

**Tesina Progetto**  
**Sistemi Informativi e Web Semantico**

**PPRL: PRIMAT Tool**

**Di**  
**Mario Cristiano**

# Indice

<i>1. Introduzione su Record Linkage</i>	<i>3</i>
<i>2. Cos'è il PPRL</i>	<i>3</i>
<i>3. Tool PRIMAT</i>	<i>4</i>
<i>3.1 Introduzione</i>	<i>4</i>
<i>3.2 Tecniche di PPRL</i>	<i>5</i>
<i>3.3 Descrizione di PRIMAT</i>	<i>5</i>
<i>4. Esempi di funzionamento con PRIMAT</i>	<i>8</i>
<i>4.1 Codice di esempio PRIMAT</i>	<i>8</i>
<i>4.1.1 DataOwners Function</i>	<i>8</i>
<i>4.1.2 Linkage Unit Function</i>	<i>9</i>
<i>4.2 Risultati raggiunti</i>	<i>11</i>
<i>4.3 Applicazione basata su PRIMAT</i>	<i>14</i>
<i>5. References</i>	<i>15</i>

# 1. Introduzione su Record Linkage

Il record linkage mira a collegare i record provenienti da differenti fonti di dati che però si riferiscono alla stessa entità del mondo reale. Di solito, quando possibile, nel mapping tra i record, si usano gli identificatori diretti (globali), come, ad esempio, un numero di identificazione personale.

In generale, però, mancano gli identificatori globali per mappare direttamente i record, quindi il collegamento può essere ottenuto solo confrontando i quasi-identificatori (quasi-identifiers) disponibili, come il nome, l'indirizzo o la data di nascita. Questi identificatori potrebbero non essere unici, potrebbero cambiare nel tempo, e potrebbero avere delle differenze di implementazione nelle singole basi di dati. Quindi, se il record linkage si basasse solo su un matching esatto tra i quasi-identificatori potrebbe esserci un numero elevato di falsi positivi (missing matches).

Tuttavia, in molti casi, i data owners sono disposti / autorizzati a fornire i loro dati per tale integrazione solo se vi è una sufficiente protezione delle informazioni sensibili per garantire la privacy delle persone (come i pazienti di un ospedale o i clienti di una struttura).

Ad esempio, nella ricerca medica, i dati di diverse fonti (ad esempio, i dati provenienti da diversi ospedali) devono essere abbinati per studiare eventuali correlazioni tra alcune malattie senza rivelare l'identità dei singoli pazienti.

In molti casi, invece, i dati devono essere protetti per legge, al fine di garantire l'anonimato dei vari soggetti.

# 2. Cos'è il PPRL

Il Privacy Preserving Record Linkage (PPRL) affronta questo problema fornendo tecniche per abbinare i vari record, preservando la loro privacy e consentendo la combinazione di dati provenienti da fonti diverse per migliorare l'analisi dei dati e la ricerca.

A tal fine, il collegamento dei record relativi alle persone si basa su valori codificati dei quasi-identificatori e i dati necessari per l'analisi (ad esempio, i dati sanitari) sono separati da questi quasi-identificatori. Perciò, i dati rilevanti possono essere forniti a un ricercatore senza i dati identificativi.

Questi approcci PPRL possono essere applicati in molti settori, come la sorveglianza della salute pubblica, gli studi demografici e le analisi di marketing.

Il PPRL deve, quindi, affrontare diverse sfide:

- è necessario garantire un elevato grado di privacy attraverso un'adeguata codifica dei dati sensibili, e l'uso di un'unità di collegamento fidata tra le parti (trusted linkage unit).
- il PPRL deve raggiungere un'elevata qualità di collegamento, evitando false o mancate corrispondenze (FP, FN).

- è necessaria un'elevata efficienza, con tempi di linkage rapidi, e una buona scalabilità con grandi volumi di dati. Un problema principale per le prestazioni è la complessità quadratica del record linkage, quando ogni record della prima fonte viene confrontato con ogni record della seconda fonte. Per una maggiore efficienza, il numero di confronti può essere ridotto adottando approcci di blocco o di filtraggio. Inoltre, il matching può essere eseguito in parallelo su più nodi di elaborazione.

