in the effectiveness of the blocking methods turns the originally negative correlation into a highly positive one. Another interesting aspect is the relatively low values that $\rho(BC, PC)$ takes across both tasks for the $SameAs$ ground-truth set of D_{BTC09}. This can be explained by the disproportionately higher increase in PC (in comparison with BC), which results from the combination of atomic blocking schemes into composite ones. In contrast, $\rho(BC, PC)$ takes higher values for IFP,

		$\mathbf{D_{movies}}$	$\mathbf{D_{infoboxes}}$	$\mathbf{D_{BTC09}}$	
				IFP	**SameAs**
Block	$\rho(BC, PC)$	0.998	-0.786	0.838	0.365
Building	$\rho(CC, PQ)$	0.755	0.999	0.998	0.998
Block	$\rho(BC, PC)$	0.991	0.986	0.897	0.319
Purging	$\rho(CC, PQ)$	0.989	0.992	0.999	0.999

Table 7.11: Pearson correlation between BC and PC as well as between CC and PQ over all data sets for Block Building and Block Purging.

because the composite blocking schemes convey similar relative increase in both PC and BC. Regarding the correlation between CC and PQ, we notice that it takes exceptionally high positive values for Block Building, with Block Purging raising it even higher, to the maximum possible levels (i.e., $\rho(CC, PQ) \approx 1$ after Block Purging). On the whole, we can deduce that BC and CC can accurately discern the best technique among a set of blocking methods with high efficiency (i.e., from a simple inspection of the blocks at hand).

The goal of the second analysis is to examine the contribution of BC and CC metrics to the functionality of Block Purging. To this end, we evaluate the correlations $\rho(BC, PC)$ and $\rho(CC, PQ)$ when applying various purging thresholds to all blocking schemes over D_{BTC09}. To ensure a large variety of purging thresholds, we employ a mechanism different from Algorithm 6.1, which derives the maximum block size from the following formula: $|b_i^{max}| = 10^{\log|\mathcal{E}|/d_i}$, where $|b_i^{max}|$ is the purging threshold, $|\mathcal{E}|$ is the size of the input entity collection, and d_i is an integer that takes all values in the interval $[1, 10]$. The outcomes of our analysis are presented in Table 7.12. We observe that there is a highly positive correlation between BC and PC, regardless of the ground-truth set: on average, their correlation is $\rho(\bar{PC}, BC) = 0.83$ for IFP and $\rho(\bar{PC}, BC) = 0.73$ for $SameAs$, with the total average being equal to $\rho(\bar{PC}, BC) = 0.78$. Regarding the correlation between CC and PQ, its minimum value across all data sets and blocking methods is 0.985, which implies that there is a positive, practically linear relation between the two measures under all settings. These high values for both $\rho(BC, PC)$ and $\rho(CC, PQ)$ lead to the safe conclusion that our metrics accurately determine whether a purging threshold has a significantly negative impact on the performance of the underlying blocking method. In such cases, a higher value for the maximum block size is required.

	$\rho(\mathbf{BC}, \mathbf{PC_{IFP}})$	$\rho(\mathbf{BC}, \mathbf{PC_{SA}})$	$\rho(\mathbf{CC}, \mathbf{PQ_{IFP}})$	$\rho(\mathbf{CC}, \mathbf{PQ_{SA}})$
Token Blocking	0.68	0.93	0.986	0.998
Infix	0.99	0.76	0.993	0.999
Infix Profile	0.97	0.82	0.987	0.999
Literal Profile	0.67	0.89	0.987	0.999
Complete Infix	0.97	0.73	0.995	0.999
Infix-Literal Pr.	0.71	0.67	0.990	0.997
Infix Pr.-Literal Pr.	0.75	0.66	0.985	0.999
Total Description	0.77	0.56	0.993	0.999

Table 7.12: Pearson correlation between BC and PC as well as between CC and PQ for the URI Semantic blocking schemes over D_{BTC09}.

7.5 Evaluation of Meta-blocking Approaches

The goal of our experimental study is manifold: *(i)* to demonstrate the benefits of meta-blocking over existing blocking methods (Section 7.5.1), *(ii)* to compare the edge-centric pruning schemes with the node-centric ones (Section 7.5.2), *(iii)* to compare the weight pruning criteria with the cardinality ones (Section 7.5.3), *(iv)* to compare the weighting schemes for building blocking graphs (Section 7.5.4), *(v)* to compare meta-blocking with the state-of-the-art approach of Iterative Blocking (Section 7.5.5), *(vi)* to examine the robustness of our pruning schemes (Section 7.5.7), and *(vii)* to investigate the time requirements of meta-blocking over large blocking graphs with millions of nodes and billions of edges (Section 7.5.8).

In the following, we evaluate the performance of our meta-blocking techniques using the combination of Token Blocking with Block Purging as baseline. Thus, the block collections given as input to our meta-blocking techniques have their efficiency significantly improved with respect to the original block building technique. In more detail, they require 1.11×10^{8} and 5.68×10^{10} pairwise comparisons for D_{movies} and $D_{infoboxes}$, respectively, while exhibiting high effectiveness (i.e., PC>99.91%) for both of data sets.

We also considered a unilateral block collection, which was extracted from the blocks produced by *Total Description* when applied to the entire BTC09 data collection. To restrict the originally massive dataset to a moderate block collection that facilitates our thorough experimental analysis, we first purged those blocks that contained none of the IFP ground-truth entities. We then removed the singleton entities, which were associated with just one block after sampling, in order to ensure a redundancy-positive block collection (BC>1) that allows for applying meta-blocking. Finally, we discarded the invalid blocks, which were left with just one entity, and applied Block Purging [PIN+12] on the remaining set of blocks. The resulting dataset is denoted by D_{BTC09} and comprises 106,462 blocks with 253,353 entities and 10,653 pairs of matching entities. Similar to the other block collections, D_{BTC09} combines a high RR(>99%) with a high PC(≈97%).

Note that we estimate the impact of meta-blocking on the effectiveness of the input block collections through the relative reduction in PC, which is measured by the ΔPC metric. Formally, this metric is defined as follows:

$$\Delta PC = \frac{PC(\mathcal{B}') - PC(\mathcal{B})}{PC(\mathcal{B})} \cdot 100\%,$$

where $PC(\mathcal{B})$ and $PC(\mathcal{B}')$ denote the pair completeness of the original and the restructured block collection, respectively. Apart from Block Building in combination with Block Purging, we also consider as a baseline the state-of-the-art approach of Iterative Blocking [WMK+09]. In essence, this method propagates every new pair of duplicates to all associated blocks (even if they have already been examined) in order to identify additional matches and to save unnecessary comparisons.

7.5.1 Effect of meta-blocking on blocking.

We applied all pruning schemes to all blocking graphs (i.e., weighting schemes) that can be derived from D_{movies}, $D_{infoboxes}$ and D_{BTC09}. We categorized the results according to the corresponding pruning scheme and analytically present them in Tables 7.13(a) to 7.13(d).

