N-Way Heterogeneous Blocking

Mayank Kejriwal  
University of Texas at Austin  
kejriwal@cs.utexas.edu

Daniel P. Miranker  
University of Texas at Austin  
miranker@cs.utexas.edu

Abstract—Record linkage concerns the linkage of records between two tabular datasets. To avoid naive quadratic computation, typical solutions employ a technique called blocking. A blocking scheme partitions records into blocks, and generates a candidate set by pairing records within a block. Current models of blocking have been restricted to two homogeneous datasets. The variety aspect of Big Data motivates heterogeneous record linkage; hence, the blocking of \( N \) heterogeneous datasets is a worthy problem. In this paper, we define a framework for N-Way heterogeneous blocking. Our model subsumes the current binary model. Within our model, a blocking scheme is defined on arbitrary numbers of heterogeneous relations, and shown to be a dependency relation. Necessary and sufficient conditions for blocking scheme transitivity are further proved. We use the framework to generalize two popular binary blocking methods, traditional blocking and sorted neighborhood, to N-ary. To avoid worst case quadratic cost in \( N \), the extended sorted neighborhood uses a novel dual windowing scheme. We show that a transitive blocking scheme enables the use of transitive closure algorithms to monotonically improve the candidate set till it is optimal. Thus, extended sorted neighborhood is shown to admit both qualitative and computational benefits.

Index Terms—N-Way heterogeneous Blocking, Record Linkage, Deduplication

I. INTRODUCTION

With the advent of Big Data [1], pairing objects that refer to the same underlying entity has become a pressing issue that has attracted interest from industry and academia alike [2]. However, the number of such pairwise comparisons grows quadratically both in the number and sizes of datasets that need to be linked. Blocking methods address this brute-force cost by efficiently selecting a small subset of pairs that are considered to be good candidates for subsequent comparison, while discarding the vast majority clearly non-coreferent [3]. This subset of pairs, referred to as a candidate set, is then input to a second (usually machine learning) technique to identify true matches. The full process, record linkage, has received widespread research attention\(^1\) and is comprehensively surveyed by Elmagarmid et al. [2].

A blocking method takes a blocking scheme and a set of tabular datasets as input, and partitions the datasets into blocks using the given scheme. Records within each block are paired in an algorithm-dependent manner and added to the candidate set, initially empty. Despite decades of research, the current model on blocking makes two restrictive assumptions. The first is that at most two tabular datasets are provided, while the second assumes the two datasets to be homogeneous, that is, they are assumed to contain identical attributes in their schemas. The variety facet of Big Data renders these assumptions problematic. Disregarding heterogeneity, current 2-Way blocking methods only mitigate quadratic cost of pairwise record comparisons given two relations. A straightforward decomposition of \( N \) relations into binary problems still incurs quadratic cost: \( O(N^2) \) times the cost of conducting blocking on each pair of relations. N-Way blocking proposes to mitigate this by considering holistic blocking on all relations. The model that we propose also relaxes the assumption of homogeneity. Thus, it addresses a broader class of problems more suited to the needs of the Big Data era.

We note that there is a precedent for similar extensions in other research areas. For example, holistic techniques have been considered for multi-class classification using Support Vector Machines, instead of decomposing the problem into \( O(N^2) \) binary classifications [4]. In data integration, holistic schema mapping was proposed recently [5]. These precedents motivated us to propose a framework for N-Way heterogeneous blocking. To the best of our knowledge, this is the first work to do so. Within the framework proposed in this paper, the 2-Way homogeneous case is subsumed as a special instance. Thus, previous work on the specialized problem is compatible with the proposed model.

We prove some key properties of the blocking schemes defined in the model. Specifically, every blocking scheme in the framework is proved to be a dependency relation, that is, reflexive and symmetric. Transitivity is shown to be guaranteed only if the set of blocks generated by the scheme satisfy some graph partitioning properties. Furthermore, extensions to popular blocking methods, traditional blocking and sorted neighborhood, are presented. A key sub-problem of sorted neighborhood, \( c \)-optimal ordering, is shown to be NP-hard. Thus far in the sorted neighborhood literature, this sub-problem has not been addressed. A motivating example also shows that a single windowing scheme, as used in the original sorted neighborhood, is quadratic in \( N \). To address this, the extended sorted neighborhood proposed employs a dual windowing scheme. It generates a candidate set linear in the total number of records placed within the blocks. In the MapReduce framework, a cost linear in the size of the largest block is guaranteed. A key proof shows that if the blocking scheme is transitive, then a candidate set can be monotonically improved by performing transitive closure on a graph abstraction of the problem. If the transitive closure

\(^{1}\)Different communities refer to the same problem by different names, e.g. entity resolution and co-reference resolution.
is allowed to naturally terminate, the resulting set is provably optimal, given that blocking scheme. In this way, extended sorted neighborhood is shown to have both computational and qualitative benefits.

The outline of the paper is as follows. Section II describes some related work in blocking. Section III describes the problem setting, and presents a model that allows for N-Way heterogeneous blocking. Section IV extends traditional blocking and sorted neighborhood previously applicable to 2-Way homogeneous blocking. Section V describes some open problems, and Section VI concludes the paper.

II. RELATED WORK

Blocking methods have been surveyed by Christen [3]. In that paper, numerous popular methods are covered, but the schemas of the relations involved are assumed identical. Also, at most two relations are considered. If exactly two relations are involved, the process is commonly referred to as record linkage; for a single relation, it is known as deduplication. Among the blocking methods surveyed in that paper that are extended in this paper are traditional blocking and sorted neighborhood. Traditional blocking has been in use since the 1960s [6]. The sorted neighborhood was originally proposed in a seminal work by Hernandez and Stolfo [7]. Several variants of the sorted neighborhood have also been proposed over the years [8], [9]. Unfortunately, all of these methods assume the 2-Way homogeneous case.

Blocking is the first step of a full record linkage system. A comprehensive survey was provided by Elmagarmid et al. on the full setup, with a major emphasis on the classification or clustering methods that constitute the second step, usually involving machine learning [2]. Many of these methods are based on the influential Fellegi-Sunter model [10]. Note that, despite the dearth of research in holistic blocking, some holistic schemes have been proposed for the schema matching domain of data integration, for example [11]. Those results are encouraging in that the issue is being addressed in a related field. Their work motivated us to attempt the same in blocking.

For some proofs in this paper, some standard results from computational complexity theory and algorithms will be used. A good reference for algorithms is the classic text by Leiserson et al. [12].

III. FRAMEWORK AND DEFINITIONS

A. Problem Setting

The problem setting for the proposed framework is tabular. Although entity resolution has been explored for multiple document types, blocking has been predominantly applied to the structured tabular setting. An example of a data model that corresponds to a tabular setting is the relational data model. Specifically, consider a relational database schema \( \mathcal{R} = \{R_i(a_1 \ldots a_m) | i \in \{1 \ldots N\}\} \) as a set of \(N\) relational schemas where each schema comprises the name\(^2\) \(R_i\) of the relation and its set of \(m\) attributes, \(\{a_1 \ldots a_m\}\). Note that \(m\) is not a constant but is dependent on the relation and could be different for different relations. We assume that every relational schema contain at least one attribute. Furthermore, each attribute \(a_k\) (of a relation \(R_i\)) is associated with a domain \(D_k\) of an elementary data type \(d_k\). An instance \(\bar{R}_i = \{r_j(a_1 \ldots a_m) | j \in \{1 \ldots |\bar{R}_i|\}\}\) of the relation \(R_i\) is a set of tuples, called records in this paper, with each tuple having a component \(\bar{a}_k\) for the \(k^{th}\) attribute \(a_k\) in the schema of \(R_i\).

For simplicity, denote \(r_j(\bar{a}_k)\) as the \(k^{th}\) attribute value of record \(r_j\).

We assume that in addition to the database schema \(\mathcal{R}\), a set \(\mathcal{R} = \{R_i | i \in \{1 \ldots N\}\}\) of \(N\) relational instances is also provided. Denote \(\mathcal{R}\) a collection of data sources, where each source is simply a relational instance. Note that exactly one instance is assumed for each relational schema. If this is not initially the case, a pre-processing supplementing step is performed on the database schema by copying\(^3\) a given relational schema with \(c\) instances \(c\) times and appending a unique identifier to the name of each copy.

The following definition formalizes what it means for \(\mathcal{R}\) to be homogeneous:

Definition 1. A database schema \(\mathcal{R}\) is homogeneous (otherwise heterogeneous) if either \(|\mathcal{R}| \leq 1\) or \(\forall R_i, R_j \in \mathcal{R}, i \neq j, R_i\) and \(R_j\) are homogeneous. Two relational schemas are homogeneous (otherwise heterogeneous) if their attribute sets are identical.

A database schema is heterogeneous if it contains even a single heterogeneous pair of relational schemas, by Definition 1. Note that the names of two homogeneous relations do not have to be the same, by this definition. Given the inputs \(\mathcal{R}\) and \(\mathcal{R}\), the problem of record linkage is to locate a set of record pairs with both records in each pair representing the same underlying entity despite syntactic differences. Note that records from two separate instances in \(\mathcal{R}\) are allowed to pair. Define such a pair to be heterogeneous, if the schemas of their corresponding instances are heterogeneous, per Definition 1. As noted earlier, blocking attempts to mitigate quadratic complexity of full pairwise comparison. The problem thus far has only been explored for homogeneous \(\mathcal{R}\) and with \(|\mathcal{R}| \leq 2\).

In this paper, these assumptions are relaxed. For simplicity and precision, the relational data model is assumed in the remainder of this paper. We note, however, that the concepts carry over to any tabular model. This is because, in no aspect, does the framework ever invoke any properties (e.g. referential integrity constraints or relational algebra) specific to relational data models per se. For notational brevity, we will often utilize the counting set \(\mathbb{Z}_p\) of integers for an integer \(p\). \(\mathbb{Z}_p\) is simply \(\{i | 1 \leq i \leq p, i \in \mathbb{Z}\}\), that is, the set

\(^2\)Technically, the symbols denoting the name of the relation and its attributes must be drawn from a countably infinite domain that does not include the (special) symbol NULL

\(^3\)As a special case, if the schema does not have a corresponding instance in \(\mathcal{R}\) we remove it from \(\mathcal{R}\)
of integers from 1 to \( p \) (inclusive).