### 3. Tool PRIMAT

#### 3.1 Introduzione



PRIMAT (Private Matching Toolbox) è uno strumento open source flessibile e scalabile, scritto in Java, sviluppato dal “Database Group” dell’università di Leipzig in Germania, che permette l’applicazione di flussi di PPRL, nonché la valutazione comparativa di diversi metodi PPRL. PRIMAT si occupa di identificare record in diversi database che si riferiscono alla stessa persona, garantendo scalabilità fino a milioni di record, e di proteggere le informazioni personali,

garantendo la privacy e riducendo al minimo il rischio di divulgazione di informazioni sensibili. Questo strumento trova applicazioni in diversi settori, tra cui la sanità, la sicurezza degli stati, le analisi di marketing, garantendo alta affidabilità e qualità nel processo.

L’integrazione di dati personali è necessaria in molte applicazioni. Tuttavia, i requisiti legali più severi in materia di protezione dei dati richiedono che sempre più spesso che la data integration rispetti la privacy e che non riveli l’identità delle persone per le quali i dati vengono combinati e analizzati. Questi requisiti sono soddisfatti dalle tecniche di Privacy-Preserving Record Linkage (PPRL) che codificano gli attributi identificativi, ad esempio il nome e la data di nascita, ed eseguono il linkage dei record codificati in un ambiente separato e fidato. Negli ultimi anni sono stati proposti un enorme numero di metodi per il PPRL. Ma, nonostante il gran numero di proposte, il loro uso pratico nelle applicazioni reali è ancora limitato a causa dell’assenza di strumenti convenienti e dell’elevata complessità nel selezionare un approccio PPRL adeguato. Sebbene esistano alcune implementazioni e prototipi di PPRL, essi si concentrano spesso più su aspetti di ricerca e non forniscono sufficienti funzionalità per l’utilizzo pratico.

È necessaria, quindi, la presenza di uno strumento facile da usare, intuitivo, affidabile e open source, per facilitare l’adozione del PPRL nelle applicazioni reali. PRIMAT fornisce differenti componenti, protocolli, modalità di linkage, e framework per la valutazione, in termini di affidabilità ed efficienza, dei diversi metodi di PPRL.

Il PPRL presenta, in particolare, tre sfide fondamentali che devono essere affrontate:

- è necessario garantire un elevato grado di privacy fornendo tecniche di codifica all'avanguardia che riducano il rischio di violazione dei dati.
- è necessario raggiungere un'elevata qualità di linkage, ossia ridurre al minimo il numero di corrispondenze false (FP - False Positive) e mancanti.
- il PPRL deve essere scalabile per grandi volumi di dati, fino a milioni di record. Per questo scopo tornano utili metodi di filtraggio e metodi di elaborazione a blocchi, distribuita e parallela.

Tipicamente il PPRL viene eseguito in batch mode, dove il linkage dei record viene eseguito in gruppi di record. Tuttavia, può essere necessario effettuare il linkage in maniera continua (incrementale), senza per forza ripetere il linkage dei record già processati.

Nel PPRL è essenziale supportare anche approcci multy-party, con due o più data owners. In questi casi non deve esserci solo un linkage dei record, ma essi devono anche essere raggruppati in cluster, in maniera tale che tutti i record in un cluster corrispondano tra loro.