Table 7.13(a) depicts the performance of WEP in conjunction with all weighting schemes across all datasets. For D_{movies} and $D_{infoboxes}$, we notice that all weighting schemes convey significant enhancements in efficiency (RR>70%), while incurring moderate reduction in PC (ΔPC<10%). Similar patterns are exhibited for D_{BTC09}: in the worst case ΔPC≈10%, while RR remains higher than 95% for all weighting schemes. The performance of most of them is actually very close over D_{BTC09}. In contrast, for D_{movies} and $D_{infoboxes}$, there is a clear trade-off between RR and PC: the higher the former gets, the lower the latter is and vice versa. Note, though, that the evolution of PQ suggests that RR decreases faster than PC increases.

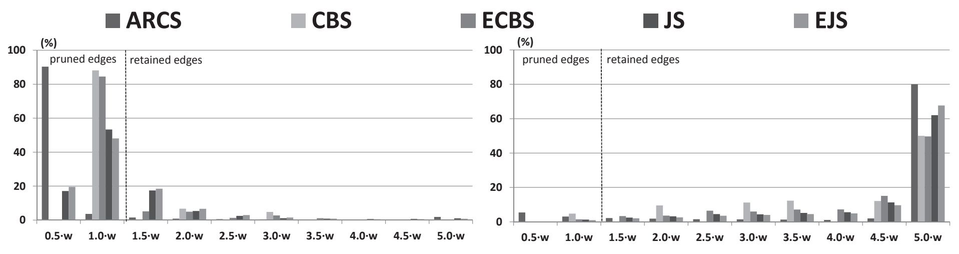
These patterns can be explained by the weight distribution lying at the core of each weighting scheme. As an example, consider Figures 7.1(a) and (b), which de-

	D$_{\textbf{movies}}$					**D**$_{\textbf{infoboxes}}$					**D**$_{\textbf{BTC09}}$				
	Comp.	*RR*	*PC*	ΔPC	*PQ*	Comp.	*RR*	*PC*	ΔPC	*PQ*	Comp.	*RR*	*PC*	ΔPC	*PQ*
	($\times 10^6$)	(%)	(%)	(%)	(10^{-2})	($\times 10^8$)	(%)	(%)	(%)	(10^{-4})	($\times 10^7$)	(%)	(%)	(%)	(10^{-4})
It. Bl.	10.41	61.06	99.39	0	0.21	255.94	35.67	99.89	0	0.35	12.98	0.84	98.22	1.32	0.81
ARCS	1.38	94.82	90.89	-8.55	1.47	2.85	99.28	92.45	-7.45	29.00	0.41	99.35	94.77	-2.24	24.85
CBS	2.71	89.88	94.68	-4.74	0.78	33.97	91.46	95.47	-4.42	2.51	2.16	96.57	86.84	-10.42	4.29
ECBS	3.52	86.82	97.95	-1.45	0.62	57.71	85.50	99.66	-0.23	1.54	1.81	97.12	86.60	-10.67	5.08
JS	6.71	74.90	97.93	-1.46	0.33	112.21	71.80	99.73	-0.16	0.79	2.15	96.58	87.13	-10.12	4.31
EJS	7.34	72.54	98.32	-1.07	0.30	110.14	72.32	99.77	-0.11	0.81	2.13	96.61	89.01	-8.18	4.45
(a) WEP															
ARCS	2.55	90.44	96.55	-2.86	0.85	14.84	96.27	99.41	-0.48	5.98	2.25	96.43	95.72	-1.26	4.54
CBS	2.86	89.31	97.19	-2.21	0.76	35.65	91.04	99.35	-0.54	2.49	2.69	95.72	91.46	-5.66	3.62
ECBS	6.92	74.10	98.64	-0.75	0.32	99.37	75.02	99.75	-0.14	0.90	3.42	94.56	91.13	-5.99	2.84
JS	10.00	62.59	98.68	-0.71	0.22	195.93	50.76	99.87	-0.02	0.46	4.22	93.29	91.43	-5.68	2.31
EJS	11.81	55.77	99.16	-0.23	0.19	199.96	49.74	99.88	-0.01	0.45	4.41	93.00	92.52	-4.56	2.24
(b) WNP															
ARCS	0.57	97.87	82.75	-16.74	3.25	0.26	99.94	79.46	-20.46	276.83	0.09	99.85	92.17	-4.92	103.99
CBS	0.57	97.87	75.78	-23.75	2.98	0.26	99.94	51.71	-48.37	179.68	0.09	99.85	24.07	-75.17	27.16
ECBS	0.57	97.87	81.58	-17.92	3.20	0.26	99.94	62.14	-37.79	216.49	0.09	99.85	42.81	-56.05	48.07
JS	0.57	97.87	79.12	-20.40	3.11	0.26	99.94	82.09	-17.83	285.98	0.09	99.85	25.77	-99.55	29.07
EJS	0.57	97.87	84.87	-14.61	3.33	0.26	99.94	79.61	-20.30	277.37	0.09	99.85	45.85	-52.71	51.73
(c) CEP															
ARCS	1.10	95.88	94.13	-5.39	1.91	0.50	99.88	96.87	-3.02	174.63	0.18	99.72	95.60	-1.38	58.22
CBS	1.10	95.88	95.20	-3.48	1.95	0.50	99.88	96.34	-3.56	173.68	0.18	99.72	88.70	-8.50	54.02
ECBS	1.10	95.88	96.69	-2.71	1.97	0.50	99.88	97.72	-2.17	176.17	0.18	99.72	84.34	-11.03	52.53
JS	1.10	95.88	94.93	-4.45	1.93	0.50	99.88	96.86	-3.03	174.62	0.18	99.72	83.79	-13.57	51.03
EJS	1.10	95.88	95.98	-3.43	1.95	0.50	99.88	97.18	-2.71	175.19	0.18	99.72	84.50	-12.83	51.46
(d) CNP															

Table 7.13: Performance of all pruning schemes in combination with all weighting schemes over the three datasets of our study.

pict the distribution for every weighting scheme over D_{movies} (similar distributions are exhibited in the other two datasets, as well, but we omit the corresponding diagrams, due to lack of space). In all histograms, the bucket size is set equal to half the average edge weight ($\bar{w}$) of the corresponding scheme across the entire blocking graph (i.e., including the links between matching and non-matching nodes/entities). Thus, the two leftmost bars correspond to the pruned edges and the remaining eight bars to the retained ones. We observe a clear polarization for all weighting schemes: the vast majority of the matching edges is concentrated on the two right-most intervals, with a negligible portion of them lying in the left half (the opposite applies to non-matching edges). In fact, the higher the *PC* of a weighting scheme over D_{movies} is, the lower is the corresponding number of matching edges in the first two intervals. On the other hand, the higher its *RR* is, the lower is the portion of non-matching edges placed in the intervals $[1.5{\cdot}\bar{w}, 5{\cdot}\bar{w}]$.

Table 7.13(b) illustrates the performance of *WNP* for all weighting schemes over all datasets. Similar to *WEP*, there is a clear trade-off between effectiveness and efficiency for D_{movies} and $D_{infoboxes}$. It is interesting to note that ranking the



(a) Non-matching edges. (b) Matching edges.