**B. Blocking Schemes**

In this section, the formalism of N-way blocking is presented. Many of the concepts defined below are not completely new, but have been restricted thus far to 2-way homogeneous blocking. Rather than introducing new terminology, therefore, the focus of these definitions is to generalize existing concepts to adapt them for a richer model. Examples are provided to illustrate many of the definitions.

Prior to defining a blocking scheme, we introduce the concept of a **Generalized Indexing Function** (or GIF) \( h_i \), associated with relational schema \( R_i \). Intuitively, a GIF takes the attribute values of a record as input and outputs a set \( Y \) of **Blocking Key Values** (or BKVs). Each BKV essentially indicates the **block** to which that record will be assigned. Traditionally, an indexing function was defined on one attribute. Blocking predicates and schemes were then defined for pairs of records by combining these indexing functions using specific operators. This setup works well for the homogeneous case, but there is no obvious way to extend it to the heterogeneous case. Therefore, in order to enrich the current model, the indexing function is not constrained to a single attribute, but can now be defined on any number of attributes. An immediate advantage of generalization is that the function can now take attribute correspondences into account when computing its output [13]. It can also be arbitrarily complex, thereby admitting a broad range of cases.

Note that just like a traditionally defined indexing function, a GIF is many-many\(^4\) since each distinct input can be assigned more than one BKV; hence, the record can potentially be placed in many blocks.

**Definition 2.** A GIF \( h_i : D_1 \times \ldots D_m \rightarrow 2^{U^*} - \Phi \) takes as input attribute values \( a_1, \ldots a_m \) of some record \( r_j \) from instance \( R_i \) and returns a finite set \( Y \) that may contain 1 or more **Blocking Key Values** (BKVs) from the set of all possible BKVs \( U^* \).

The **range** of the indexing functions, or equivalently, the type of a BKV, can be any fixed datatype. Although the domain \((D_1 \times \ldots D_m)\) and definition \((h_i)\) of GIFs are both dependent on the relation \( R_i \), the range\(^5\) \( 2^{U^*} - \Phi \) is global and common to all GIFs defined for the database schema \( R \). The utility of this underlies the **blocking step**, which is described in detail in the subsequent section. Each relation \( R_i \) is constrained to have exactly one GIF \( h_i \). Furthermore, denote the set \( H \) to be the set of GIFs \( \{h_i\} \) where \( h_i \) is the GIF on relation \( R_i \) per Definition 2.

Although the GIF is defined to take all attribute values of a record as arguments, it might not be using them all in practice. For example, the GIF might only be tokenizing the first attribute of the relation, with each token an output BKV.

In such an instance, the only relevant attribute is the first, with the other attributes being irrelevant. This notion is formalized below.

**Definition 3.** An attribute \( a_k \) of a relation \( R_i \) is considered irrelevant (otherwise relevant) with respect to its GIF \( h_i \) if \( \forall a_k \in D_k \), the output of \( h_i \) is identical.

Attributes of a relation are only relevant or irrelevant with respect to a defined GIF. If the GIF for that relation were to be changed, the sets of relevant and irrelevant attributes could also change.

**Example 1.** An example of a GIF on an employees relational schema \( Employees(SSN, Gender, First Name, Last Name, Address) \) is \( Tokens(Address) \cup Tokens(\text{Last Name}) \cup \text{Last4Chars}(SSN) \), where \( Tokens \) is a function that tokenizes a string according to some special delimiters (like comma, space and slash) and \( \text{Last4Chars}(SSN) \) extracts the last four characters from the SSN attribute. The fixed datatype of the GIF (and hence, of Tokens and Last4Chars) is String; hence, \( U^* \) is the (countably infinite) set of all possible strings. Moreover, since attributes Gender and First Name are not getting used at all by this GIF, they are irrelevant, with the other attributes (SSN, Last Name, and Address), relevant.

There is an important class of GIFs, known as **disjoint GIFs** (d-GIF), that deserve special merit. A d-GIF can only return a set of cardinality exactly 1. Thus, each record gets mapped to only one BKV. A d-GIF, therefore, is explicitly many-one\(^6\). Given a d-GIF, the records of an instance can be partitioned so that there is no overlap between any of the partitions.

**Definition 4.** The **overlap** of a GIF \( h_i \) with respect to given instance \( R_i \) is given by \( \max|Y| \) with \( Y \) evaluated over all records in \( R_i \).

Note that the overlap of a GIF is dependent on the given instance \( R_i \).

**Example 2.** For the employees relation of Example 1, an example of a disjoint indexing function is \( \text{Last4Chars}(SSN) \), since a given record can only have one BKV output string associated with it. The indexing function \( \text{Tokens}(\text{Address}) \cup \text{Tokens}(\text{Last Name}) \cup \text{Last4Chars}(SSN) \), on the other hand, is not guaranteed to be disjoint since the BKV set of a record can contain several BKVs.

With the above definitions, a blocking scheme can now be defined. Recall that for brevity, the counting set of integers from 1 to \( N \) \( (\{1, \ldots N\}) \) is denoted \( \mathbb{Z}_N \).

**Definition 5. A blocking scheme** \( f((r, h_i), (s, h_j)) \) is a boolean predicate function that returns True iff \( h_i(\bar{r}) \cap h_j(\bar{s}) \neq \Phi \) and returns False otherwise, where \( i, j \in \mathbb{Z}_N \), \( h_i \) and \( h_j \) are the GIFs of schemas \( R_i \) and \( R_j \) respectively, and \( r \) and \( s \) are records in instances \( R_i \) and \( R_j \) respectively.

\(^4\)Technically, a GIF is classified as a multifunction or a set-valued function

\(^5\)Note that the range can be uncountably infinite. This would be the case, for example, if the fixed datatype is String, making \( U^* \) countably infinite and \( 2^{U^*} - \Phi \) uncountably infinite

\(^6\)A d-GIF, then, is a function in the proper mathematical sense since it is not multivalued any longer
Note that the blocking scheme is technically defined on the domain $M \times M$ where $M$ itself is the set of pairs $\bigcup_{i \in \mathbb{N}} \bigcup_{r \in R_i} \{(r, h_i)\}$. Thus, a blocking scheme is dependent on the set of GIFs $H$ and the collection of instances $R$.

Some simple properties that hold for all blocking schemes are now shown. In particular, it is proved that a blocking scheme is a dependency relation.

**Theorem 1.** A blocking scheme is always reflexive and symmetric.

**Proof:** We observe that $f((r, h_i), (r, h_j))$ always evaluates to True for some $i \in \mathbb{N}$, since Definition 2 states that $|h_i(r)| \geq 1$; hence, $|h_i(r) \cap h_j(r)| = |h_i(r)| \geq 1$, and the scheme always evaluates to True for the reflexive case. The scheme is also symmetric in inputs $(r, h_i)$ and $(s, h_j)$ because $h_i(r) \cap h_j(s) = h_j(s) \cap h_i(r)$.

Note that, in general, a blocking scheme does not have to be transitive. We show a counterexample in Example 3.

**Example 3.** Transitivity is violated for the case where $h_i(r) = \{\text{Smith, Jones}\}$, $h_j(s) = \{\text{Smith, Roberts}\}$ and $h_k(t) = \{\text{Roberts, Riley}\}$. $h_i(r) \cap h_j(s) = \{\text{Smith}\}$ while $h_j(s) \cap h_k(t) = \{\text{Roberts}\}$. However, $h_i(r) \cap h_k(t) = \emptyset$; hence, the blocking scheme is not transitive.

A special class of blocking schemes do form an equivalence class by always obeying transitivity. These are the blocking schemes that only accept d-GIFs as input. Although this is a sufficient condition, it is not necessary (for transitivity).

**Theorem 2.** $\forall i \in \mathbb{N}$, the disjointness of $h_i$ is sufficient for the transitivity of blocking scheme $f$.

**Proof:** Assume that $f((r, h_i), (s, h_j))$ and $f((s, h_j), (t, h_k))$ are both True. Also, $h_i, h_j$ and $h_k$ are disjoint for all records $r, s$ and $t$ in instances $R_i, R_j$ and $R_k$ respectively. Then, $|h_i(r) \cap h_j(s)| = 1$; hence, both $h_i(r)$ and $h_j(s)$ contain exactly the same element, designated $y_1$. Similarly, $h_j(s)$ and $h_k(t)$ contain exactly the same element, designated $y_2$. Since $|h_j(s)| = 1$, it must be the case that $y_1 = y_2 = y$ (say). Hence, $f((r, h_i), (t, h_k))$ evaluates to True since $h_k(t)$ and $h_i(r)$ have at least the element $y$ in common. This completes the proof.

The proof above shows that requiring all GIFs to be d-GIFs is too strong for transitivity. It was never really required in Theorem 2 for $h_i$ and $h_k$ to be disjoint; the proof would have still worked if only the intermediate argument $h_j$ was disjoint. Nevertheless, d-GIFs constitute an important class of indexing functions. As noted in Section IV, the basic version of the traditional blocking method explicitly requires its blocking scheme to only admit application of disjoint GIFs to constituent records. The necessary condition for guaranteeing transitivity is weaker, and is proved in the subsequent section, where the blocking step is described in detail.

**C. The Blocking Step**

Given a blocking scheme $f$, the blocking step of the record linkage process applies GIF $h_i$ to each record in instance $R_i$. This process is repeated for all instances. Because $N$ is finite, and each instance also contains a finite number of records, and also because we explicitly defined GIFs to always return finite sets of cardinality at least 1, the full set of unique Blocking Key Values $\Upsilon$ must be finite, and is given by:

$$\Upsilon = \bigcup_{i} \bigcup_{r \in R_i} h_i(r)$$

The unions in Equation 1 are valid because the range of all GIFs is common per the discussion following Definition 2. It was precisely for this reason that the GIFs were constrained to possess this global property.