### 3.2 Tecniche di PPRL

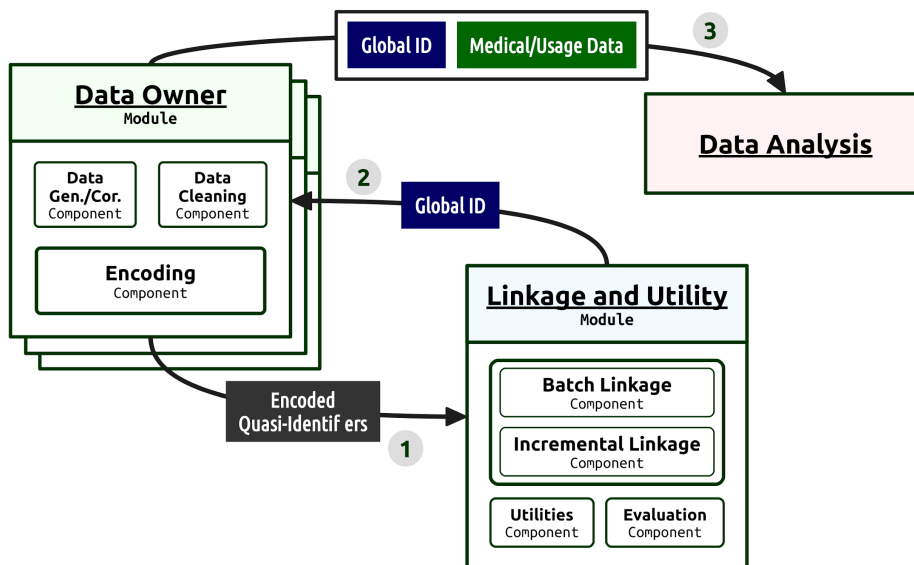
Gli approcci più recenti riguardo il PPRL si basano per lo più su tecniche di codifica basate su *Bloom-filter*, che sfruttano trusted linkage unit centralizzate che eseguono il linkage dei record codificati. Questa tecnica è basata su un vettore di bit, inizialmente inizializzati a zero, e su un insieme di funzioni di hash. Ogni funzione hash viene applicata a ogni attributo del record e restituisce un indice all'interno del vettore di bit. Infine, i bit in queste posizioni dell'indice vengono impostati a uno.

Recentemente sono stati sviluppati anche altri tool, tutti con delle inefficienze nell'implementazione del flusso di PPRL: *MergeToolBox*, che però fornisce un supporto limitato al linkage dei record; *Mainzel-liste*, che però non supporta l'elaborazione a blocchi e le attuali tecniche di codifica che tutelano la privacy, peccando, quindi, in scalabilità e protezione dei dati; *SOEMPI*, tool che, però, non supporta metodi di evaluation e non è ottimizzato per il linkage incrementale; *LSHDB*, che non supporta a pieno il clustering dei record.

### 3.3 Descrizione di PRIMAT

Si distinguono due ruoli nel processo di PPRL: Database Owners (DOs) e Trusted third parties.

I Database Owners gestiscono i record con i dati sensibili di cui poi dovrà essere fatto il linkage, mentre le terze parti fidate effettuano il linkage vero e proprio. Nel caso di PRIMAT, esiste una Trusted Linkage Unit (LU) che effettua il linkage dei record codificati forniti dai DOs.



Ogni processo di PPRL consiste in 3 step:

1. Ogni Database Owner prepara i record per il linkage e codifica gli attributi identificativi di ogni entità, i quali poi vengono mandati alla Linkage Unit.
2. La Linkage Unit esegue il linkage dei record e restituisce i global ID ai Database Owners. Pertanto, ogni coppia di record che ha lo stesso ID globale viene considerata come una corrispondenza.
3. I dati ottenuti possono essere combinati per ulteriori Data Analysis.

In PRIMAT esistono, quindi, due moduli:

- DO Module: usato per fare pre-processing e codifica dei record del database per preparare i record al linkage.
- Linkage and Utility Module: fornisce varie tecniche, metodi e procedure per effettuare il linkage dei record.

Nella tabella sottostante sono riportate le funzionalità di PRIMAT già implementate e quelle pianificate per futuri aggiornamenti.