Figure 7.1: Normalized histograms of the weight distributions in all blocking graphs of D_{movies}, where w denotes the average edge weight of the blocking graph for each weighting scheme.

weighting schemes in descending order of RR (i.e., ascending order of PC) results in the same order as in Table 7.13(a). For D_{BTC09}, all weighting schemes achieve similar, high performances with respect to all metrics. Compared to WEP, though, the combination of every weighting scheme with WNP yields significantly higher PC as well as lower RR and PQ.

Table 7.13(c) presents the performance of CEP in combination with all weighting schemes across the three datasets. By definition, they all achieve the same RR, which amounts to 97.48%, 99.94% and 99.85% for D_{movies}, $D_{infoboxes}$ and D_{BTC09}, respectively. In absolute numbers, this corresponds to 11, 15 and 3 comparisons per entity, respectively, thus requiring 2 orders of magnitude fewer comparisons than the input block collection. Apparently, this is at the cost of lower effectiveness, since PC is reduced in all datasets by more than 14%, regardless of the weighting scheme (the only exception is $ARCS$ for D_{BTC09}). The worst performance usually corresponds to CBS and JS, because there are many pairs of entities that share exactly the same number or portion of blocks, respectively. Again, this behavior can be explained by the normalized histograms in Figures 7.1(a) and (b), since CEP generally retains the edges of the rightmost interval; the more matching edges and the less non-matching ones it contains, the higher is the PC of the corresponding weighting scheme.

Finally, Table 7.13(d) presents the performance of CNP for all weighting schemes across all datasets. Similar to its edge-centric counterpart, it exhibits excessively high efficiency for both datasets (i.e., RR>95%). In absolute numbers, this corresponds to 22, 28 and 7 comparisons per entity for D_{movies}, $D_{infoboxes}$ and D_{BTC09}, respectively. Its impact on effectiveness is rather limited, reducing PC at most by 5% for the Clean-Clean ER datasets and less than 14% for the Dirty ER one.

7.5.2 Edge-centric vs. node-centric pruning schemes.

The relative performance of these two types of pruning schemes depends on the pruning criteria that lie at their core. Thus, an equal basis comparison requires exactly the same configuration. This is impossible, though, for the weight criteria: WEP can only be combined with a global one, while WNP makes sense only when coupled with a local one (its conjunction with a global threshold renders it identical to WEP).

The configuration of Section 5.3 approximates the ideally equal settings, assuming similar criteria for both algorithms (i.e., average edge weight). For this configuration, our experiments suggest that the edge-centric algorithms perform a deeper pruning that results in the lowest number of comparisons and detected matches (i.e., lowest ΔPC). Nevertheless, they are more accurate in discarding superfluous comparisons, achieving higher PQ across all datasets and weighting schemes. For example, consider the combination of $ARCS$ with WEP and WNP over D_{movies}: PQ suggests that for every 100 comparisons, the former identifies around 1.5 matches and the latter almost half of them.

On the other hand, the node-centric schemes are more conservative in pruning edges, retaining even double as much comparisons. Thus, they have a significantly smaller impact on PC, which is also ensured by the more even distribution of comparisons among entities; unlike the edge-centric algorithms, which completely disregard the entities/nodes that are associated with none of the top weighted edges, they ensure that *every* node remains connected with the most similar of its co-occurring entities.

In the case of cardinality pruning criteria, it is possible to apply the same global threshold to both CEP and CNP. However, these settings merely allow for comparing the relative effectiveness, since they involve the same number of executed comparisons for both algorithms. We put these settings into practice using as threshold for CEP the total number of comparisons required by CNP. The outcomes with respect to PC are presented in Table 7.14 and confirm that the node-centric algorithms achieve a significantly higher effectiveness than the edge-centric ones, across all datasets and weighting schemes.

In summary, the most appropriate meta-blocking settings for the application at hand depend on its performance requirements and the available resources (assuming the configuration of Section 5.3). The node-centric pruning schemes are suitable for applications emphasizing on effectiveness, provided that they can afford the high space requirements (these pruning schemes store a threshold or a

	$\mathbf{D_{movies}}$		$\mathbf{D_{infoboxes}}$		$\mathbf{D_{BTC09}}$	
	PC_{CEP}	PC_{CNP}	PC_{CEP}	PC_{CNP}	PC_{CEP}	PC_{CNP}
ARCS	89.16%	94.13%	83.82%	96.87%	93.22%	95.60%
CBS	80.42%	95.20%	60.46%	96.34%	31.97%	88.70%
ECBS	87.17%	96.69%	67.85%	97.72%	65.78%	86.25%
JS	89.22%	94.93%	86.02%	96.86%	35.97%	83.79%
EJS	91.03%	95.98%	85.26%	97.18%	51.85%	84.50%

Table 7.14: Comparing effectiveness between CEP and CNP for the same number of comparisons across all datasets.

certain number of comparisons *per entity*). They are also particularly useful for tasks that are inherently expressed in terms of entities (e.g., applications like social networks that seek duplicates for a specific subset of the input entities) and for entity collections that are expected to contain a large portion of duplicate profiles (i.e., there is a matching entity for most of the nodes). In contrast, the edge-centric pruning schemes are suitable for applications like incremental ER that focus on efficiency, especially when the portion of matching entities is expected to be rather low; in these settings, the top weighted edges are more likely to correspond to the few duplicate profiles.

7.5.3 Weight vs. cardinality pruning criteria.

There is a clear pattern in the relative performance of weight and cardinality pruning thresholds for the configuration of Section 5.3: the former put more emphasis on effectiveness and the latter on efficiency. In fact, the combination of any weighting scheme with a cardinality threshold requires at least half the comparisons than its combination with the corresponding weight one, regardless of the selected pruning algorithm. In most of the cases, this difference amounts to a whole order of magnitude in the actual number of comparisons. Note, though, that this radical increase in efficiency is accompanied by a moderate difference in effectiveness, due to the efficacy of cardinality thresholds in distinguishing the matching comparisons from the superfluous ones. Comparing the PQ of CEP (CNP) with that of WEP (WNP), we observe that the former is usually higher than the latter by a whole order of magnitude. Still, weight thresholds exhibit higher PC, reducing it — in the worst case — half as much as the corresponding meta-blocking settings with a cardinality criterion. Therefore, there is no dilemma when choosing the appropriate criterion with respect to the application requirements. Note, though, that this decision also depends on the available resources, since the cardinality criteria have higher memory requirements.

7.5.4 Comparison between weighting schemes.

For D_{BTC09}, $ARCS$ consistently achieves the highest performance with respect to all block quality metrics, while the rest of the weighting schemes exhibit similar, but lower performance in most of the cases. For the Clean-Clean ER dataset, the choice depends on the functionality of the pruning criterion. In more detail, $ECBS$ offers a balanced choice for the weight pruning criteria, combining high efficiency enhancements with negligible reductions in PC. For the cardinality pruning criteria, where RR remains stable across all weighting schemes, EJS consistently achieves the (nearly) best efficiency-effectiveness balance, scoring the highest PC values in most of the cases.