Another important point to note is that a BKV $\upsilon$ can only exist ($\upsilon \in \Upsilon$) if there is at least one record that includes the key in its result set $Y$ when its associated GIF is applied to it. In other words, $\upsilon$ is guaranteed to reference at least one record. We say that $\upsilon$ uniquely identifies a block $B_\upsilon$, a non-empty set of records:

$$B_\upsilon = \bigcup_{i} \{r | r \in R_i, \upsilon \in h_i(r)\}$$

Records placed in the same block $B_\upsilon$ are said to co-occur in $B_\upsilon$. The full set of blocks $\Pi$ is the set of all generated blocks.

$$\Pi = \{B_\upsilon | \upsilon \in \Upsilon\}$$

Given $\Pi$ generated by a blocking scheme $f$, an optimal candidate set of pairs $\Gamma_o$ can further be generated:

$$\Gamma_o = \bigcup_{B \in \Pi} \{(r, s) | r, s \in B\}$$

Such a candidate set is denoted optimal because, given the present blocking scheme, this is the best that can be achieved. In practice, as explored in Section IV, generating $\Gamma_o$ can be prohibitive. In particular, Equation 4 implies that the BKVs referencing the largest blocks would take up a disproportional amount of time as compared to smaller blocks. Hence, it is often the case that the actual candidate set of pairs $\Gamma_c$ generated by the blocking method is suboptimal, $\Gamma_c \subset \Gamma_o$.

The following definition gives a notion for what it means for $\Pi$ to be transitive.

**Definition 6.** The full set of blocks $\Pi$ is transitive iff $\forall r, s, t$ where $r, s, t$ are distinct records, co-occurrence of $r$ and $s$ in block $B_a$, and co-occurrence of $s$ and $t$ in block $B_b$ together imply co-occurrence of $r$ and $t$ in some block $B_c$.

Definition 6 and Equation 4 together imply the intuitive result that iff $\Pi$ is transitive, then the occurrence of record pairs $\{r, s\}$ and $\{s, t\}$ in $\Gamma_o$ imply that $\{r, t\} \in \Gamma_o$.

Furthermore, Lemma 1 shows that the notion of transitivity per Definition 6 has an intimate connection to transitivity on

$^7$The definition only requires the records to be distinct. The blocks do not have to be distinct.
Lemma 1. II is transitive (per Definition 6) iff the blocking scheme \( f \) that generated it was transitive.

Proof: For the forward direction, assume II is transitive. Consider any three distinct records \( r, s, t \) such that \( r \) and \( s \) co-occur in a block \( B_r \) with BKV \( a \); this implies \( \{a\} \subseteq h_r(s) \cap h_r(t) \). Similarly \( s \) and \( t \) share BKV \( b \) and co-occur at least in block \( B_r \); hence, \( \{b\} \subseteq h_s(s) \cap h_s(t) \). By Definition 6, \( r \) and \( t \) must also co-occur in some block \( B_r \) with BKV \( c \); this would indicate \( \{c\} \subseteq h_r(r) \cap h_r(t) \). Thus, the blocking scheme would evaluate to True. The reverse direction is similar. Assume the blocking scheme is transitive and that \( f((r, h_1), (s, h_2)) \) and \( f((s, h_2), (t, h_3)) \) are both true. Then, \( r \) and \( t \) would also co-occur in some block, proving the claim.

Some powerful properties about transitivity can further be proved if we consider the graph abstraction of II, defined below.

Definition 7. The graph abstraction of the full set of blocks II is an undirected graph \( G_{II} = (V, E) \) that is constructed by mapping each distinct record that occurs at least once in some block in II to a distinct vertex \( v_r \) in \( V \), and by connecting two vertices \( v_r \) and \( v_s \) with an edge \( \{v_r, v_s\} \) iff \( r \) and \( s \) co-occur in at least one common block.

An interesting property to note is that for the graph abstraction of II, \( |V| = \Sigma_{R \in R} |R| \); in other words, every record in the instance collection \( R \) is represented by a vertex in \( V \). The reason why this is true is because Definition 2 requires a GIF to return at least one BKV for a record. Thus, a record must be placed in some block, and by Definition 7, it will be mapped to a vertex.

Analogous to Definition 7, the graph abstraction of a candidate set \( \Gamma_c \) is defined.

Definition 8. The graph abstraction of the candidate set \( \Gamma_c \) is an undirected graph \( G_{\Gamma} = (V, E) \) that is constructed by mapping each distinct record that occurs at least once in some record pair in \( \Gamma_c \) to a distinct vertex \( v_r \) in \( V \), and by connecting two vertices \( v_r \) and \( v_s \) with an edge \( \{v_r, v_s\} \) iff \( r \) and \( s \) co-occur in at least one common block.

We note that if \( \Gamma_c = \Gamma_o \), then \( G_{II} = G_{\Gamma} \).

Theorem 3. Consider the graph abstraction \( G_{II} \) of II per Definition 7. If II is transitive, the vertex set \( V \) of \( G_{II} \) can be partitioned exactly one way such that both conditions below hold:

1) Each partition is a maximal clique.
2) There is no edge between two vertices that fall in two different partitions.

Proof: For the forward direction, suppose II is transitive. Generate the optimal candidate set \( \Gamma_o \) per Equation 4. Recall that every vertex \( v_r \in V \) (the vertex set of \( G_{II} \)) was mapped from record \( r \). An edge exists between vertices \( v_r \) and \( v_s \) if the records \( r \) and \( s \) co-occur in some block in II, or equivalently, if the pair \( \{r, s\} \in \Gamma_o \). Initialize the partitioning by placing each vertex \( v_r \) in its own partition. We now apply the following rule iteratively: merge two partitions \( A \) and \( B \) if \( \forall v_r \in A, \forall v_s \in B, \{r, s\} \in \Gamma_o \). Thus, every vertex in a partition must have an edge to every vertex in the other partition for both to get merged. We note that because each partition starts with a single node, successful application of this rule implies that within a partition, every vertex is connected to every other by an edge. Thus, every partition represents a clique. We note, moreover, that the iterative procedure is guaranteed to terminate since, in each iteration, we only merge two partitions. This results in the cardinality of the partition set decreasing by 1 in each iteration. The minimum cardinality of the set is 1, which would indicate that \( G_{II} \) is a complete graph. Since we started with a partition set of cardinality \( |V| \), we can have at most \( |V| \) iterations. Thus, the merging rule above can only be applied finitely.

We prove by contradiction that exhaustively applying the merging rule guarantees that each partition represents a maximal clique. Suppose that the clique in partition \( A \) was not maximal. Then there exists a vertex \( v_s \) in some partition \( B \) such that there is an edge between every vertex in \( A \) and \( v_s \). Because of the merging rule above, \( v_s \) itself has an edge to every other vertex in \( B \). By transitivity, this implies that there is an edge between every vertex in \( A \) and every vertex in \( B \). But this implies \( A \) and \( B \) should be merged, which contradicts the assumption that we have exhaustively applied the merging rule. Note that a byproduct of this claim is that there cannot be an edge between two vertices \( v_r \in A \) and \( v_r \in B \) either, where \( A \neq B \). If there was, then by transitivity, every vertex in \( A \) would be connected to \( v_s \), and similarly, every vertex in \( B \) would be connected to \( v_s \). By the above proof, neither \( A \) nor \( B \) would then be maximal.

For the backward direction, assume we have a graph \( G \) (the graph abstraction of some II) with its vertex set \( V \) partitioned so that the two rules hold. We construct a candidate set \( \Gamma_o \) by initializing it as empty and adding a pair \( \{r, s\} \) to it if \( \{v_r, v_s\} \) co-occur in the same partition. Furthermore, we note that if \( v_r, v_s \) co-occur in some partition \( A \) and \( v_s, v_t \) co-occur in some partition \( B \) then it must be the case that \( A = B \). If not, it would imply \( v_s \) occurs in two partitions, which violates the definition of partitioning. But because each partition is a clique, it must be the case that there is an edge \( \{v_r, v_t\} \) in the partition; hence \( \{r, t\} \) gets added to \( \Gamma_o \). This proves that \( \Gamma_o \) is transitive. By Definition 6 and Equation 4, this also implies II is transitive.

As a last step, we show that two different partitionings of the same graph abstraction that obey the same two properties in the theorem statement, cannot exist. Since the partitionings are
different, but follow the two properties in the theorem, it must be the case that the two have at least one uncommon partition each. Designate a partition from the first partitioning \(A_1\) and from the second partitioning \(A_2\) respectively. Without loss of generality, choose \(A_1\) and \(A_2\) such that \(A_1 \neq A_2\) and \(\max|A_1 \cap A_2|\) are both true. Hence, \(A_1\) and \(A_2\) have maximum overlap of vertices. Note that if this maximum overlap evaluates to 0, then that implies the partitionings are equal. Since this is not the case, \(A_1\) and \(A_2\) contain at least one common vertex, say \(v_1\), in their intersection. Moreover, because they are unequal, at least one of the partitions (say \(A_1\)) contains a vertex \(v_2 \notin A_2\). Then, \(v_2\) must be in some other partition in \(A_2\). Because of the second property, there cannot be an edge between \(v_1\) and \(v_2\) per the second partitioning. However, \(v_1\) and \(v_2\) do share an edge per the first partitioning, because of the first property. Since both partitionings were derived from a common graph abstraction, this yields a contradiction. Thus, at most\(^8\) one partitioning of the graph abstraction can exist, that obeys the two properties. If \(\Pi\) if transitive, exactly one such partitioning can exist. This completes the proof.