Comp.	Function / Feature	Status
$D_1$	Data generation	Implemented
	Data corruption	Planned
$D_2$	Split/merge/remove attributes	Implemented
	Replace/remove unwanted values and stopwords	Implemented
	OCR transformation	Implemented
	Intra-source record linkage	Planned
$D_3$	Bloom filter encoding and hardening techniques	Implemented
	Client-side standard blocking	Implemented
$D_1 - D_3$	Graphic user interface	Planned
$L_1$	Pre-processing templates	Planned
	Private schema matching	Planned
$L_2$	Standard & LSH-based blocking; metric space filtering	Implemented
	Threshold-based classification	Implemented
	Post-processing	Implemented
	Multi-threaded processing	Implemented
	Distributed processing	Planned
	Multi-party support	Planned
$L_3$	Incremental linkage	Planned
$L_4$	Quality & scalability evaluation	Implemented
	Privacy-preserving match result visualization	Planned

*Data Generation and Corruption ( $D_1$ ):* componente che consente di generare dataset sintetici realistici, possibilmente basati su dati reali, che possono essere utilizzati per analizzare i flussi di PPRL.

*Data Cleaning ( $D_2$ ):* componente che consente di effettuare pre-processing sui record, e, in particolare, operazioni di cleaning e standardizzazione dei dati.

*Data Encoding ( $D_3$ ):* gli attributi identificativi di ogni record vengono codificati (utilizzando tecniche di codifica basate su Bloom-Filter).

*Utilities ( $L_1$ ):* componente che fornisce metodi per preparare i dati per il linkage.

*Batch Linkage ( $L_2$ ):* è il componente principale della Linkage Unit. Esso implementa diverse tecniche di linkage. In particolare, per ridurre la complessità del linkage vengono forniti approcci di blocking standard e basati su LSH, e approcci di filtraggio. Per la classificazione vengono usate misure di similarità come la Jaccard o la Dice similarity. Sono inclusi, inoltre, metodi di post-processing, che vanno a migliorare la qualità del linkage, selezionando i migliori match candidates se ci sono più record che eccedono una determinata soglia di similarità.

*Incremental Linkage ( $L_3$ ):* fornisce supporto per memorizzare e recuperare risultati di match precedenti, rendendo possibile il linkage incrementale.

*Evaluation Tool ( $L_4$ ):* fornisce metodi per valutare il flusso di PPRL, fornendo, ad esempio, metriche di qualità come precision, recall e F-measure.

## 4. Esempi di funzionamento con PRIMAT

### 4.1 Codice di esempio PRIMAT

#### 4.1.1 DataOwners Function

```
public static void DataOwners(String inputPathName, String partyName) throws IOException {

    // Dataset Reading

    final NamedRecordSchemaConfiguration rsc = new NamedRecordSchemaConfiguration.Builder()
        .add(0, NonQidAttributeType.ID)
        .add(1, NonQidAttributeType.PARTY)
        .add(2, QidAttributeType.STRING, "fname")
        .add(3, QidAttributeType.STRING, "gname")
        .add(4, QidAttributeType.STRING, "address")
        .add(5, QidAttributeType.STRING, "dob")
        .build();

    final String inputPath = inputPathName + ".csv";
    final String outputPath = inputPathName + "_encoded.csv";

    final DatasetReader reader = new DatasetReader(inputPath, true, rsc);
    final List < Record > records = reader.read();

    // Data Cleaning

    final PartySupplier partySupp = new PartySupplier();
    partySupp.preprocess(records);

    final SplitDefinition splitDef = new SplitDefinition();
    splitDef.setSplitter("address", new BlankSplitter(2));
    splitDef.setSplitter("dob", new DotSplitter(3));

    final FieldSplitter fs = new FieldSplitter(splitDef);
    fs.preprocess(records);

    final NormalizerChain normChain = new NormalizerChain(
        List.of(new UmlautNormalizer(), new TrimNormalizer(), new LowerCaseNormalizer(), new
        AccentRemover(),
        new SpecialCharacterRemover(), new SubstringNormalizer(0, 12)));

    final NormalizerChain numberNorm = new NormalizerChain(new
    LetterLowerCaseToNumberNormalizer(),
    new LetterUpperCaseToNumberNormalizer());

    final NormalizeDefinition normDef = new NormalizeDefinition();
    normDef.setNormalizer(0, normChain);
    normDef.setNormalizer(1, normChain);
    normDef.setNormalizer(2, numberNorm);
    normDef.setNormalizer(3, normChain);
    normDef.setNormalizer(4, numberNorm);
    normDef.setNormalizer(5, numberNorm);
    normDef.setNormalizer(6, numberNorm);

    final FieldNormalizer fn = new FieldNormalizer(normDef);
    fn.preprocess(records);

    // Encoding
```



```

final FeatureExtractor featEx = new BigramExtractor(true);
final int k = 10;

final BloomFilterExtractorDefinition exDef = new BloomFilterExtractorDefinition();
exDef.setColumns(0, 1, 2, 3, 4, 5, 6);
exDef.setExtractors(featEx);
exDef.setNumberOfHashFunctions(k);

final HashingMethod hashing = new RandomHashing(1024, RandomFactory.SECURE_RANDOM);

final BloomFilterDefinition def1 = new BloomFilterDefinition();
def1.setName("BS");
def1.setBfLength(1024);
def1.setHashingMethod(hashing);
def1.setFeatureExtractors(List.of(exDef));
def1.setHardener(new NoHardener());

final Encoder encoder = new BloomFilterEncoder(List.of(def1));

final List < Record > encodedRecords = encoder.encode(records);

// Final dataset writing

final CSVWriter csvWriter = new CSVWriter(outputPath);
csvWriter.writeRecords(encodedRecords, encoder.getSchema());
}