Of particular interest, though, is the comparison between the plain weighting schemes and their enhanced versions; that is, between CBS and $ECBS$ as well as between JS and EJS. The actual question is whether the more information included in the enhanced schemes leads to a better balance between RR and PC than the plain ones. The weight pruning criteria does not offer a clear answer; we can merely observe that the enhanced schemes offer lower RR and lower PQ at the benefit of higher PC. In contrast, the cardinality pruning criteria allow for a direct comparison: RR is the same across all weighting schemes, but the enhanced ones achieve higher PC in practically all the cases. PQ also takes significantly higher values for $ECBS$ and EJS. We can conclude, therefore, that the enhanced schemes convey significant enhancements in the performance of CBS and JS.

7.5.5 Comparison with Iterative Blocking.

Before examining the performance of Iterative Blocking, it is worth clarifying that its functionality in the context of Clean-Clean ER is reduced to discarding part of the superfluous comparisons. In fact, it propagates all detected duplicates to the subsequently processed blocks and merely saves those comparisons that involve at least one entity that has been matched to some other. This approach conveys significant efficiency enhancements when applied to redundancy-positive block collections: its RR exceeds 60% for D_{movies} and 35% for $D_{infoboxes}$. All meta-blocking methods, though, achieve higher efficiency gains, as they have a broader scope, targeting all superfluous comparisons. This is also verified by PQ, which indicates that Iterative Blocking executes the highest portion of superfluous comparisons across both datasets. Its only advantage is that it incurs no impact on effectiveness. In practice, though, this is of minor importance, given that most meta-blocking approaches have limited cost in effectiveness in the context of Clean-Clean ER.

The real strength of Iterative Blocking lies in Dirty ER, especially in applications that involve equivalence classes of high cardinality. In these settings, it puts more emphasis on identifying additional matches, thus yielding the highest PC among all methods. This is exactly the case with D_{BTC09}: although the original PC is already high, amounting to 97%, Iterative Blocking increases it by more than 1%. The re-examination of large blocks, though, increases the number of executed comparisons and prevents significant enhancements in efficiency. Indeed, it merely saves around 1% of all comparisons in the case of D_{BTC09}. Thus, its efficiency is significantly lower than meta-blocking, which again discards more superfluous comparisons.

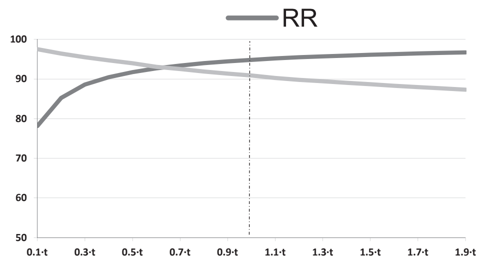
In summary, Iterative Blocking is only appropriate for applications that place effectiveness in priority and are satisfied with rather conservative savings in efficiency. For the rest of them, meta-blocking offers a better balance between effectiveness and efficiency.

7.5.6 Discussion

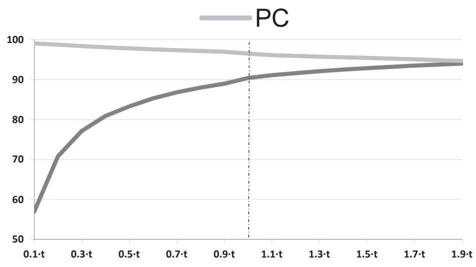
In summary, we can conclude that among the weighting schemes, $ECBS$ consistently offers a good balance between effectiveness and efficiency over Clean-Clean ER. For Dirty ER, though, $ARCS$ offers the best approach. We also observe that the node-centric approaches perform a shallow pruning that yields lower PQ and RR values than edge-centric ones. This allows them to retain almost intact the original effectiveness, especially when combined with weight thresholds. Therefore, applications that place more emphasis on effectiveness should opt for node-centric pruning schemes, while those focusing on efficiency should consider the edge-centric ones. Among the two types of pruning criteria, the weight thresholds are more robust with respect to effectiveness, while the cardinality thresholds are appropriate for applications emphasizing on efficiency, such as incremental ER.

7.5.7 Sensitivity Analysis

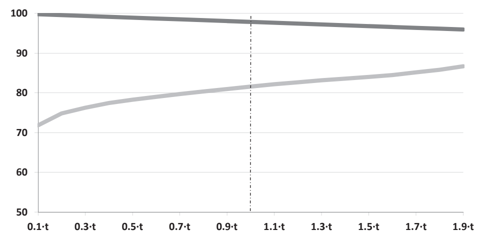
As mentioned above, the performance of pruning algorithms depends largely on the underlying pruning criterion — regardless of its scope or functionality. To examine



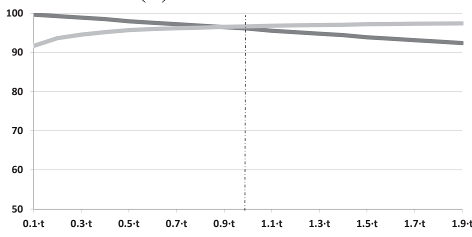
(a) *WEP* over *ARCS*



(b) *WNP* over *JS*



(c) *CEP* over *ECBS*



(d) *CNP* over *CBS*

Figure 7.2: Sensitivity analysis of every pruning algorithm in conjunction with a specific weighting scheme.

how our pruning schemes behave as a function of their thresholds, we performed sensitivity analyses of *RR* and *PC* for all schemes over the three datasets of our study. In Figures 7.2(a) to (d), we present the behavior of each pruning algorithm in combination with a specific weighting scheme over D_{movies} (for each algorithm, the rest of the weighting schemes demonstrated similar patterns and, thus, are omitted for brevity. Nevertheless, we tried to cover all of them, considering in each diagram a different one.). Every diagram was derived by incrementing the pruning threshold from $0.1{\cdot}t$ to $1.9{\cdot}t$ with a step of $0.1{\cdot}t$, where t denotes the threshold derived from the configuration of Section 5.3 (e.g., the average edge weight in the case of *WEP*).

In every figure, we observe that there is a clear trade-off between *RR* and *PC*. Higher thresholds increase *RR* and reduce *PC* for the weight pruning criteria, and vice versa for the cardinality ones. In fact, the evolution of *PC* is practically linear for all pruning schemes. The same applies to *RR* for the cardinality criteria, whereas for the weight ones, the linear evolution is preceded by a steep rise for the interval $[0.1{\cdot}t, 0.5{\cdot}t]$. The thresholds of Section 5.3 correspond to the vertical dotted line intersecting the middle of the x-axes. We observe that in every case, small variations in the size of t lead to small variations in the resulting performance. This suggests that the threshold we selected for each pruning scheme achieves a good balance between effectiveness and efficiency. Thus, it provides a good basis for adjusting a meta-blocking method to the requirements of the application at hand. For example, an application employing CEP could double the threshold specified by our approach in order to rise PC by 10% for double as many comparisons.

In summary, the sensitivity analysis of Figures 7.2(a) to (d) demonstrate that our meta-blocking methods are robust with respect to the threshold configurations of Section 5.3.

