# SuperPart: Supervised graph partitioning for record linkage

Russell Reas
Amazon
Seattle, WA
reas@amazon.com

Stephen Ash
Amazon
Seattle, WA
ashstep@amazon.com

Robert A. Barton
Amazon
New York, NY
rab@amazon.com

Andrew Borthwick
Amazon
Seattle, WA
andborth@amazon.com

## ABSTRACT

Identifying sets of items that are equivalent to one another is a problem common to many fields. Systems addressing this generally have at their core a function $s(d_i, d_j)$ for computing the similarity between pairs of records $d_i, d_j$. The output of $s()$ can be interpreted as a weighted graph where edges indicate the likelihood of two records matching. Partitioning this graph into equivalence classes is non-trivial due to the presence of inconsistencies and imperfections in $s()$. Numerous algorithmic approaches to the problem have been proposed, but (1) it is unclear which approach should be used on a given dataset; (2) the algorithms do not generally output a confidence in their decisions; and (3) require error-prone tuning to a particular notion of ground truth. We present SuperPart, a scalable, supervised learning approach to graph partitioning. We demonstrate that SuperPart yields competitive results on the problem of detecting equivalent records without manual selection of algorithms or an exhaustive search over hyperparameters. Also, we show the quality of SuperPart's confidence measures by reporting Area Under the Precision-Recall Curve metrics that exceed a baseline measure by 11%. Finally, to bolster additional research in this domain, we release three new datasets derived from real-world Amazon product data along with ground-truth partitionings.

**PVLDB Reference Format:**
Russell Reas, Stephen Ash, Robert A. Barton, Andrew Borthwick. SuperPart: Supervised graph partitioning for record linkage. *PVLDB*, 11 (5): xxxx-yyyy, 2018.
DOI: https://doi.org/TBD

## 1. INTRODUCTION

Computing a *partitioning* of a weighted graph is an important problem whenever one is trying to determine equivalence relationships in a database [7]. One example application is the detection of duplicate records corresponding to the same real-world item in a database of structured records [17] [14] [19]. Another example is identifying functionally related proteins whose sequence similarity indicates that they share a common evolutionary history and have a similar function [8]. In these cases, the output partitioning groups all records detected to refer to the same real-world entity into the same equivalence class. All records are assigned to an equivalence class, and no records are assigned to multiple equivalence classes; it is a hard partitioning of the input dataset. This is a different problem from K-means clustering [21] in that we do not know $K$ in advance, and in general, the number of equivalence classes $|E|$ is unknown and is usually expected to grow proportional to $|D|$.

By definition, the equivalence relationships are reflexive, symmetric, and transitive. Thus, we have a definition of equivalency such that $a \equiv a$, $a \equiv b \implies b \equiv a$, and $a \equiv b \wedge b \equiv c \implies a \equiv c$. However, in real-world record linkage these equivalence relationships must be inferred from noisy measures over noisy data, making this a hard problem.

In this work, we frame the problem as: given $d$ records as vertices with $e$ computed edges, *e.g.* by $s(d_i, d_j)$, with possible errors in $e$, find a partitioning of approximately equivalent records such that the confidence of the resulting partitions, as determined by a model $M$, is maximized. Given the variety of real-world definitions of equivalence (such as the one highlighted in Figure 2), we seek to build $M$ from training data such that the resulting partition confidence scores reflect human labeling decisions. Additionally, in many industrial settings the confidence scores are needed to prioritize costly human workflows (*e.g.* auditing). The need for confidence scores produced by $M$ to be calibrated and accurate is included in our problem scope.

### 1.1 Background

For the database deduplication/record linkage application that we are studying, the most common approach to detecting equivalence relationships in $D$ is to divide the problem into four stages [20] [12]:

1. **Normalization**: records are run through simple and sophisticated standardization procedures to remove unimportant typographical variance. This significantly improves results [3].