D. Metrics

In this section, metrics for evaluating blocking schemes are described. This evaluation implicitly covers the evaluation of GIFs also, since a blocking scheme is associated with a set of these functions, one for each relation. As shown in Section 5, the choice of this set determines some important properties of the blocking scheme, particularly with respect to transitivity. To gain some intuition into the tradeoffs involved, define the full set of pairs \(\Omega\) that would be generated in the absence of blocking. Such a set would simply pair every record with every other record; hence, every possible pair of records would be generated for subsequent evaluation:

\[
\Omega = \bigcup_i \bigcup_j \{\{r,s\} | r \in \mathcal{R}_i, s \in \mathcal{R}_j, r \neq s\}
\]

Note that \(i\) and \(j\) do not have to be distinct, but a record is not allowed to pair with itself (\(r \neq s\)). Recall that if the schemas of \(R_i\) and \(R_j\) are homogeneous (guaranteed for \(i = j\)) then the pairing of their constituent records is also homogeneous per Definition 1 and the discussion following it. It is assumed that \(\Omega\) can be partitioned into a set of true positives (duplicates) \(\Omega_m\) and true negatives (non-duplicates) \(\Omega_n\). Both conditions below are required to hold:

\[
\Omega = \Omega_m \cup \Omega_n
\]

\[
\Phi = \Omega_m \cap \Omega_n
\]

Intuitively, these conditions state that given two records, they either refer to the same entity or they do not. For \(N\) relations, each containing at least \(t\) records, the cardinality of \(\Omega\) would be at least \(t(N^2) \approx O((tN)^2)\). Traditionally, blocking was proposed as an improvement over quadratic complexity in the number of records in the smaller relation with at most two relations \((N = 2)^9\). In the general case, the complexity of generating \(\Omega\) is quadratic in the total number of records, as demonstrated above.

Usually, there is a tradeoff between ensuring that true positives get included in the candidate set \(\Gamma_c\) generated by a blocking method\(^10\) and between cutting down on its size. Another scenario often desired is when the blocks are dense in duplicates; that is, the blocks contain many (ideally all) true positives and few (ideally none) true negatives. Three metrics Pairs Quality, Reduction Ratio and Pairs Completeness have commonly been employed to express these conflicting goals \([3], [14]\).

The metric Pairs Completeness \(\Lambda\) measures how many of the true positives are in \(\Gamma_c\) versus those \((\Omega_m)\) in the full of pairs \(\Omega\):

\[
\Lambda = \frac{|\Omega_m \cap \Gamma_c|}{|\Omega_m|} \quad (8)
\]

\(\Lambda\), therefore, captures the notion of recall in the framework of blocking.

The Reduction Ratio \(\Xi\) is a quantification of how well the blocking scheme minimizes the number of generated candidate pairs. It can be expressed by the formula:

\[
\Xi = 1 - \frac{|\Gamma_c|}{|\Omega|} \quad (9)
\]

To understand what the optimal value of \(\Xi\) should be, assume that the \(N\) relations contain \(|Q|\) entities between them where \(Q = \{q_i | i \in \mathbb{Z}_{|Q|}\}\) is the set of entities. Each entity is modeled as a cluster with \(|q_i|\) records in entity \(i\). The best pairwise reduction ratio \(\Xi_o\) achievable is:

\[
\Xi_o = \frac{\sum_q |q_i| \cdot (|q_i| - 1) / 2}{|\Omega|} \quad (10)
\]

Note that this is the best \(\Xi\) achievable under the constraint of perfect recall (see Equation 8). The third metric, Pairs Quality \(\Theta\) is given by the equation:

\[
\Theta = \frac{|\Omega_m \cap \Gamma_c|}{|\Gamma_c|} \quad (11)
\]

Pairs Quality comes closest to expressing precision in a blocking framework, although the semantics are different\(^11\).

Intuitively, \(\Theta \approx 1.0\) indicates that the blocks are extremely dense in duplicates. The F-Score \(F\) or the \(f\)-measure between any two quantities can be used to express dual tradeoffs. For example, the \(f\)-measure \(F_{\Lambda \Theta}\) can be used to express the analog of typical precision-recall tradeoff in this framework:

\[
F_{\Lambda \Theta} = \frac{2 \cdot \Lambda \cdot \Theta}{\Lambda + \Theta} \quad (12)
\]

Other combinations are also possible, e.g., \(F_{\Lambda \Xi}\).

A natural premise to consider is what constitutes an optimal

\(^8\)If \(\Pi\) is not transitive to begin with, no such partitioning will exist, by the first part of the proof.

\(^9\)Assuming there are no duplicates within a relation.

\(^10\)Typically, \(\Gamma_c\) is evaluated instead of \(\Omega_m\), as \(\Gamma_o\) is dependent only on the scheme and not the method. Evaluating \(\Gamma_c\) allows one to evaluate the blocking scheme and method in tandem.

\(^11\)Blocking is a preprocessing step to overall record linkage. Precision and recall can formally be evaluated only after the full record linkage terminates.
set of blocks \( \Pi \), that is, the best case scenario. For an optimal set, all three metrics achieve their respective optima, that is \((\Xi, \Lambda, \Theta) = (\Xi_0, 1.0, 1.0)\). For such a set, we observe that it would render subsequent record linkage unnecessary, since each block represents its own entity. Moreover, no two records spanning two different blocks refer to the same entity.

In practice, this is almost never the case. To compound the problem, the optimal value for the reduction ratio is typically unknown.

IV. BLOCKING METHODS

In this section, blocking methods for N-Way heterogeneous blocking are proposed. Throughout this section, the provision of GIFs, one for each relation, is assumed, possibly by a domain expert. All algorithms in this section will accept the database schema \( \mathcal{R} \), the collection of instances \( \bar{\mathcal{R}} \) and the set of GIFs \( H \) as input, and will output the candidate set of pairs \( \Gamma_c \). The candidate set may not always be optimal \((\Gamma_c \subset \Gamma_o)\) for some algorithms, per the discussion in Section III-D. The complexity of each method will be the size of the candidate set generated \(|\Gamma_c|\). This is because \( \Gamma_c \) is the input to the second, and more expensive step of record linkage [2]. This second step is normally assumed to dominate the time complexity of the entire process. The candidate set of pairs input to this step, therefore, is the strongest indicator of the performance of the blocking step, all else remaining the same [3].

A. Traditional N-Way Blocking

Traditional blocking is the simplest of existing blocking schemes. It assumes that the GIFs are disjoint (see Definition 4); a d-GIF places each record in exactly one block. A natural extension to this scheme is to loosen disjointness and allow each record to be placed in multiple blocks. Both methods have straightforward extensions to the case of \( N \) relations. The pseudocode of (extended) traditional blocking for the generic case is shown in Algorithm 1. Note that the blocks in \( \Pi \) are heterogeneous, since records from different relations might share blocking key values.

Algorithm 1 can also be written as a MapReduce program [15]. The Map step will apply the appropriate GIF to each record and essentially execute lines 1-4. Each BKV in the resulting output set will be a key for the record, with the record itself being the value. Note that for extended traditional blocking, there is a potential for record replication in the shuffling step, since records could have multiple BKVs. A loose upper bound on the replication factor is the maximum overlap of any of the given GIFs. To limit this cost, low overlap indexing functions should be preferred. In the Reduce step, each block will be collected, and pairs within that block generated (lines 5-6) and stored (line 7) in the Distributed File System (DFS) [15].

An analysis of Algorithm 1 shows that \(|\Gamma_c|\) depends on the distribution of BKVs. With a total of \( n \) records and \( b \) possible BKVs, the resulting set of \( b \) blocks \( \Pi \) would have exactly \( \frac{n}{b} \) records in each block. The more skewed the distribution, the worse traditional blocking would perform [3]. The complexity would only increase if traditional blocking was extended with records potentially placed in multiple blocks.

For the N-Way heterogeneous case, the analysis can be extended similarly, if the total number of possible BKVs \( b \) is known a priori. In practice, it is difficult to place guarantees on either \( b \) or the frequency of the \( b \) BKVs. In the worst case, traditional blocking (both basic and extended) can have a complexity of \( O(n^2) \), with \( n \) the total number of records in \( \mathcal{R} \); the advantage of blocking is null if this worst case is achieved. Intuitively, the worst case can occur if every record shares at least one common BKV.

Note that the analysis above does not assume the number or homogeneity of relations. The worst case depends only on the total number of records. Without explicit guarantees, therefore, the method is not recommended for large datasets. Nevertheless, it continues to be popular because it has a simple implementation [3].

B. Sorted Neighborhood

Sorted neighborhood was proposed to address the complexity shortcomings of traditional blocking [7]. The original sorted neighborhood method mitigated the problem by

---

**Algorithm 1 Traditional N-Way Heterogeneous Blocking**

**Input:**
- A database schema \( \mathcal{R} \)
- A collection of instances \( \bar{\mathcal{R}} \)
- A set of Generalized Indexing Functions \( H \)

**Output:**
- A candidate set of record pairs \( \Gamma_c \)

**Method:**
1) Initialize \( \Pi \) as an empty map with \((BKV, Block)\) entries
2) Initialize empty candidate set of record pairs \( \Gamma_c \)
3) for all instances \( \mathcal{R}_i \in \bar{\mathcal{R}} \) do
   for all records \( r \in \) in instance \( \mathcal{R}_i \) do
     
     if \( \Pi \) does not contain key \( y \) then
       Initialize empty Block \( B \)
       Place key-value pair \( (y, B) \) in \( \Pi \)
     end if
   end for
4) end for
5) for all \((y, B) \in \Pi \) do
   Generate all record pairs \( \{r, s\} \) within \( B \)
   if \( \{r, s\} \notin \Gamma_c \) then
     Add \( \{r, s\} \) to \( \Gamma_c \)
   end if
6) end for
7) Output \( \Gamma_c \)
first sorting records using their BKVs as the sorting key. Then, a c-window (of pre-specified size c) is slid over the sorted list. Records co-located in a c-window are paired and added to the candidate set. The assumption was that similar records would have similar BKVs. By choosing c reasonably, a high quality candidate set could be generated in time \(O(n\log(n) + (n-c)(c-1)+c^2)\) with \(n\) the total number of records. The first term refers to the sorting cost while the rest represent the cost of generating pairs by sliding the c-window over the entire list. A detailed derivation is provided by Christen [3]. Accuracy is further shown to improve by repeating the procedure for different blocking schemes and collecting results [7]. Each pass increases the run-time in practice, but bounding the number of passes by a constant precludes an asymptotic effect. Passes could also be parallelized in independent runs. Lastly, several proposed variants of the original method have augmented performance even further [3].

A subtle problem occurs in the sorting step of sorted neighborhood. Assume a situation where there are \(q > c\) records with the same BKV. The issue is the ordering of these \(q\) records so that the candidate set subsequently generated by the c-windowing scheme is optimal. In other words, no other ordering yields a better candidate set, according to some given similarity function. Abstractly, the problem is to discover a c-optimal ordering given a set of \(q_s\) objects and a symmetric similarity function \(S\), computable in constant time.