7.5.8 Time Requirements of Meta-blocking

The real usefulness of meta-blocking depends on the relation between the time required for building and pruning the blocking graph and the time consumed while performing the (spared) pairwise comparisons. The goal of this section is to examine whether the former is significantly lower than the latter, thus justifying the use of our approaches. To this end, we evaluate the time requirements of meta-blocking using three measures:

- *Materialization Time* (MT) refers to the time required by the first two steps of meta-blocking, i.e., graph building and edge weighting.
- *Restructure Time* (RT) corresponds to the last two steps of meta-blocking, i.e., graph pruning and block collecting.
- *Comparison Time* (CT) indicates the time required for performing the (retained) pairwise comparisons.

As the baseline method, we consider the one that iterates over the input blocks, executing all the comparisons they entail, without any further processing (i.e., its processing time exclusively corresponds to CT, while MT=RT=0). For all methods, the similarity of entity profiles is defined as the Jaccard coefficient of their tokenized attribute values; any other entity comparison technique is also applicable, but this choice is orthogonal to the proposed method, thus not altering our experimental results.

The outcomes of our experiments are presented in Tables 7.15. We notice the following patterns for the vast majority of meta-blocking approaches across all datasets: first, the overall processing time of the weighting pruning criteria is dominated by CT, with MT and RT merely accounting for a fraction of it. Exception to this rule is $ARCS$ in conjunction with WEP and WNP, as the low discernibility of its weights ($\ll$0.1 in most of the cases) results in a time-consuming meta-blocking process. Second, there is a balance between CT and $MT + RT$ for the cardinality pruning criteria, since they entail a very low number of comparisons with respect to

		$\mathbf{D_{movies}}$ (minutes)				$\mathbf{D_{infoboxes}}$ (hours)				$\mathbf{D_{BTC09}}$ (minutes)			
		MT	*RT*	*CT*	$\sum$	*MT*	*RT*	*CT*	$\sum$	*MT*	*RT*	*CT*	$\sum$
Baseline		.0	.0	14	14	.0	.0	128	128	0	0	111	111
	ARCS	.1	.6	1.0	1.6	3.2	24.4	25.7	53.3	.2	2.5	5.9	8.7
W	*CBS*	.1	.1	.9	1.1	3.3	7.0	21.2	31.6	.2	1.5	19.8	21.5
E	*ECBS*	.1	.2	1.2	1.4	3.1	6.7	30.8	40.6	.2	1.8	17.2	19.2
P	*JS*	.1	.1	2.1	2.3	3.2	6.0	51.2	60.4	.2	1.9	20.2	22.3
	EJS	.1	.2	2.3	2.5	3.2	6.7	52.0	62.0	.2	2.0	20.2	22.4
	ARCS	.1	.6	1.3	1.9	3.5	25.9	28.7	58.1	.2	2.7	21.5	24.5
W	*CBS*	.1	.1	1.0	1.2	3.2	6.2	24.4	33.9	.2	1.7	24.3	26.3
N	*ECBS*	.1	.2	2.1	2.4	3.6	7.5	33.4	44.6	.2	2.1	30.9	33.2
P	*JS*	.1	.1	3.0	3.2	3.5	7.0	55.4	65.9	.2	2.1	37.7	40.1
	EJS	.1	.2	3.6	3.8	3.6	8.0	58.5	70.1	.2	2.4	39.6	42.1
	ARCS	.1	.6	.2	.9	3.2	24.5	.1	27.9	.2	2.6	.8	3.6
C	*CBS*	.1	.1	.2	.4	4.2	7.4	.1	11.7	.2	1.5	.8	2.5
E	*ECBS*	.1	.2	.2	.4	4.4	8.0	.1	12.6	.2	1.9	.8	2.9
P	*JS*	.1	.2	.2	.4	4.2	7.5	.1	11.8	.2	1.9	.8	2.9
	EJS	.1	.2	.2	.4	3.2	7.1	.1	10.4	.2	2.2	.8	3.2
	ARCS	.1	.6	.3	1.0	3.2	24.7	.2	28.1	.2	2.7	1.5	4.4
C	*CBS*	.1	.1	.3	.5	3.8	6.7	.2	10.8	.2	1.6	1.5	3.3
N	*ECBS*	.1	.2	.3	.6	3.7	6.9	.2	10.9	.2	2.0	1.5	3.6
P	*JS*	.1	.2	.3	.6	3.2	6.3	.2	9.8	.2	1.9	1.5	3.6
	EJS	.1	.2	.3	.6	3.2	7.1	.2	10.6	.2	2.3	1.5	4.0

Table 7.15: Processing time for all meta-blocking methods over the three datasets of our experimental study.

the size of the graph. Again, *ARCS* corresponds to the least efficient meta-blocking process.

We also notice that for every dataset, *MT* and *RT* take almost identical values for all weighting schemes, with the small variations corresponding to the different functionality of each weighting scheme. Regarding *CT*, we observe that it takes significantly lower values for the cardinality pruning criteria than for the weight ones. This overhead is caused not only by the lower number of comparisons retained by the former, but also by the fact that the latter iterate over all edges of the blocking graph during the comparisons phase.

In summary, we observe that all combinations of pruning schemes with a weighting one require significantly less time than the baseline method. For example, the most efficient meta-blocking techniques for D_{movies} (i.e., *CEP* in conjunction with *CBS* or *JS*) are 35 times faster than the baseline. Even the most time-consuming meta-blocking settings for each dataset run at least 2 times faster than the baseline. As explained in Section 5.1, this should be attributed to the efficient materialization of the blocking graph, which involves lower complexity than the string-based techniques for comparing entity profiles.

Note that optimization techniques can be integrated into the implementation of the meta-blocking and the entity comparison methods. For instance, during the pruning of the blocking graph, edges with weights lower than the specified threshold can be identified more efficiently with the help of prefix filtering. No such technique was considered, though, in our experimental study, since it is orthogonal to our evaluation.

7.6 Evaluation of Block Processing Approaches

In this section, we analyze the performance of three different ER workflows, which share the same core: they are all based on Trigram Graphs AC for the creation of blocks and on Block Purging and Duplicate Propagation for their processing. They differ, though, in the additional methods they involve:

- WF_1 adds exclusively block-refinement methods to the core, namely Block Scheduling and Block Pruning [PINF11].
- WF_2 combines block-refinement methods with comparison-refinement ones, namely Block Scheduling with Comparison Propagation and Comparison Pruning [PIN+11b].
- WF_3 is the only workflow that operates exclusively on the level of individual comparisons, involving Comparison Scheduling, Comparison Propagation and Comparison Pruning[11].

We selected these workflows for a number of reasons: they were all formed according to the guidelines of Section 6.4 and collectively cover all efficiency methods presented in Chapter 6. WF_1 and WF_2 have already been examined over Token Blocking in [PINF11] and [PIN+11b], respectively; given that we employ the same data sets, our results are directly comparable with prior work. WF_3 is a novel workflow, but it is based on WF_2, modifying it so that it is compatible with Comparison

[11]In all cases, the ES_{min} threshold for Comparison Pruning was derived from Formula 6.3 by setting $a = 0.20$. This is a conservative value lying very close to $a = 0.25$, which induced a minor reduction in PC of Token Blocking, while boosting its PQ [PIN+11b]. The slightly lower value of a is justified by the substantially higher number of blocks produced by Trigram Graphs AC in comparison to Token Blocking.