*Proceedings of the VLDB Endowment,* Vol. 11, No. 5

DOI: https://doi.org/TBD

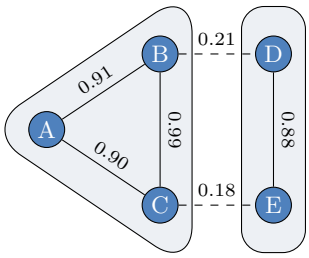


Figure 1: Graph partitioning problem easily solved by connected components at a threshold of 0.5

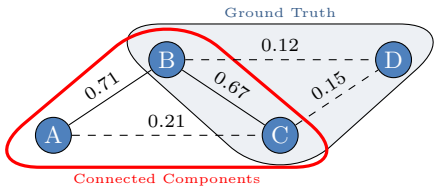


Figure 2: Graph partitioning problem where no threshold for connected components results in the correct clustering

2. **Blocking**: blocking algorithms [22] [23] [25] [27] are used to cheaply identify possibly matching pairs of records in a way that avoids doing $O(n^2)$ comparisons over all possible pairs of records

3. **Pairwise Matching**: a pairwise matching algorithm [6] [5] computes the score that each pair of records proposed by the blocking stage is actually a match. We denote the matching similarity function as $s(x, y) \mapsto \mathbb{R}$, where higher values of $s()$ indicate a greater likelihood that $x$ and $y$ belong to the same equivalence class.

4. **Graph Partitioning**: the output of the previous stage can be interpreted as a (usually large) sparse weighted graph in which the vertices correspond to input records and the weighted edges are the model-calculated similarity score, $s()$. In this phase, we seek a partitioning of the graph into non-overlapping clusters of records such that each partition corresponds to a real-world entity.

This work focuses on the graph partitioning stage. The graph partitioning problem has received attention in the field of record linkage/data deduplication [14] [24] and is also a problem of importance in the different domain of computational biology [29] [30] [1].

It may, at first glance, be unclear why the Graph Partitioning stage is even worthy of study. If we have a perfect similarity function $s()$ which always assigns a high score to record pairs where the ground truth indicates they are equivalent and a low score to pairs that are not equivalent, then we can trivially partition the data by a standard connected component algorithm [18]. In this solution, we compute the connected components over the edges whose score is above some threshold separating the *high* and *low* scoring pairs as illustrated in Figure 1. Even an imperfect but consistently erroneous similarity function $s()$ will likely result in a graph for which a connected component algorithm is sufficient. However, if the ground truth indicates that a particular node should be connected and $s()$ consistently says that it should be separated, there is no reasonable basis for a connected component algorithm to overrule $s()$ (see Figure 2).

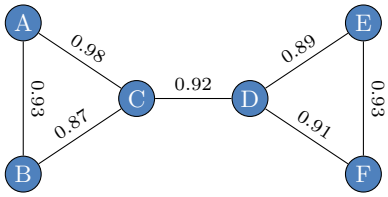


Figure 3: Single, possibly erroneous edge agglomerates two tight clusters

The necessity of a sophisticated graph partitioner stems from the fact that $s()$ may output inconsistent signals, but by examining the larger context of the graph instead of only the pairwise score, we are able to learn better ways to partition the graph into equivalence clusters. For example, Figure 3 illustrates a situation in which nodes $[a, b, c]$ are tightly linked, nodes $[d, e, f]$ are tightly linked, and the two sets of three are linked by a single high probability edge between $c$ and $d$. It seems intuitive that the high value of $s(c, d) = 0.92$ may be erroneous in this case. The naïve transitive closure algorithm will put them all into the same partition.

## 1.2 Related Work

A broad range of algorithms have been proposed to attack graph partitioning. Prior surveys of the field [14] [29] have found inconsistent results as to which algorithms are the best performers on individual problems and have thus recommended that practitioners test multiple algorithms to identify the one that performs the best on their domain. For real-world, social networks others have evaluated which descriptive metrics best uncover the ground-truth network communities [31].

Besides the lack of a single best algorithm, a second issue facing prior work is the lack of a confidence measure for a produced clustering. In our experience, a measure of confidence is essential for many practical applications. It is common for customers to wish to distinguish between detected entities in which the system is highly confident (which can be automatically merged), entities about which the system is uncertain (which should be human reviewed), and entities that the system deems to be unlikely (which can be rejected). Most prior work yields a partitioning but does not give a measure of confidence on each partition. Pairwise match algorithms typically output a continuous measure of similarity, $s(x, y)$, that can often be regarded as a confidence [9] [6]. However, due to the inconsistent signals from $s$ mentioned previously, naïve attempts to combine pairwise scores to quantify overall cluster quality (*e.g.* mean of all cluster edge weights) are not obviously suitable or optimal.