```

## 4.1.2 Linkage Unit Function

```

public static void LinkageUnit(String inputPathName) throws IOException {

    // Dataset Reading

    // Schema Configuration
    final NamedRecordSchemaConfiguration rsc = new NamedRecordSchemaConfiguration.Builder()
        .add(0, NonQidAttributeType.ID)
        .add(1, NonQidAttributeType.GLOBAL_ID)
        .add(2, NonQidAttributeType.PARTY)
        .add(3, QidAttributeType.BITSET, "BS")
        .build();

    System.out.println("#Named Record Schema Configuration rsc "
        + rsc.getNonQidAttributeMap()
        + rsc.getQidAttributeNameMap());

    // Parameters
    final double threshold = 0.8;
    final int matches = 20;

    final String namePartyA = "A";
    final String namePartyB = "B";

    // Read the dataset file
    final DatasetReader reader = new DatasetReader(inputPathName, true, rsc);
    final List<Record> dataset = reader.read();
    System.out.println("#dataset " + dataset);

    // List the records for the 2 dataset A and B
    final List<Record> datasetA = dataset.stream()
        .filter(r ->
            r.getPartyAttribute().getValue().getName().equals(namePartyA)).collect(Collectors.toList());
}

```

```

    System.out.println("#dataset A " + datasetA);

    final List<Record> datasetB = dataset.stream()
        .filter(r ->
            r.getPartyAttribute().getValue().getName().equals(namePartyB)).collect(Collectors.toList());
    System.out.println("#dataset B " + datasetB);

    // Create input for the 2 dataset
    final Map<Party, Collection<Record>> input = new HashMap<>();
    input.put(new Party(namePartyA), datasetA);
    input.put(new Party(namePartyB), datasetB);
    System.out.println("#input " + input);

    System.out.println("#Records source " + namePartyA + ": " + datasetA.size());
    System.out.println("#Records source " + namePartyB + ": " + datasetB.size());

    final ComparisonStrategy compStrat = ComparisonStrategy.SOURCE_CONSISTENT;

    final LshKeyGenerator keyGen = new RandomHammingLshKeyGenerator(16, 30, 1024, 42L);
    final Blocker blocker = new LshBlocker(keyGen);

    final RecordSimilarityCalculator simCalc = new BaseRecordSimilarityCalculator(
        List.of(new
    BitSetAttributeSimilarityCalculator(List.of(BinarySimilarity.JACCARD_SIMILARITY))));

    final SimilarityVectorFlattener flattener = new BaseSimilarityVectorFlattener(
        List.of(DoubleListAggregator.FIRST));
    final FlatSimilarityVectorAggregator aggregator = new BaseSimilarityVectorAggregator(
        DoubleListAggregator.FIRST);
    final SimilarityVectorAggregator agg = new SimilarityVectorAggregator(flattener,
        aggregator);

    final Classifier classifier = new ThresholdClassifier(threshold, agg);

    final MatchStrategyFactory<Record> matchFactory = new
    SimilarityGraphMatchStrategyFactory<>();
    final NonMatchStrategyFactory<Record> nonMatchFactory = new
    IgnoreNonMatchesStrategyFactory<>();