		Compar. ($\times 10^6$)	**Duplic.**	**PQ** ($\times 10^{-2}$)	**PC** (%)	**Time** (min.)
	Block Purging	73.036	22,301	0.031	99.54	0.14
WF_1	Block Scheduling	0.383	22,301	5.820	99.54	0.05
	Block Pruning	0.267	22,295	8.369	99.51	0.05
WF_2	Comp. Propagation	60.878	22,301	0.037	99.54	0.67
	Block Scheduling	0.315	22,301	7.074	99.54	0.05
	Comp. Pruning	0.097	21,454	22.042	95.76	0.51
WF_3	Comp. Propagation	60.878	22,301	0.037	99.54	0.67
	Comp. Pruning	2.418	21,454	0.887	95.76	0.51
	Comp. Scheduling	0.087	21,454	24.611	95.76	0.06

Table 7.16: Perfomance of three different workflows over D_{movies}, when applied on top of Block Purging and Trigram Graphs AC.

Scheduling. They also differ significantly in the complexity of their functionality: WF_1 conveys minimum space and time requirements, whereas WF_3 involves the most complex methods with respect to both aspects; WF_2, on the other hand, lies in the middle of these two extremes.

The performance of all workflows over D_{movies} and $D_{infoboxes}$ is presented in Tables 7.16 and 7.17, respectively, with the individual methods of each workflow appearing in the order they are executed. Note that the performance of the two scheduling methods is actually derived from their combination with Duplicate Propagation. It denotes, therefore, how many comparisons are saved just by ordering the block's or comparisons' execution and propagating the detected duplicates. This explains why Block Scheduling appears below Comparison Propagation in WF_2.

We can notice that methods targeting the repeated and superfluous comparisons (i.e., Block Scheduling, Comparison Propagation and Comparison Scheduling) have no effect on PC, although they significantly enhance PQ. The only methods that affect PC (apart from Block Purging) are Block and Comparison Pruning. We can compare their performance simply by contrasting the performance of WF_1 with that of WF_2. We can identify the following pattern across both data sets: Block Pruning has a negligible effect on PC, reducing it by less than 1.5%, whereas Comparison Pruning has a considerable impact on it, conveying a decrease of 5%. Thus, the latter sacrifices PC to a larger extent in favor of higher efficiency (i.e., PQ), involving around 50% less comparisons than Block Pruning. The main advantage of Comparison Pruning, though, is that it can be seamlessly combined with Compar-

		Compar. ($\times 10^8$)	**Duplic.**	**PQ** ($\times 10^{-4}$)	**PC** (%)	**Time** (hrs.)
	Block Purging	241.98	892,463	0.37	99.99	0.05
WF_1	Block Scheduling	15.55	892,463	5.74	99.99	0.16
	Block Pruning	0.72	879,446	121.44	98.53	0.01
WF_2	Comp. Propagation	123.74	892,463	0.72	99.99	5.75
	Block Scheduling	9.20	892,463	9.70	99.99	0.16
	Comp. Pruning	0.50	837,286	168.08	93.80	4.14
WF_3	Comp. Propagation	123.74	892,463	0.72	99.99	5.75
	Comp. Pruning	4.32	837,286	19.37	93.80	4.14
	Comp. Scheduling	0.45	837,286	187.63	93.80	0.51

Table 7.17: Perfomance of three different workflows over $D_{infoboxes}$, when applied on top of Block Purging and Trigram Graphs AC.

ison Scheduling (WF_3), which further reduces comparisons by around 10%, at no cost in PC.

Regarding the execution time of the workflows, we can notice the following patterns: WF_2 and WF_3 share almost the same time requirements across both data sets, with the latter taking slightly longer to complete its processing. On the other hand, WF_1 is around 100 times faster, due to its coarse granularity of functionality. Even in the worst case, though, WF_3 requires less than 12 hours for processing the 3 million entities of $D_{infoboxes}$. Among the individual blocking methods, Comparison Propagation and Comparison Pruning involve the most time-consuming processing. Compared to them, all other techniques require at most 1/10 of their processing time.

On the whole, both data sets advocate that WF_3 requires the lowest number of comparisons per entity, followed by WF_2 and WF_1. Judging from its PQ, around 25% (2%) of the comparisons it retains over D_{movies} ($D_{infoboxes}$) involve a pair of duplicates. Its substantially higher efficiency, though, comes at the cost of slightly lower effectiveness, as it detects around 4% less duplicates than WF_1. It also involves the highest execution time and consumes more resources, mainly due to Comparison Scheduling. For small data sets with millions of comparisons, the computational cost of WF_3 is affordable, thus constituting the best option. However, for large-scale applications with billions of comparisons, the choice depends on the performance requirements and the available resources of the application at hand. WF_1 is suitable for applications that have limited access to resources or are very strict with respect to effectiveness. Given that it involves the largest number of comparisons, it is particularly suitable for applications with small entity profiles that are efficiently compared. In this case, it can compensate for the higher number of comparisons it involves in comparison to WF_2 and WF_3. WF_2 lies in the middle of these two extremes, offering the same effectiveness with WF_3 at slightly lower blocking efficiency and time complexity.

7.7 Summary

Our thorough experimental analysis verified that, in the context of Clean-Clean ER, our Attribute Clustering algorithm significantly outperforms the basic Token Blocking approach as well as other, established clustering algorithms, such as EM. This applies to all representation models that were combined with AC, with the best performance corresponding to character n-gram graphs. In the context of Dirty ER, we validated that combining atomic URI Semantics blocking schemes into composite ones leads to a better balance between efficiency and effectiveness than Token Blocking. We also demonstrated that Block Purging conveys huge efficiency enhancements at a negligible cost in effectiveness under all settings, while *BC* and *CC* exhibit high correlation with *PC* and *PQ*, respectively. Another major outcome of our analysis is that meta-blocking is suitable for enhancing the efficiency of the input block collection(s) for all application requirements with respect to blocking effectiveness and resource consumption. Last but not least, we also proved that combining complementary block processing into ER workflows yields significantly higher efficiency for the desired levels of effectiveness.