**Definition 9.** Given a constant \(c > 1\), a constant time symmetric similarity function \(S: R \times R \rightarrow \mathbb{Z}^+ \cup \{0\}\) where \(R\) is a set of objects, and an input set of objects \(q_s \subseteq R\), a c-optimal ordering of \(q_s\) is an ordered list \(q_i\) of the objects such that its c-windowing score, computed according to Algorithm 2, is maximized.

Given the ordered list, Algorithm 2 essentially calls the windowing subroutine (or second step) of sorted neighborhood. The score \(Q\) is then evaluated for the candidate set \(\Gamma\) generated. Note that Algorithm 2 has complexity \(O(\min(c^2,|q_i|)) = O(|q_i|)\), since \(c\) is constant. If \(|q_i| < c\) all pairs, bounded above by \(c^2\), are generated. If \(|q_i| \geq c\), the usual c-windowing scheme applies. Since the window can only be in \(|q_i| - c + 1\) positions, and each window yields constant number of pairs, Algorithm 2 runs in time \(O(|q_i|)\). Additionally, Algorithm 2 is abstract in that it admits a generic list of objects \(q_i\) and similarity function \(S\).

Theorem 4 shows that even a 2-optimal ordering of the set \(q_s\) is NP-hard. The claim is proved by reducing from the Traveling Salesman Problem [12]. Theorem 5 shows the general problem is as hard, by reducing from Theorem 4 for any \(c\) such that \(|q_s| > c > 1\).

**Theorem 4.** A 2-optimal ordering of a set of objects \(q_s\) is

\[\text{Algorithm 2 Computing c-Windowing Score for a list } q_i\]

**Input:**
- A list of \(q_i\) objects
- A window constant \(c\)
- A constant time computable similarity function \(S: R \times R \rightarrow \mathbb{Z}^+ \cup \{0\}\)

**Output:**
- An integer valued c-windowing score \(Q\)

**Method:**

1. Initialize empty candidate set of pairs \(\Gamma\)
2. Initialize c-windowing score \(Q\) to 0
3. \(\Gamma = \{\{q_i, q_j\}|q_i \neq q_j \land q_i, q_j \in q_i\}\)
   Goto line 8
4. end if
5. for all \(i \in \{1, \ldots |q_i| - c + 1\}\) do
   for all \(j \in \{i + 1, \ldots i + c - 1\}\) do
   \(\Gamma = \Gamma \cup \{\{q_i, q_j\}\}\)
   end for
6. end for
7. \(\Gamma = \Gamma \cup \{\{q_i, q_j\}|q_i \neq q_j \land i, j \in \{|q_i| - c + 2, \ldots |q_i|\}\}\)
8. for all \((m, n)\) do
   \(Q = Q + S(m, n)\)
9. end for

Proof: We first show a polynomial time reduction from the undirected Traveling Salesman path problem (TSpP) to the 2-optimal ordering. Namely, a complete, undirected graph with positive integer edge weights is given as input. The problem is to find the shortest path such that every vertex in the graph is on the path, but no vertex is repeated. TSpP is polynomial-time reducible from the Hamiltonian path problem; the reduction is analogous to the one for the tour problem TSP (recall that TSP is reduced from the Hamiltonian cycle problem) in Chapter 34 of the standard text [12].

To reduce TSpP to an instance of 2-optimal ordering, construct the set of objects \(q_i\) by mapping each node in the graph to an object. For each pair of objects \((m', n')\), the value of the similarity function is \(W_E - W(m, n)\) where \(m, n\) are the corresponding nodes in the graph, \(W\) is the weight of the edge between them and \(W_E\) is the sum of all the edge weights in the graph. Technically, the similarity function is being constructed discretely (for each possible pair of inputs), thus in a compact functional form. Since the number of such discrete inputs is \(O(N^2)\) where \(N\) is the total number of nodes in the graph, the reduction is polynomial time. The list returned by the 2-optimal ordering oracle \(q_i\) is converted to an optimal solution to the original TSpP instance, if we interpret the ordered list as a path from the first node in the list to the last. Because of the way the similarity function values were set up above, a maximum score obtained by 2-optimal ordering corresponds to a minimum score for the TSpP. The converted answer, therefore, is a minimum weight path that covers all vertices exactly once, which is the solution we were seeking. ■

---

12That is, a c-window contains c contiguous records
13However, Algorithm 2 never pairs an object with itself. In this respect, \(\Gamma\) in Algorithm 2 is like a generic version of the candidate set of record pairs earlier encountered.
Theorem 5. A c-optimal ordering of a set of objects $q_i$ is NP-hard for all $|q_i| > c > 1$. For $c \geq |q_i|$, every ordering (of the set of $|q_i|$ orderings) is optimal and has score $\sum_{q_i,q_j} Sim(q_i,q_j)$.

Proof: See Appendix for the first part. For the second part, because $c \geq |q_i|$, line 3 in Algorithm 2 will always apply, regardless of the input list $q_i$. Thus, the full set of pairs, given by $O(c^2)$, is generated. Line 8 sums the similarity scores of all these pairs, which is precisely $\sum_{q_i,q_j} Sim(q_i,q_j)$.

Current work using 2-Way homogeneous sorted neighborhood has not explicitly addressed finding approximate optimal orderings. We assume, therefore, that either an unstated heuristic is employed or worse, sub-BKV orderings are random. Presumably, the effects of a sub-optimal ordering are assumed small, or otherwise mitigated sufficiently by multi-pass blocking.

We argue that, for N-Way heterogeneous blocking, sub-BKV orderings are expected to be more common, particularly as N increases. Employing good heuristics for the generic problem is a necessity, therefore. We conjecture that the problem does not exhibit strong inapproximability and leave for future work to prove (or disprove) the claim. Regardless, developing practical heuristics warrants attention.

A related problem that precludes a straightforward extension of homogeneous sorted neighborhood is illustrated by the following scenario. Suppose that (on average) $p$ records in each instance $R_i$ share the common BKV $v$. Then, the heterogeneous block $B_v \in \Pi$ will contain $|N_p|$ records. Partition this block into a set of $N$ homogeneous sub-blocks, $\{B_{v,i} | i \in \mathbb{Z}_N\}$ with sub-block $B_{v,i}$ containing, on average, $p$ records from instance $R_i$. If $p > c$, and the records in the block are sorted so that records from the same instance are closer together, the generated pairs would be overwhelmingly homogeneous and also (potentially) highly sub-optimal. This situation resembles one where each instance essentially undergoes independent deduplication. There would be many cases of records from different instances not getting paired at all, if their associated sub-blocks are situated far apart in the ordering. The resulting candidate set would not be diverse enough to enable good (subsequent) record linkage.

A more sophisticated windowing scheme is clearly necessary. An obvious extension is to use the sliding window to explicitly account for heterogeneity. Consider the example above. One could slide the c-window over records of N sub-blocks independently and simultaneously. In each windowing iteration, the combined $Nc$ records in all the N c-windows would be paired with each other. The pair generation complexity of one iteration alone would thus be $O((Nc)^2)$. The conclusion is that a single windowing parameter employed this way avoids the worst case results of traditional blocking within a sub-block, but not across sub-blocks of a given block. To obtain this guarantee, a second windowing parameter $d$ is proposed. The

---

Algorithm 3 N-Way Heterogeneous Sorted Neighborhood

**Input:**
- A database schema $\mathcal{R}$
- A collection of instances $\mathcal{I}$
- A set of Generalized Indexing Functions $H$
- Homogeneous window size $c$
- Heterogeneous window size $d$

**Method:**

1. Initialize $\Pi$ as an empty map with (BKV, Block) entries
2. Initialize empty candidate set of record pairs $\Gamma_c$
3. for all instances $\mathcal{R}_i \in \mathcal{I}$ do
   - for all records $r$ in instance $\mathcal{R}_i$ do
     - Let $Y := h_i(r)$
     - for all Blocking Key Values $y \in Y$ do
       - if $\Pi$ does not contain key $y$ then
         - Initialize empty Block $B$
         - Place key-value pair $(y, B)$ in $\Pi$
       - end if
     - end for
   - end for
- for all $(y, B) \in \Pi$ do
  - Empty each list in $L$
  - for all records $r \in B$ do
    - Determine which relation $r$ is from $R_i$
    - Place $r$ in $L_r$ where $r \in R_i$
  - end for
  - Sort each sub-block in $L$ using $C$
  - Sort List $L$ using $D$
  - for all $i \in \{1, \ldots, |L|\}$ do
    - if $i \neq |L|$ then
      - Slide a c-window simultaneously over sub-blocks $L_i$ and $L_j$
      - Pair every record in the first window with every record in the second window
      - Add (these heterogeneous) pairs to $\Gamma$
    - end if
  - end for
  - Generate pairs between records within $i$’s c-window
  - Add (these homogeneous) pairs to $\Gamma$
- end for
- end for
4. end for

---

14That is, ordering records that have the same BKV
15The original paper does not even address the question of whether the sorting algorithm is stable
ordering of sub-blocks within a heterogeneous block. Because the problem is defined generically, both instantiations are technically valid.

The first six lines apply the appropriate GIF to each record of each relational instance and generate the set II of heterogeneous blocks. Next, a list $\mathcal{L}$ of homogeneous sub-blocks is constructed. For an illustration of the dual windowing process, see Figure 1. The worst-case run-time analysis of Algorithm 3 is not straightforward. The first six steps require $O(\Sigma_i |R_i|)$ akin to traditional blocking. For the overall algorithm, Theorem 6 claims that worst-case pair generation for a given block is linear in the size of the block:

**Theorem 6.** Algorithm 3 generates at most $O(|B| + N)$ pairs for block $B$, with $|B|$ being the total number of records in that block.