A significant obstacle to the adoption of a supervised learning approach to graph partitioning is that the instances of a graph partitioning problem may be too large to be feasible either from the point of view of human-labeling ground-truth examples or for executing the underlying algorithms that would yield the required signals. For instance, one of the algorithms considered in this work is Markov Clustering

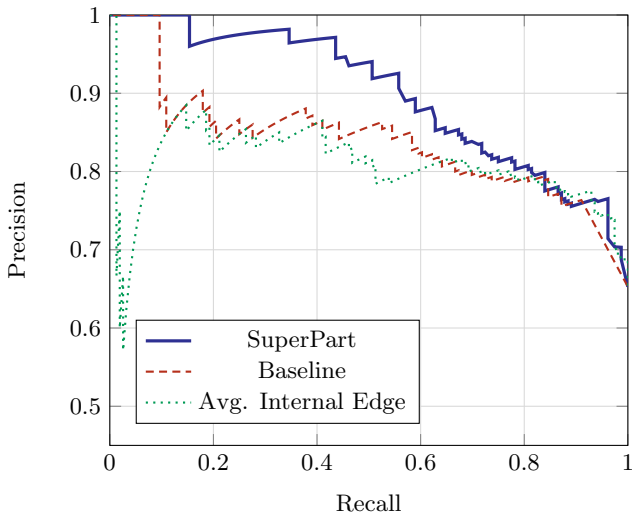


Figure 4: Precision-Recall curve on binary classification task for the IC-Random dataset. SuperPart achieves AURPC 0.898, baseline achieves AUPRC 0.839, and the naïve internal edge weight method achieves AURPC 0.807.

(MC) [26], which is $O(n^3)$ for naïve implementations. Additionally, Sequential Rippling (SR) in the Ricochet family of clustering algorithms [28] is also $O(n^3)$ in the worst case. Hence, whenever one wishes to employ one of these super-linear algorithms, there is a need to partition the input graph into subgraphs such that each partition is a tractable size.

An additional obstacle impeding progress in the field is a lack of published benchmark datasets. Prior surveys in both structured record linkage [14] and computational biology [29] have described toolkits for addressing the multi-step problem of record linkage [14] or identifying expression patterns of proteins [29]. However, there are few published real-world datasets for record linkage problems in the form of weighted or unweighted graphs that would form the direct input to a graph partitioning algorithm; and even fewer have associated ground truth.

## 1.3 Contributions

To address the issues discussed in the previous section, we make the following contributions:

- **Apply supervised learning to graph partitioning:** Given a modest corpus of ground truth partitions over a weighted graph, we train a machine-learned model to recognize characteristics of a good partitioning. Our model uses a feature vector constructed from a diverse ensemble of unsupervised partitioning algorithms as well as descriptive measures of a graph partitioning. Uniquely, our approach produces a learned confidence score for a given partitioning and a set of alternative partitionings with potentially higher scores.

- **Describe how to scale the approach:** We use a Problem Instance Detector that coarsely partitions a graph into smaller, reasonably-sized sub-graphs, which we then further partition using our ensemble of competing partitioning algorithms.

- **Demonstrate effectiveness without tuning:** We report results using internal Amazon real-world record linkage datasets, which show that the SuperPart method produces competitive results (best or within 0.5%) over manually selected and tuned clustering algorithms. Our results require no exhaustive search for optimal hyper parameters or algorithms. On a task comparing SuperPart's confidence quality to pre-existing baseline methods, SuperPart improved the AUPRC by 11% as highlighted Figure 4.

## 1.4 Reproducibility

To facilitate future research, we provide a repository of weighted graphs with associated ground-truth partitionings. This repository facilitates apples-to-apples comparisons in future research by eliminating the complex and domain-specific task of computing the pairwise similarity graph from the underlying data. We contribute three labeled graphs anonymized from internal Amazon product-related datasets along with human audited cluster-wise ground truth labels.