    final LinkageResultPartitionFactory<Record> linkResFac = new
    LinkageResultPartitionFactory<>(matchFactory, nonMatchFactory);
    final SimilarityClassification simClass = new BatchSimilarityClassification(compStrat,
        simCalc, classifier,
        RedundancyCheckStrategy.MATCH_TWICE, linkResFac);
    final ThresholdClassificationRefinement threshRef = new
    StandardThresholdClassificationRefinement();

    final PostprocessingStrategy<Record> postprocessor = new PostprocessingStrategy<>();
    postprocessor.setPostprocessor(LinkageConstraint.ONE_TO_ONE, new
    MaxBothPostprocessor<Record>());
    postprocessor.setPostprocessor(LinkageConstraint.MANY_TO_ONE, new
    MaxRightPostprocessor<Record>());
    postprocessor.setPostprocessor(LinkageConstraint.ONE_TO_MANY, new
    NoPostprocessor<Record>());
    postprocessor.setPostprocessor(LinkageConstraint.MANY_TO_MANY, new
    NoPostprocessor<Record>());

    final Matcher<Record> matcher = new BatchMatcher(blocker, simClass, threshRef,
        postprocessor);

    final LinkageResult<Record> linkRes = matcher.match(input);

    final TrueMatchChecker trueMatchChecker = new IdEqualityTrueMatchChecker();
    final QualityEvaluator<Record> evaluator = new QualityEvaluator<>(trueMatchChecker);

    final PartyPair partyPairAB = new PartyPair(new Party(namePartyA), new Party(namePartyB));

```

```

final LinkageResultPartition<Record> part = linkRes.getPartition(partyPairAB);
evaluator.addMatches(part.getMatchStrategy().getMatches());

final long truePos = evaluator.getTruePositives();
final long falsePos = evaluator.getFalsePositives();

final double recall = QualityMetrics.getRecall(truePos, matches);
final double precision = QualityMetrics.getPrecision(truePos, truePos + falsePos);
final double fmeasure = QualityMetrics.getFMeasure(recall, precision);

System.out.println("Recall: " + recall);
System.out.println("Precision: " + precision);
System.out.println("F-Measuer: " + fmeasure);

}

```

## 4.2 Risultati raggiunti

### Dataset 1

#### Input

Dataset A: 20 record

Dataset B: 20 record

Esempio:

<u>id</u> ,	<u>fname</u> ,	<u>gname</u> ,	<u>address</u> ,	<u>dob</u>
0,	Dimitrios,	Huebner,	81379 Muenchen,	17.12.1974
1,	Ruth,	Schroeder,	42899 Remscheid,	11.06.1953

#### Data Cleaning

- Accent Remover
- Special Character Remover
- Lower Case Normalizer
- UmlautNormalizer
- TrimNormalizer

#### Test

- Funzione di Similarità: Jaccard Similarity  
Threshold: 0.7

True Positive (TP): 20  
 False Positive (FP): 0  
 Precision: 1  
 Recall: 1  
 F-measure: 1  
 Match completati correttamente: 20 / 20

- Funzione di Similarità: Jaccard Similarity  
Threshold: 0.8

True Positive (TP): 18  
False Positive (FP): 0  
Precision: 1  
Recall: 0.9  
F-measure: 0.9474  
Match completati correttamente: 18 / 20

- Funzione di Similarità: Dice Similarity  
Threshold: 0.8

True Positive (TP): 20  
False Positive (FP): 0  
Precision: 1  
Recall: 1  
F-measure: 1  
Match completati correttamente: 20 / 20