**Proof:** In the most general case, block $B$ can be partitioned into $N$ homogeneous sub-blocks $\{B_{i,j} \mid i \in \mathbb{Z}_N\}$ with sub-block $i$ containing $|\hat{R}_i| \geq p_i \geq 0$ records from instance $\bar{R}_i$, and with $\Sigma_i p_i = |B|$. Given heuristics $C$ and $D$ in Algorithm 3, an ordering is imposed both on the array of sub-blocks (denoted outer ordering) as well as records within each sub-block (inner ordering). Note that the ordering (equivalently, the effectiveness of the heuristics) will potentially affect quality metrics like $\Lambda$ and $\Theta$ of $\Gamma_c$ from Section III-D but not the cardinality (in other words, they do not affect $\Xi$).

Consider the first $d$ sub-blocks in the outer ordering. For the first $c$ records in all sub-blocks within this window, we generate record pairs between the first $c$ records of the first sub-block and the first $c$ records of the other $d-1$ sub-blocks. This results in exactly $c^2(d-1)$ pairs. We advance the inner sliding window by a single record within each of these $d$ sub-blocks. Note that we do not need to repeat the first step and generate $c^2(d-1)$ pairs again because there is an overlap of exactly $c-1$ records between the inner $c$-windows, since we only slid by one record. Thus, the new pairs generated between the first sub-block and the other $d-1$ sub-blocks will be $c(d-1)$. This same argument applies till the inner window reaches either the end of the shortest sub-block within the current $d$-window or the end of the first sub-block. With $p_1$ records in the first sub-block, there are exactly $p_1 - c + 1$ c-window positions. The first window yields $c^2(d-1)$ unique pairs, while the other $p_1 - c$ windows add $c(d-1)$ unique pairs, for a total of $c^2(d-1) + (p_1 - c)(c(d-1)) = p_1 c(d-1)$ unique pairs. Note that pairs were generated between records of only the first sub-block and the other $d-1$ sub-blocks, and not between all the sub-blocks within that $d$-window. Therefore, the outer window can advance by only one sub-block in the next iteration. In this second (outer) iteration, the procedure above is repeated. By the same argument as for the first sub-block, $c^2(d-1) + (p_2 - c)(c(d-1)) = p_2 c(d-1)$ unique pairs will be generated between the records of the second sub-block and the next $d-1$ sub-blocks. In fact, this procedure will apply to the first $N - d$ outer iterations. Note that the $(N - d + 1)^{th}$ (in other words, last) outer windowing position is special in the same way that the first inner windowing position was special for each of the sub-blocks. To make the analysis easier, let us augment the $N$ sub-blocks so that there are now $N + d - 1$ sub-blocks. The extra $d - 1$ sub-blocks that we have added have no corresponding instances, and are thus empty. We do not have to treat the $(N - d + 1)^{th}$ iteration in a special way, anymore. Instead, we continue sliding the outer window forward and generate a loose upper bound of $c^2(d-1) + (p_i - c)(c(d-1)) = p_i c(d-1)$ pairs per iteration, with $i$ the position of the first sub-block (in the block) in the current $d$-window. We continue till $i = N$. In the $N^{th}$ iteration, only the $N^{th}$ sub-block is (possibly) non-empty, the following $d - 1$ sub-blocks were constructed to be empty. At this point, we can terminate the procedure. The upper bound is loose because after the $(N - d + 1)^{th}$ iteration, we start encountering increasing numbers of empty sub-blocks at the end of the window. For example, the number of unique pairs generated for the $(N - d + 2)^{th}$ iteration is $c^2(d-2) + (p_{N-d+2} - c)(c(d-2)) = p_{N-d+2} c(d-2)$. Thus, the scheme above overestimates constant coefficients. This does not change the asymptotic analysis. In fact, neglecting the constant coefficients, we conclude that $\Sigma_i p_i$ heterogeneous pairs get generated, if we sum over the pairs generated for each iteration. Because $\Sigma_i p_i = |B|$, at most heterogeneous $O(|B|)$ pairs get generated for this block. As for homogeneous pair generation within each sub-block, we note that Algorithm 3 generates homogeneous pairs by pairing every record with every other record in the same $d$-window for the same sub-block. For the $i^{th}$ sub-block, the number of homogeneous pairs generated will therefore be $c(c-1)/2 + (c-1)(p_i - c)$. Asymptotically, this is $O(C + p_i)$ where $C$ equals the (constant) first term. Summing over all sub-blocks and adding to the expression obtained for heterogeneous pair generation complexity, the cardinality $O(|B| + N)$ is obtained for $\Gamma_c$. ■

Algorithm 3 can also be written in MapReduce. The Map step is identical to that proposed for Algorithm 1. The Reduce step collects all records of a heterogeneous block and generates pairs per the dual windowing scheme in Algorithm 3. Since each block is processed in its own Reduce task, the complexity is $O(max(|B|))$, by Theorem 6. An upper bound on this quantity is the total number of records in $\mathcal{R}$. Thus, the
Algorithm 3 is overlap. This is the same as a single linear scan of the full database, empty BKV set when applied to a record (see Definition 2). This is the same as a single linear scan of the full database, which is unavoidable. An upper bound (of the serial run) would depend again on the overlap of GIFs. Thus, a common way of limiting costs of both Algorithms 1 and 3 is to bound overlap of GIFs.

Given that sorted neighborhood focuses aggressively on Reduction Ratio $\Xi$ (see Section III-D) at the expense of Pairs Completeness $\Lambda$, an interesting issue arises if a cost $\Xi_{\text{max}}$ is specified, with the aim of maximizing $\Lambda$ under the constraint $\Xi \leq \Xi_{\text{max}}$. If the $\Xi$ achieved by the algorithm is less than $\Xi_{\text{max}}$ the sorted neighborhood was much too aggressive. One way to mitigate this shortcoming if the blocking scheme is transitive is to compute the transitive closure of $\Gamma_c$ incrementally till $\Xi = \Xi_{\text{max}}$. Transitive closure algorithms have been well studied for decades [16]. These algorithms can be applied on the graph abstraction $G_\Gamma$ of $\Gamma_c$, constructed according to Definition 8. Lemma 2 shows that a path exists between every pair of vertices $v_r$ and $v_s$ in $G_\Gamma$ if records $r$ and $s$ co-occur in at least one block. Lemma 3 shows that each time the transitive closure adds a new pair, incrementally to $\Gamma_c$, the condition $\Gamma_c^+ \subseteq \Gamma_o$ is assured, where $\Gamma_c^+$ is the incremented candidate set. The resulting Theorem 7 uses Lemmas 2 and 3 to guarantee that the end result of transitive closure is the graph abstraction of $\Gamma_o$.

**Lemma 2.** Consider a candidate set $\Gamma_c$ constructed using Algorithm 3 and a transitive blocking scheme. A path exists between two vertices $v_r$ and $v_s$ in the graph abstraction $G_\Gamma$ if the records $r$ and $s$ co-occur in at least one block.

**Proof:** We first show that if $r$ and $s$ occur in the same sub-block $B_o$, but do not fall within a common c-window (else $\{r, s\} \in \Gamma_c$), they are still connected by a sequence of pairs of the form $(\{r, u_1\}, \{u_1, u_2\}, \ldots, \{u_{t-1}, u_t\}, \{u_t, s\})$ where $1 \leq t$ and $\forall i, u_i \in B_o$. Note that two adjacent pairs in the sequence share exactly one common element. Suppose the positions of $r$ and $s$ in the sub-block are $p_r$ and $p_s$ respectively. Then, there are $|p_r - p_s| + 1$ records between them, including $r, s$. Let $m = \lceil (p_r - p_s) / c \rceil$, with $m \geq 2$ since $r, s$ don’t co-occur within a c-window. We can now construct the sequence above by letting $u_i$ be the record at position $p_r + i(c - 1)$ in the sub-block, for $1 \leq i < m$. Each pair (except possibly the last) in such a sequence will contribute its constituent records to the first and last positions in some c-window. Moreover, the $\lceil \cdot \rceil$ function guarantees that the last pair of records $\{u_{m-1}, s\}$ will fall within at least one common c-window. We note that in the graph abstraction of $\Gamma_c$, the existence of such a sequence guarantees the existence of a path between $v_r$ and $v_s$.

We can use the same logic but with $c$ replaced by $d$ to show that two records $r$ and $s$ are also connected by a sequence of the form above as long as they occur in the same slice, that is $|p_r - p_s| < c$ where $p_r$ and $p_s$ are the positions of $r$ and $s$ in their respective sub-blocks. The proof above shows that the two sub-blocks are not required to be within distance $d$ of each other.

For the general case, assume that $r$ and $s$ are neither in the same slice, nor the same sub-block. Without loss of generality, assume $p_r < p_s$. Consider any record $t$ in the same slice as $r$ and the same sub-block as $s$. Then, by the observations above, there is a sequence (and thus, a path in the graph abstraction) between $r$ and $t$ and also between $s$ and $t$; hence, we conclude there is a path between $v_r$ and $v_s$ in $G_\Gamma$ given $r$ and $s$ co-occur in a block.

**Lemma 3.** Consider the candidate set of pairs $\Gamma_c$ generated by Algorithm 3 using a transitive blocking scheme. Augment the set by a new pair $\{r, s\}$ corresponding to the edge $\{v_r, v_s\}$ generated by an incremental transitive closure algorithm on $G_\Gamma$, so that $\Gamma_c^+ = \Gamma_c \cup \{r, s\}$. Then, $\Gamma_c^+ \subseteq \Gamma_o$.

**Proof:** Suppose the transitive closure algorithm detects a path between two vertices $v_r$, $v_s$, but not an edge; it thus adds edge $\{v_r, v_s\}$ in that iteration. We note that $\Gamma_c \subseteq \Gamma_o$ to begin with since Algorithm 3 only pairs records if they fall within the same block. By transitivity of blocking schemes, the inclusion of pairs $\{r, s\}$ and $\{s, t\}$ in $\Gamma_c$ imply $r$ and $t$ co-occur in some block $B$. By induction, if a subset of $\Gamma_c$ can be organized as a sequence $\{(r, u_1), \ldots, (u_i, u_{i+1}), \ldots, (u_p, s), (s, t)\}$ where two adjacent pairs share exactly one common record and $p \geq 1$, $\{r, t\}$ co-occur in some block $B$. This implies that the edge $\{v_r, v_s\}$ exists in the graph abstraction of $\Gamma_o$. Thus, the graph abstraction of $\Gamma_c^+ = \Gamma_c \cup \{r, s\}$ is a subgraph of $\Gamma_o$, which in turn implies that $\Gamma_c^+ \subseteq \Gamma_o$.