# 2. SYSTEM OVERVIEW

We begin by defining the inputs and outputs of the proposed system more precisely. Given a database $D$ which we wish to partition into distinct equivalence classes, we assume that there exists a pairwise similarity function $s(d_i, d_j) \mapsto \mathbb{R}$ for any $d_i, d_j \in D$. We assume that higher scores from $s$ indicate a greater likelihood that the pair of items are *similar*, meaning that the pair is likely to be in the same equivalence class. We also assume the availability of ground truth data that is available for training and testing. In this work, we consider a ground truth of cluster-wise labels in which each record is assigned a `cluster_id` such that records with the same `cluster_id` represent the same real-world entity. More specifically, our ground truth datasets contain a `record_id` and a `cluster_id` as a cross reference. In each of our real-world datasets, the cluster-wise labels are collected from human auditors, which are presented with a graphical user interface containing a list of records. The auditor then drags records into distinct groups to indicate their equivalence classes.

Figure 5 illustrates the schematic of SuperPart at run time. The overall goal of the process is to compute a partition of database $D$ and a set of confidences on the partitioning. We start by doing a coarse partitioning to split up the graph into *problem instances* that are of a manageable size so as to be computationally tractable. In our problem domain of record linkage, we use a connected components algorithm that only considers edge weights above a threshold $t = 0.5$. Given the sparsity of typical record linkage similarity graphs, this has worked well in practice. Now that we have a number of problem instances, we compute any missing edges in order to complete the graph. Edges might be missing in a record linkage context, because blockers only emit a small subset of possible record pairs, and when the connected components are calculated, the records close over edges never proposed by the blocker. Next, we evaluate multiple clustering algorithms on each problem instance. Each algorithm produces (possibly non-distinct) *proposals* for how to partition the problem instance. Then a trained proposal scoring model evaluates each proposal and assigns a confidence score indicating the degree to which the model believes

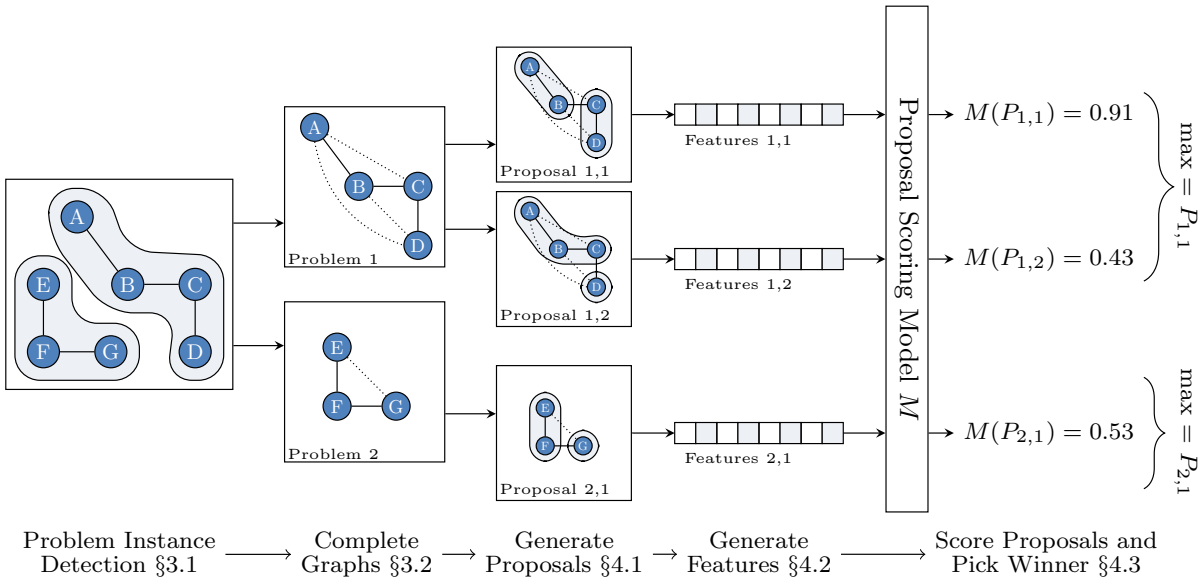


Figure 5: System schematic for SuperPart

the proposal is the correct partitioning. The proposal with the maximum score is picked as the final clustering.