## Dataset 2

### Input

Dataset A: 5000 record

Dataset B: 5000 record

Esempio:

rec\_id, given\_name, surname, street\_number, address\_1, address\_2, suburb, postcode, state,  
date\_of\_birth, soc\_sec\_id

rec-1070-org, michaela, neumann, 8, stanley street, miami, winston hills, 4223, nsw, 19151111,  
5304218

rec-1016-org, courtney, painter, 12, pinkerton circuit, bega flats, richlands, 4560, vic, 19161214,  
4066625

### Data Cleaning

- Accent Remover
- Special Character Remover
- Lower Case Normalizer
- Umlaut Normalizer
- Trim Normalizer

## Test

- Funzione di Similarità: Jaccard Similarity  
Threshold: 0.8

True Positive (TP): 4950  
False Positive (FP): 50  
Precision: 0.9990  
Recall: 1  
F-measure: 0.9994  
Match completati correttamente: 4950 / 5000

Dataset A	Dataset B	Similarity
rec-3768-org, UWSlpZfjaKalMAY...	rec-3768-org, UWSlpZfjaKalMAY...	0.9358024691358025
rec-4887-org, kRQRo6/nbbOaNj...	rec-4887-org, kRVxo6/nbbOaNj...	0.9321663019693655
rec-1688-org, gT4E4IfiSSKYPHH...	rec-1688-org, gT4E4IfiSSaYPHH...	0.9395465994962217
rec-4065-org, gPAAoUiEZJAYAC...	rec-4065-org, gPAAociEZJQYAC...	0.8498402555910544
rec-85-org, 1XQiFce4baDZKCZiA...	rec-85-org, 1XQiFcf4baDZaCZiAl...	0.9232613908872902
rec-3119-org, kKwAJCuiaADIJG...	rec-3119-org, krwAJCumaADIZ...	0.898936170212766
rec-4238-org, wRQDsYtgygjKLG...	rec-4238-org, wRUDsYtgygjKLG...	0.9420654911838791

- Funzione di Similarità: Dice Similarity  
Threshold: 0.8

True Positive (TP): 4950  
False Positive (FP): 50  
Precision: 0.9990  
Recall: 1  
F-measure: 0.9994  
Match completati correttamente: 4950 / 5000

## 4.3 Applicazione basata su PRIMAT

Esiste anche una versione grafica di PRIMAT (PRIMAT Application:  
<https://github.com/gen-too/primat>).

### Data Owner APP

The screenshot displays the PRIMAT DO Module application. The main window features a menu bar with 'File' and 'Help'. On the left is a sidebar with icons and labels for 'Selection', 'Data Cleaning', 'Encoding', and 'Indexing'. The central area contains a table with 15 rows and 5 columns: 'id', 'fname', 'gname', 'address', and 'dob'. The first row (id 0) is highlighted in orange. Below the table is a control bar with buttons: 'Add', 'Add All', 'Remove', 'Remove All', 'Remove Duplicates', and a 'Confirm' button. At the bottom, a 'Data' section shows a smaller table with the first 5 rows of the main table, and a '#Rows: 20' indicator.