Observing that the number of new pairs required to meet the $\Xi_{\text{max}}$ specification is $(\Xi_{\text{max}} - \Xi)\Omega$, with $\Omega$ the full set of pairs per Equation 5, the running time of the transitive closure algorithm will be $O((\Xi_{\text{max}} - \Xi)\Omega f(\Gamma_o))$, where $f(\Gamma_o)$ is the (amortized) time for finding a new pair to add. Thus, transitivity of blocking schemes is a valued property in that it provides a powerful way of monotonically improving the quality of the candidate set by investing a desired amount of computation.

If $\Gamma_o$ is desired, a transitive closure algorithm should be allowed to terminate, per Theorem 7.

**Theorem 7.** Given a transitive blocking scheme, the transitive closure of $G_\Gamma$, denoted $T(G_\Gamma)$, is exactly the graph abstraction of $\Gamma_o$.

**Proof:** Recall that the graph abstractions of $\Gamma_o$ and the

---

16In the graph abstraction, this is equivalent to adding an edge between two vertices iff a path exists between them.

17In which case, the result would be equivalent to that of running Algorithm 1 using the provided blocking scheme.
full set of blocks $B$ are the same, per Definitions 7 and 8. By Lemma 2, a path always exists in $G_T$ between two vertices $v_r$ and $v_s$ if $r$ and $s$ co-occur in some block. Then, the transitive closure will add an edge between $v_r$ and $v_s$ if one does not exist. Lemma 3 shows that, given a transitive blocking scheme, the incremented set will still be a subset of $\Gamma$. We conclude that, per Lemma 2, every pair of records that co-occur will end up getting connected by an edge in $T(G_T)$ and that, by Lemma 3, any such incrementation is valid. Thus, $T(G_T) = G_H$ and exactly equals the graph abstraction of $\Gamma$.

V. DISCUSSION AND OPEN PROBLEMS

As noted earlier, the community on blocking has not formally addressed the problem of N-Way heterogeneous blocking. Thus, the field as a whole is open to a wide range of open questions. Some of these are described below.

A. Learning blocking schemes

Learning blocking schemes for heterogeneous N-Way blocking remains a completely open area that merits further investigation. One could develop supervised and unsupervised algorithms by extending past work in the 2-Way homogeneous setting [17]–[19], or in a modified and more convenient framework. A key cross-fertilization research opportunity is to investigate use of schema mappings to guide heterogeneous learning.

B. Extensions of other blocking methods

Extensions to other blocking methods are also needed, for the N-Way case. Traditional blocking and sorted neighborhood are among the most popular, but many variants and other promising methods have also been proposed, including q-gram indexing, suffix array based indexing, clustering methods and string-map based methods [3].

C. Efficient heuristics for c-optimality

Heuristics that yield appropriate solutions for the c-optimal problem are also needed, together with evaluations and proofs on their approximability properties.

D. Empirical investigations of N-Way heterogeneous blocking

On the empirical front, an investigation of N-Way blocking techniques on real-world Big Data is a promising avenue and urgently warranted, given the velocity with which Big Data continues to explode.

VI. CONCLUSION

In this paper, a model for N-Way heterogeneous blocking was introduced. Some key properties of blocking schemes were proved. Two popular blocking methods, traditional blocking and sorted neighborhood, were extended. A component of sorted neighborhood, c-optimal ordering, was shown to be NP-hard. The N-Way heterogeneous sorted neighborhood thus proposed aims to efficiently generalize existing sorted neighborhood by using a novel dual windowing scheme. The algorithm is shown to present linear guarantees, while still generating a diverse candidate set that can be further augmented qualitatively, if the blocking scheme is transitive.

ACKNOWLEDGEMENTS

The authors thank Dr. Vijaya Ramachandran\(^{18}\) for showing a reduction of 2-optimal ordering from TSP. They also acknowledge Matt Johnson\(^{19}\) for helping refine the proof of Theorem 5.

REFERENCES


\(^{18}\)vlr@cs.utexas.edu
\(^{19}\)mattjohnson@utexas.edu
APPENDIX

We give the proof for Theorem 5.

Proof of Theorem 5

Proof: Theorem 4 showed that the 2-optimal ordering was NP-hard. Hence, it suffices to reduce from 2-optimal to a set of \(|q_s|\) objects to generic c-optimal on a set of \(|q_N|\) objects, since \(c < |q_s|\) implies \(c < |q_N|\).

For any such constant \(2 < c < |q_s|\), construct a new set \(q_N\) of \((c - 1)|q_s|\) objects. Intuitively, we are decomposing each object \(O_i\) into a cluster of \((c - 1)\) new objects \(\{O_{ij} | j \in Z_{c-1}\}\) (see Figure 2). For short, we henceforth denote this cluster as cluster \(i\). We determine the pairwise scores between each of these new objects using the following new similarity function \(\text{Sim}_{n}\):

1. \(\text{Sim}_{n}(O_i^k, O_i^l) = (c\Sigma_{m,o \in q_s} \text{Sim}(O_m, O_o)\) for all \(i\)
   and with \(1 \leq k, l < c\).
2. For two clusters \(i, j, i \neq j\), \(\text{Sim}_{n}(O_i^k, O_j^l) = \text{Sim}(O_i, O_j)\) for all \(1 \leq k < c\).
3. \(\text{Sim}_{n} = 0\) for all other object pairs.

Effectively, Rule 1 uniformly assigns a very strong (constant) similarity to the object pairs that lie within the same cluster. Rule 2 shows that a (possibly) non-zero score can only exist between the \(k\)th objects of any two clusters and is equal to the original score between the objects represented by the two clusters in the 2-optimal problem instance. Note that the score is independent of \(k\). Rule 3 assigns every other object pair in the scaled problem a score of zero. Note that the discrete computation of these scores is in polynomial time since \(c\) is a constant and computing pairwise comparisons for \((c - 1)|q_s|\) objects is quadratic in \(c - 1\) and \(|q_s|\). A subtle point to note is that every pair of objects within the same cluster has the same score and is independent of even the cluster itself, per Rule 1.

We perform c-optimal ordering on this transformed problem. Before showing a reduction of the solution to this c-optimal problem to that of the original 2-optimal ordering, we prove some properties of any c-optimal solution obtained by solving this problem:

Property 1. Consider the set \(L\) containing ordered lists of the form \(L = [O_{i_1}^{j_1}, O_{i_2}^{j_2}, \ldots, O_{i_{c-1}}^{j_{c-1}}, O_{i_1}^{j_1}, O_{i_2}^{j_2}, \ldots, O_{i_{c-1}}^{j_{c-1}}, \ldots, O_{i_{q_s}}^{j_{q_s}}, O_{i_{q_s}}^{j_{q_s}}, \ldots, O_{i_{q_s}}^{j_{q_s}}, O_{i_{q_s}}^{j_{q_s}}]\) with the set \(\{i_k | k \in Z_{|q_s|}\}\) having a bijection to the set \(Z_{|q_s|}\), and the set \(\{j_k | k \in Z_{c-1}\}\) having a bijection to the set \(Z_{c-1}\). \(\exists L \in L\) that is c-optimal.

Proof: The property essentially guarantees that at least one c-optimal solution is contained in the set of orderings such that clusters do not interleave in any given ordering in the set (see Figure 3). To see why this must be true, consider Rule 1. The summation in Rule 1 (not including the scaling factor \(c\)) is the same as the maximum possible score that can be achieved (invoking Line 3 of Algorithm 2\(^{20}\)) on the original set \(q_s\).

Furthermore, recall that \(|q_s|\) corresponds to the number of clusters in \(|q_N|\) and that each cluster contains \(c - 1\) elements. By Rule 2, only the \(k\)th elements between two clusters can have a non-zero score, non-aligned elements between any two clusters will have zero scores. This means that, given any two clusters \(i_1\) and \(i_2\), a maximum of \(c - 1\) pairwise comparisons between objects in \(i_1\) and objects in \(i_2\) can be non-zero. Moreover, by Rule 2, each of those \(c - 1\) scores are equal to each other and to \(\text{Sim}(O_{i_1}, O_{i_2})\) where \(O_{i_1}\) is the object in \(q_s\) corresponding to cluster \(i_1\), and similarly for \(i_2\). Generalizing to the full set of clusters indexed by \(i_1, i_2, \ldots, i_{|q_N|}\), if we consider all pairs such that both objects in the pair come from two separate clusters, the maximum achievable score is \((c - 1)\Sigma_{m,o \in q_s} \text{Sim}(O_m, O_o)\). Unless the term inside the summation equals zero\(^{21}\), this is strictly less than the pairwise score given in Rule 1. Thus, even a single pairwise score for two objects belonging to the same cluster trumps the sum of all pairwise scores for objects from two different clusters.

We note further that the window size can be at most \(c\) and hence to compute all (that is, \(c(c - 1)/2\)) pairwise scores within a cluster (denote the sum of this the intra-cluster score: formally, for each cluster, the intra-cluster score would then be the quantity in Rule 1 scaled additionally by \(c(c - 1)/2\), all the \(c - 1\) elements must fall within at least one common c-window. The argument for this is as follows. Suppose the position of the first occurrence of an object of cluster \(i_1\) is \(s\), and of the last occurrence (the \(c - 1\)th object) \(l\). If the objects of the cluster do not fall within a \(c\) window, then \(l - s \geq c\). Clearly, the first and last object will not pair with each other. There are two possible schemes we can use to arrange all the clusters so that we are assured of including all \(|q_s|\) intra-cluster scores in our score computation. Note that an optimal score would have to include all these \(|q_s|\) non-zero scores. By the very first argument, it’s maximum payoff for not doing this will be strictly less than if it did.