Chapter 8

Conclusions

8.1 Summary

This dissertation introduced a novel methodology for tackling blocking-based Entity Resolution in the context of large-scale, highly heterogeneous, noisy data collections, which nowadays abound in the Web. Our approach extends the state-of-the-art blocking techniques in several ways. First, we distinguished blocking-based ER into three independent, but complementary subtasks, namely block building, meta-blocking and block processing. Individually, each task focuses on improving a specific performance aspect, i.e., either the effectiveness or the efficiency of the overall procedure. In combination, though, these tasks maximize both performance aspects. Second, we coined a rich diversity of blocking methods that are able to create blocks of high performance over HHIS, without taking any schema information into account. Their basic characteristic is that they consider (parts of) attribute values in order to create redundancy-positive block collections. Third, we demonstrated that meta-blocking enhances the efficiency of redundancy-positive block collections to a significant extent, by exploiting the information encapsulated in their block assignments. Fourth, we introduced several block processing techniques and categorized them according to a two-dimensional taxonomy that facilitates their independent use as well as their combination in workflows of higher performance. To this end, we also proposed specific guidelines that consider the available resources and the performance requirements of the underlying application. Last but not least, we introduced the *BC-CC* metric space, which facilitates the design of new blocking methods and allows for a-priori estimating their (relative) performance. All our techniques were thoroughly examined through an extensive experimental study that involved three large-scale, real-world data sets. Its outcomes validate that our methodology achieves an excellent balance between effectiveness and efficiency under varying settings.

8.2 Ongoing and Future Work

The following paragraphs discuss the most interesting aspects of our ongoing work and future research.

Parallelizing Blocking Techniques

The voluminous data of HHIS are likely to yield redundancy-positive block collections with excessive aggregate cardinality, despite the significant efficiency enhancements conveyed by our blocking techniques. As an example, consider the D_{BTC09} data set, which requires more than 5,000 comparisons per entity, after applying Block Purging to Total Description Blocking. In these settings, parallelization is indispensable for achieving reasonable running times. Our research in this field focuses on the MapReduce paradigm [DG08], which requires that the processing in every node is independent of the others, comprising a sequence of separate Map and Reduce phases. A large body of work has already examined how to integrate MapReduce with (blocking-based) ER [VCL10, KTR12b, KTR12a, KTR12c, KTR11]. None of them, though, has considered redundancy-positive block collections over HHIS. Part of our efficiency techniques for redundancy-positive block collections can be adapted to MapReduce in a straight-forward way, as they merely require an effective load-balancing method (e.g., Comparison Pruning and the node-centric meta-blocking). Our goal is to adapt the rest of them, as well, and to develop novel, specialized techniques that inherently support parallelization.

Constraints for ER

The ER constraints encapsulate the conditions that have to be satisfied by the outcome of Entity Resolution. There is a large body of work in this field, as the contribution of constraints to ER performance is twofold [CSGK07]:

- They enhance efficiency, by reducing the search space of the ER process.
- They enhance effectiveness, by ensuring that the intermediate as well as the final outcomes of the ER process satisfy specific quality requirements.

Several types of constraints have been proposed in the literature, with the most important ones pertaining to pairs of entity profiles [ARS09, DLLH03, FLM+11, WGM10, WBGM09], or to even larger groups [CSGK07, KWH+12]. These are typically applied after block building, during the execution of pairwise comparisons, or during the final processing of the output. It is possible, however, that some constraints can be incorporated into the creation of blocks in order to achieve two goals:

- Efficient check of constraints. The computational cost of expressing a constraint through a string similarity metric is typically larger than that of a semantically equivalent constraint that is expressed in terms of blocks.
- Higher blocking efficiency and effectiveness. This can be accomplished by partitioning large blocks into smaller ones. In this way, the aggregate cardinality of the block collection is reduced, and the entities placed in every block satisfy all relevant constraints, thus being more likely to match.

On the whole, the challenge here is to develop techniques for incorporating ER constraints into blocking schemes that are suitable for HHIS.

Incremental ER

Incremental ER is employed in two different cases:

- When there are frequent changes in the ER algorithm that has been applied to a given entity collection and, thus, its outcome has to be often updated.
- When there are limited resources in terms of time and/or computational resources.

In the former case, the goal is to derive the up-to-date outcome without applying the new algorithm to the entire entity collection. Instead, we can significantly reduce the computational cost by identifying the relation between the latest and the previous logic of the ER algorithm so as to perform the minimum necessary operations. This problem has been examined in [WGM10] with respect to the general form of ER. Given, though, that the logic of an ER algorithm can be expressed as a (series of) ER constraint(s), this problem is related to that of Section 8.2. Hence, our research goal is to extend the work described in Section 8.2 so as to incorporate evolving constraints into redundancy-positive block collections[1].

[1]In this respect, this version of Incremental ER is also related to our work in [PGN+11], which addresses the problem of blocking-based ER in the context of evolving entity profiles.

The second form of Incremental ER is referred to as *Pay-As-You-Go Entity Resolution* [WMGMar, MCD^{+}07]. The gist of this task is to converge as quickly as possible to the complete deduplication of the input entity collection(s). In other words, duplicate entities should be detected gradually, with the minimum cost in pairwise comparisons at every iteration, due to the limited amount of available time and/or computational resources. This problem has already been examined in the context of iterative blocking over HOIS [WMGMar]. Our goal is to complement this approach with incremental blocking techniques for HHIS. We also intend to investigate whether our Block and Comparison Scheduling techniques can be adapted to this task, e.g., by combining them in a more specialized methodology.

Bibliography

[ACG02] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

[Adl09] Noha Adly. Efficient record linkage using a double embedding scheme. In *DMIN*, pages 274–281, 2009.

[AMC07] R.B. Almeida, B. Mozafari, and J. Cho. On the evolution of wikipedia. In *ICWSM*, 2007.

[AO05] Akiko N. Aizawa and Keizo Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI*, pages 30–39, 2005.

[ARS09] Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, pages 952–963, 2009.

[BCC03] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *SIGKDD*, volume 3, pages 25–27, 2003.

[BdMNW12] C. Böhm, G. de Melo, F. Naumann, and G. Weikum. Linda: Distributed web-of-data-scale entity matching. In *CIKM*, 2012.

[BG06] Indrajit Bhattacharya and Lise Getoor. A latent dirichlet model for unsupervised entity resolution. In *SDM*, 2006.

[BG07] Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *TKDD*, 1(1), 2007.

[BHBLBL09] Christian Bizer, Tom Heath, Tim Berners-Lee, and Tim Berners-Lee. Linked data - the story so far. *IJSWIS*, 5(3):1–22, 2009.

[BKM06] Mikhail Bilenko, Beena Kamath, and Raymond J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96, 2006.

[Chr12a] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.

[Chr12b] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537–1555, 2012.

[CRF03] William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[CSGK07] Surajit Chaudhuri, Anish Das Sarma, Venkatesh Ganti, and Raghav Kaushik. Leveraging aggregate constraints for deduplication. In *SIGMOD*, pages 437–448, 2007.

[DG08] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DH05] AnHai Doan and Alon Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.

[DHM05] Xin Dong, Alon Y. Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pages 85–96, 2005.

[DLLH03] AnHai Doan, Ying Lu, Yoonkyong Lee, and Jiawei Han. Object matching for information integration: A profiler-based approach. In *IIWeb*, pages 53–58, 2003.

[DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society.*, pages 1–38, 1977.

[DN09] U. Draisbach and F. Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. In *QDB*, pages 51–56, 2009.

[dVKCC09] Timothy de Vries, Hui Ke, Sanjay Chawla, and Peter Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 1565–1568, 2009.

[dVKCC11] Timothy de Vries, Hui Ke, Sanjay Chawla, and Peter Christen. Robust record linkage blocking using suffix arrays and bloom filters. *TKDD*, 5(2):9, 2011.