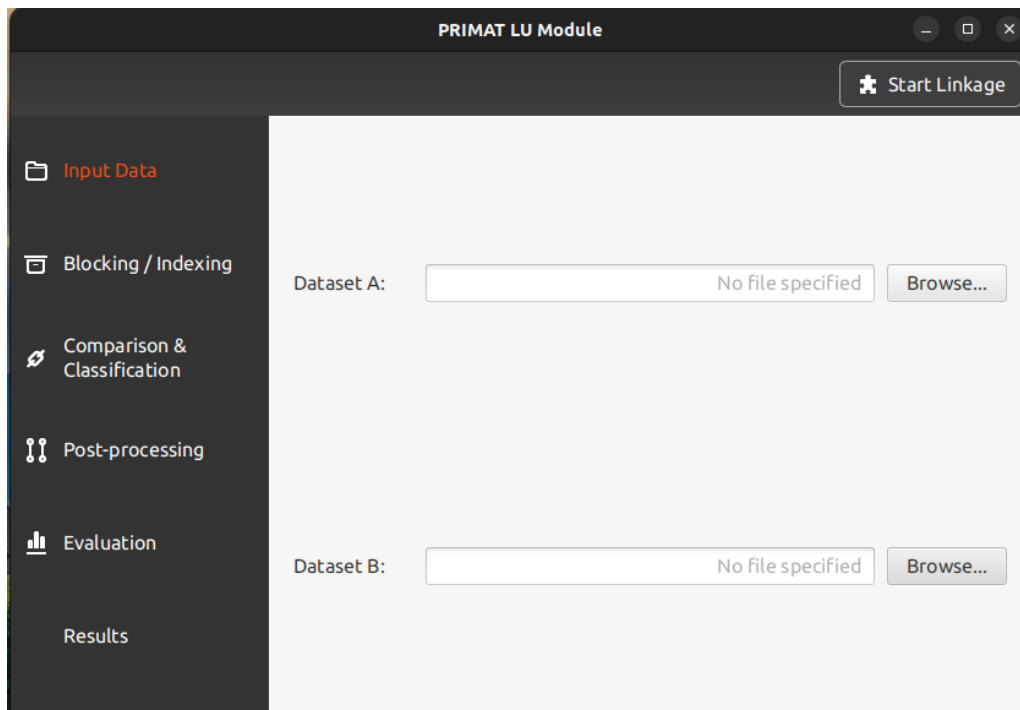
id	fname	gname	address	dob
0	Dimitrios	Huebner	81379 Muenchen	17.12.1974
1	Annette	Gerth	41189 Moenchengladbach	16.03.1971
2	Ruth	Schroeder	42899 Remscheid	11.06.1953
3	Alexander	Bergner	45894 Gelsenkirchen	26.11.1962
4	Bernhard	Schumacher	52538 Selfkant	10.03.1968
5	Carmen	Raschke	82547 Eurasburg	12.10.1949
6	Ute	Neumann	66539 Neunkirchen/Saar	12.02.1954
7	Heinz	Becker	42781 Haan	03.01.1980
8	Elmar	Jahn	44625 Herne	08.11.1965
9	Ilka	Sieg	12559 Berlin	16.03.1986
10	Ferdinand	Wahl	12587 Berlin	31.10.1951
11	Christian	Baumann	22844 Norderstedt	08.09.2003
12	Dagmar	Winter	49681 Garrel	26.01.1959
13	Hartmut	Eichler	59199 Boenen	18.01.1966
14	Melanie	Kupfer	46147 Oberhausen	22.11.1940

Buttons: Add, Add All, Remove, Remove All, Remove Duplicates, Confirm

#Rows: 20

id	fname	gname	address	dob
0	Dimitrios	Huebner	81379 Muenchen	17.12.1974
1	Annette	Gerth	41189 Moenchengladbach	16.03.1971
2	Ruth	Schroeder	42899 Remscheid	11.06.1953
3	Alexander	Bergner	45894 Gelsenkirchen	26.11.1962
4	Bernhard	Schumacher	52538 Selfkant	10.03.1968
5	Carmen	Raschke	82547 Eurasburg	12.10.1949

## Linkage Unit APP



## 5. References

- Scientific paper: Privacy Preserving Record Linkage (Rainer Schnell)
- Scientific paper: PRIMAT: A Toolbox for Fast Privacy-preserving Matching (Martin Franke, Ziad Sehili, Erhard Rahm)
- PRIMAT: <https://git.informatik.uni-leipzig.de/dbs/pprl/primat>
- PRIMAT Application: <https://github.com/gen-too/primat>
- <https://www.boozallen.com/insights/ai/privacy-preserving-record-linkage.html>
- <https://www.sciencedirect.com/science/article/abs/pii/S0306437921001526>
- <https://github.com/data61/anonlink-entity-service>
  
- My Code + Presentation: <https://github.com/mariocris/SIWS.git>