The first scheme corresponds to (any) element of \(L\). Namely, the clusters are not interleaved. For clarity, we use the terminology that a c-window has \(c\) slots, each of which contains an element (see Figure 3). Then, the windows that begin at

\(^{20}\)Recall this is the sum of scores between all pairs in the list

\(^{21}\)If it were, the proof is vacuously true, since \(\text{Sim}_{c} = 0\) for all inputs
positions (We start the count from 1) 1, c, ..., \(|q_k| - 1)(c - 1)\) will include all \(c - 1\) elements from the clusters \(i_1, i_2, ..., i_{|q_k|}\) respectively. Note that \(|q_k| - 1)(c - 1)\) is the last position a c-window can occupy in any list of size \(|q_N| = |q_k|(c - 1)\) (see Figure 4) without sliding past the list. Observe that the window at position 1, by this scheme, would also include (in its last slot) the first element of cluster \(i_2\). This is true for every c-window in the above positions except for the last one, which, in its first slot, contains the last element of the second-last cluster.

Exploiting the fact that each cluster has only \(c - 1\) elements, but each window has \(c\) slots, leads to the second scheme. By this scheme, we can interleave the clusters in a limited way. Specifically, we can take an ordering in \(L\), containing only non-interleaved clusters, and for a pair \((i_k, i_{k+1})\) of adjacent clusters \(i_k\) and \(i_{k+1}\) \((k \in \mathbb{Z}|q_k| - 1)\), we can exchange the last element of cluster \(i_k\) with the first element of cluster \(i_{k+1}\). For brevity, call this operation a swapping. An additional limitation then is that each cluster can participate in at most one swapping. For example, if we swap \((i_k, i_{k+1})\), then we cannot also swap \((i_{k+1}, i_{k+2})\) for some valid \(k\). If we do, then the first element of cluster \(i_{k+1}\) will not share a c-window with the \(c - 1^{th}\) element (see Figure 5).

Again, we see that if we obey these limitations, then the c-windows in list positions 1, 2, ..., \(|q_k| - 1)(c - 1)\) will again include all the elements from clusters \(i_1, i_2, ..., i_{|q_k|}\) respectively. Designate as \(L_r\) the set of orderings corresponding to this scheme and not the first scheme. We can construct \(L_r\) by starting with the empty set, taking each ordering in \(L\), performing every arbitrary combination of (at least one) legal swap (note the definition of swapping above; we are not swapping the clusters themselves) operations on its adjacent cluster pairs and add the resulting ordering to \(L_r\). It can be shown that an upper bound on the number of such orderings that can be generated, per a given ordering in \(L\), is \(2^{|q_k| - 1} - 1\). This is the exact bound if a cluster can participate in more than one swap, which it cannot by the above limitation. Since we also require that at least one swap operation per ordering be performed, at least two clusters will be interleaved in every ordering in \(L_r\). This implies that \(L_r \cap L = \emptyset\). Furthermore, the arguments above also imply that \(L_r \cup L\) contains all orderings such that the final score computed by the c-windowing scheme includes all the intra-cluster scores in the sum.

We further note that a swap operation cannot increase the total score computed using c-windowing, compared to the same ordering but without the swap. To see why, consider again a swap operation on \((i_k, i_{k+1})\). For simplified analysis, assume the first or last cluster is not involved \((|q_k| - 1 > k > 1)\). Then, the swapped ordering will exclude exactly two (potentially) non-zero pairs that will be included in the non-swapped ordering. These are the pairs between the last elements of clusters \(i_k\) and \(i_{k-1}\) and between the first elements of clusters \(i_{k+1}\) and \(i_{k+2}\). Potentially, by Rule 2, both these scores can be non-zero (see Figure 6).

On the other hand, the swapped ordering will not contain any new element pairs that can have non-zero scores. Specifically, the score between the last element of a cluster and the first element of another cluster can only be zero, since only Rule 3 can apply. Moreover, since a cluster can participate in at most one swap operation, the above effect cannot be mitigated by considering multiple swaps instead of just one. As a final note, if we do include the first and last cluster in the analysis, the swapped ordering will potentially exclude one pair (instead of two) that has non-zero score.

Finally, we note that there must be an optimal ordering such that objects within the cluster are aligned, that is, if a slot at position \(p\) contains the \(k^{th}\) element of a cluster \(i_k\), then the slot at position \(p + c - 1\) must contain the \(k^{th}\) element of the adjacent cluster \(i_{k+1}\). By induction, this means that once we decide the ordering of objects within the first cluster, the ordering of objects in the next \(|q_k| - 1\) clusters are also decided. To see why, consider Figure 7.

If the first objects of two clusters are not aligned, as in Figure 7, then at least one pair \((O_1, O_2)\) are potentially not
getting compared, and could have non-zero similarity, per Rule 2. We could inductively apply the argument (after aligning the first element of each cluster) to the second element, all the way to the $c-1^{th}$ element. We also note that it does not matter what the specific alignment is, since Rule 2 is independent of $k$. The only requirement is the alignment itself. Thus, if the clusters in an optimal ordering are not all aligned, we can align (all of them) according to a default alignment, and the score, per Rule 2, will remain the same.

Because we constructed $L$, by performing (minimum one) swap operations on $L$ we are assured that if $L_c$ contains a c-optimal ordering, then the ordering’s ‘non-swapped’ version (from which it was constructed) will also be c-optimal. This proves the property that $L$ has to contain a c-optimal ordering.

Given a c-optimal ordering in $L$, Property 2 shows that it correctly reduces to a solution of 2-optimal ordering in the original problem. For an intuition into the reduction, see Figure 8.

**Property 2.** A c-optimal ordering of the form $[O_1^{n_1}, O_2^{n_2} \ldots O_{j-1}^{n_{j-1}}, O_{j+1}^{n_{j+1}}, O_{j+2}^{n_{j+2}} \ldots O_k^{n_k}]$ reduces to a 2-optimal ordering of the form $[O_{i_1}, O_{i_2} \ldots O_{i_{qs}}]$ in the original problem.

**Proof:** Suppose not. That is, suppose the 2-ordering $L = [O_{i_1}, O_{i_2} \ldots O_{i_{qs}}]$, not optimal, given $[O_1^{n_1}, O_2^{n_2} \ldots O_{j-1}^{n_{j-1}}, O_{j+1}^{n_{j+1}}, O_{j+2}^{n_{j+2}} \ldots O_k^{n_k}]$ is c-optimal in the transformed problem. Instead, let the 2-optimal ordering be $L' = [O_{i'_1}, O_{i'_2} \ldots O_{i'_{qs}}]$. The set $\{i'_k| k \in Z|q_k|\}$ has a bijection to the set $Z|q_k|$ just like the set $\{i_k| k \in Z|q_k|\}$; however, $\exists k$ s.t. $i'_k \neq i_k$. By definition of optimality,

$$\sum_{k \in Z|q_k|} Sim(O_{i_k}, O_{i_k+1}) > \sum_{k \in Z|q_k|} Sim(O_{i'_k}, O_{i'_k+1})$$

(13)

Consider a solution $[O_1^{m_1}, O_2^{m_2} \ldots O_{j-1}^{m_{j-1}}, O_{j+1}^{m_{j+1}}, O_{j+2}^{m_{j+2}} \ldots O_k^{m_k}]$ to the transformed problem. To compute the score of this solution according to Algorithm 2, we observe (per Property 1) that for all solutions where clusters are not interleaved, the intra-cluster score of each cluster will be a common component of the score.

The differences between orderings will arise due to the scores computed between objects in different clusters. We note that for object $O_k$ (the $j^{th}$ object in cluster $i_k$, $k \in Z|q_k|$, $l \in Z|c-1|$), the pairs containing that object and the objects in cluster $k + 1$ onwards will be in the set $\{(O_{i_k}^m, O_{l+1}^m)|m \in Z\}$. For all values of $m$ but $m = l$, Rule 3 in the construction of similarity function $Sim_c$ dictates that $Sim_c(O_{i_k}^m, O_{i_{k+1}}^m) = 0$. But $Sim_c(O_{i_k}^m, O_{i_{k+1}}^m) = Sim(O_{i_k}, O_{i_{k+1}})$ Summing over $l$ which can only range from 1 to $c-1$ and considering Rule 2 in the construction of $Sim_c$, we get the c-score for the list $L'$ as $I_c + \sum_{k \in Z|q_k|} cSim(O_{i'_k}, O_{i'_{k+1}})$, where $I_c$ is sum of the aforementioned intra-cluster scores, for all clusters, and is common to all lists that don’t interleave clusters. Furthermore, because $c$ is a positive constant, this implies (by Equation 13) that $score(L') > score(L)$, by multiplying both sides of Equation 13 with $c$ and adding $I_c$. But we were given that $L$ is c-optimal, leading to a contradiction. Hence, it must be the case that the solution $[O_{i_1}, O_{i_2} \ldots O_{i_{qs}}]$ in the original problem is 2-optimal.

We note that the reduction also happens in polynomial time since we only need to scan the c-optimal list once. There are three technical possibilities we must account for. The first is if the oracle returns an optimal solution that is in $L_r$ instead of $L$. For such a case, we can reverse-swap to an element of $L$ as shown in Property 1, and the resulting ordering will still be optimal. Specifically, if we find that a swap operation has taken place between two adjacent clusters, we reverse swap so that those two clusters are not inter-leaved any more. Since a cluster can participate in at most one swap, and there are $|q_k|$ clusters, the reverse mapping also happens polynomially. A second technicality is if the elements within the clusters are not aligned. Again, as shown in Property 1, we can align all the clusters with respect to any ordering and the resulting list will still be optimal. This transformation also requires a single scan of the list and is polynomial.

A third technicality is if the oracle returns a list that consists of interleaved clusters. As shown in Property 1, if this happens, we are guaranteed $Sim_c = 0$ for all pairs. But then, every ordering is optimal (both in the transformed and original problems). We pick any random element from $L$ as an answer. Note that after accounting for these three technicalities, we get an element from $L$, and the reduction in Property 2 applies. Thus, any instance of a 2-optimal ordering problem can be reduced to a c-optimal ordering of $(c-1)|q_k|$ objects. The transformation happens in polynomial time. Once given the solution to the transformed problem, the reduction of the solutions also happens in polynomial time. Since 2-optimal ordering is NP-hard by Theorem 4, we conclude that in the general case, c-optimal ordering is also NP-hard.