[DZN12] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. Freeq: an interactive query interface for freebase. In *WWW (Companion Volume)*, pages 325–328, 2012.

[EIV07] Ahmed Elmagarmid, Panagiotis Ipeirotis, and Vassilios Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[FLM+11] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. Interaction between record matching and data repairing. In *SIGMOD*, pages 469–480, 2011.

[FS69] I.P. Fellegi and A.B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, pages 1183–1210, 1969.

[GD05] Lise Getoor and Christopher Diehl. Link mining: a survey. *SIGKDD Explorations*, 7(2):3–12, 2005.

[GIJ+01] Luis Gravano, Panagiotis Ipeirotis, H. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[GKVS08] George Giannakopoulos, Vangelis Karkaletsis, George A. Vouros, and Panagiotis Stamatopoulos. Summarization system evaluation revisited: N-gram graphs. *TSLP*, 5(3), 2008.

[GMP+12] George Giannakopoulos, Petra Mavridi, Georgios Paliouras, George Papadakis, and Konstantinos Tserpes. Representation models for text classification: a comparative analysis over three web document types. In *WIMS*, 2012.

[GP10] George Giannakopoulos and Themis Palpanas. Content and type as orthogonal modeling features: a study on user interest awareness in entity subscription services. *International Journal of Advances on Networks and Services*, 3(2), 2010.

[HFM06] Alon Y. Halevy, Michael J. Franklin, and David Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.

[HS95] Mauricio Hernández and Salvatore Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[HS98] Mauricio A. Hernández and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2(1):9–37, 1998.

[HSM08] Robert Hall, Charles A. Sutton, and Andrew McCallum. Unsupervised deduplication using cross-field dependencies. In *KDD*, pages 310–317, 2008.

[JLM03] Liang Jin, Chen Li, and Sharad Mehrotra. Efficient record linkage in large data sets. In *DASFAA*, pages 137–146, 2003.

[JW04] Ian Jacobs and Norman Walsh. Architecture of the world wide web, volume one. *W3C Recommendation*, December 2004.

[KL10] H. Kim and D. Lee. HARRA: fast iterative hashed record linkage for large-scale data collections. In *EDBT*, pages 525–536, 2010.

[KMC05] Dmitri V. Kalashnikov, Sharad Mehrotra, and Zhaoqi Chen. Exploiting relationships for domain-independent data cleaning. In *SDM*, 2005.

[KTR11] Lars Kolb, Andreas Thor, and Erhard Rahm. Block-based load balancing for entity resolution with mapreduce. In *CIKM*, pages 2397–2400, 2011.

[KTR12a] Lars Kolb, Andreas Thor, and Erhard Rahm. Dedoop: Efficient deduplication with hadoop. *PVLDB*, 5(12):1878–1881, 2012.

[KTR12b] Lars Kolb, Andreas Thor, and Erhard Rahm. Load balancing for mapreduce-based entity resolution. In *ICDE*, pages 618–629, 2012.

[KTR12c] Lars Kolb, Andreas Thor, and Erhard Rahm. Multi-pass sorted neighborhood blocking with mapreduce. *Computer Science - R&D*, 27(1):45–63, 2012.

[KWH+12] Georgia Koutrika, Ryan Wisnesky, Mauricio Hernandez, Rajaseka Krishnamurthy, and Lucian Popa. Hil: A high-level scripting framework for entity resolution and integration. In *Technical Report by IBM Research (RJ10499)*, 2012.

[LR96] AJ Lait and B. Randell. An assessment of name matching algorithms. *Technical Report*, 1996.

[MCD+07] Jayant Madhavan, Shirley Cohen, Xin Luna Dong, Alon Y. Halevy, Shawn R. Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[MK06] Matthew Michelson and Craig A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[MP80] W.J. Masek and M.S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

[MRS08] C.D. Manning, P. Raghavan, and H. Schuetze. *Introduction to information retrieval*, volume 1. Cambridge University Press, 2008.

[MW04] Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with application to noun coreference. In *NIPS*, 2004.

[NK62] Howard B. Newcombe and James M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, 1962.

[NMMMBLP07] Jordi Nin, Victor Muntés-Mulero, Norbert Martínez-Bazan, and Josep-Lluis Larriba-Pey. On the use of semantic blocking techniques for data cleansing and integration. In *IDEAS*, pages 190–198, 2007.

[Pap11] George Papadakis. Efficient entity resolution methods for heterogeneous information spaces. In *ICDE Ph.D. Workshop*, pages 304–307, 2011.

[PDKF10] George Papadakis, Gianluca Demartini, Philipp Kärger, and Peter Fankhauser. The missing links: Discovering hidden same-as links among a billion of triples. In *iiWAS*, pages 453–460, 2010.

[PGN$^+$11] George Papadakis, George Giannakopoulos, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Detecting and exploiting stability in evolving heterogeneous information spaces. In *JCDL*, pages 95–104, 2011.

[PIN$^+$11a] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL*, pages 85–94, 2011.

[PIN$^+$11b] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. To compare or not to compare: making entity resolution more efficient. In *SWIM*, 2011.

[PIN$^+$12] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Beyond 100 million entities: Large-scale blocking-based resolution for heterogeneous data. In *WSDM*, pages 53–62, 2012.

[PINF11] George Papadakis, Ekaterini Ioannou, Claudia Niederée, and Peter Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*, pages 535–544, 2011.

[PIP$^+$ar] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *TKDE*, 2013 (to appear).

[PKPNar] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. Meta-blocking: Taking entity resolution to the next level. *TKDE*, 2013 (to appear).

[PMM$^+$02] Hanna Pasula, Bhaskara Marthi, Brian Milch, Stuart J. Russell, and Ilya Shpitser. Identity uncertainty and citation matching. In *NIPS*, pages 1401–1408, 2002.

[PZ84] J.J. Pollock and A. Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4):358–368, 1984.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[RDG11] Vibhor Rastogi, Nilesh N. Dalvi, and Minos N. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4(4):208–218, 2011.

[SD06] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *ICDM*, pages 572–582, 2006.

[VCL10] Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD*, pages 495–506, 2010.

[WBGM09] Steven Euijong Whang, Omar Benjelloun, and Hector Garcia-Molina. Generic entity resolution with negative rules. *VLDB J.*, 18(6):1261–1277, 2009.

[WGM10] Steven Whang and Hector Garcia-Molina. Entity resolution with evolving rules. *PVLDB*, 3(1):1326–1337, 2010.

[WGM12] Steven Euijong Whang and Hector Garcia-Molina. Joint entity resolution. In *ICDE*, pages 294–305, 2012.

[Win06] William E Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Cencus, 2006.

[WMGMar] S. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *TKDE*, 2013 (to appear).

[WMK$^+$09] Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.

[WYP10] WE Winkler, WE Yancey, and EH Porter. Fast record linkage of very large files in support of decennial and administrative records projects. In *American Statistical Association*, 2010.

[YLKG07] Su Yan, Dongwon Lee, Min-Yen Kan, and C. Lee Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *JCDL*, pages 185–194, 2007.