At train time, we use the same process to generate all of the proposals and their feature vectors. Then we train a binary classifier, a Random Forest in our experiments, to classify each proposal based on whether it matches the ground truth clustering or not. The following sections describe each step of this process in detail.

# 3. CREATING PROBLEM INSTANCES

## 3.1 Problem Instance Detection

The SuperPart algorithm uses supervised learning to produce a confidence measure on a partitioning of a graph. Central to the practicality of this approach is a need to coarsely partition the input sparse weighted graph $G = (V, E)$ into a set of $m$ reasonably-sized graphs $G'_i = (V'_i, E'_i)$, where $V'_i$ in this context refers to the vertices in $G'_i$ and $E'_i$ is the set of all edges between vertices in $V'_i$.

We will refer to this first phase of SuperPart as the task of *Problem Instance Detection*, for which we require a Problem Instance Detector (PID) $\phi$ where the PID is defined as $\phi(G = (V, E)) \mapsto \{G'_0, \ldots, G'_{m-1}\}$, where $\forall_{V'_i \in V} |V'_i| \leq k$. In this context $k$ is some reasonable upper bound on the size of each problem instance. In our experience, a value of $k \leq 200$ is reasonable because that is the most number of records on a screen that a human auditor can effectively deal with at a single time.

In this work, we use a thresholded a transitive closure clustering algorithm [18] as a simple implementation of $\phi$. More precisely, we define $\hat{G} = (V, \hat{E})$ where $\hat{E} = \forall_{(x,y)\in E}\, s(x, y) \geq t$. Then, our simple $\phi$ emits $m$ problem instances $G'_i$ for each of the $m$ connected components in $\hat{G}$. In our experiments, we picked a threshold $t = 0.5$, which conveniently resulted in no problem instance $G'_i$ having more than 200 vertices. The choice of 0.5 is reasonable when edge weights are the probability that two nodes are equivalent. In many record linkage approaches $s()$ can be made to emit a probability or some other well-calibrated score for which one can easily pick a threshold. Although more sophisticated Problem Instance Detectors are easy to imagine, this rather naïve approach has worked surprisingly well on the domains we have examined.

## 3.2 Creating complete weighted graphs

As noted above in section 1, practical record linkage systems require a *blocking* step, because it is infeasible to compute the pairwise similarity $s(d_i, d_j)$ for all record pairs $d_i, d_j$ for a database $D$ when $|D|$ is large. The implication then is that $G$ will be sparse and hence the partitions $G'_i$ will likewise be sparse. Although the core graph partitioning algorithm described below in section 4 can, in principle, operate on a sparse graph, we choose to compute the complete graph $\hat{G}'_i = (V'_i, \hat{E}'_i)$ and $\hat{E}'_i = E'_i \cup \{s(x, y) \mid (x, y) \in V'_i \times V'_i \wedge (x, y) \notin E'_i\}$ (*i.e.* any missing edges from the problem instance). This is cheap to compute on our problem instances, because it is clear that $\sum_i |E'_i| \ll \binom{|V|}{2}$. We include the complete graph, because we want the proposal scoring model to have a complete picture of the problem instance. Having all of the weights may provide interesting signal in descriptive features (section 4.2.2), which the model can use to distinguish good clusterings from bad. Note that in subsequent sections, when we describe working on problem instances, $G'_i$, it is implied that we are working on the completed problem instance graph.

# 4. PARTITIONING THE INSTANCES

Given a problem instance $G'_i = (V'_i, E'_i)$, we now wish to find the best partitioning. We proceed in three stages which we will describe in turn.

## 4.1 Proposers

Since the number of unique partitions for a set of size $n$ is exponential in $n$, it is clearly infeasible to consider every possible partitioning. Instead, we rely on a collection of

*Proposers*, $\pi$, where each proposer $\pi_p \in \pi$ is a clustering algorithm that computes a candidate partitioning of the problem instance $G'_i$; that is $\pi_p(G'_i) \mapsto \{G''_0, \ldots, G''_{q-1}\}$ where in this case the proposer suggested $q$ final partitions for the problem instance. Running all $\pi_p \in \pi$ proposers over the problem instance, we get $n$ different unique candidate partitionings of $G'_i$.

With a large number of proposers, it is unlikely that they will all suggest a *different* clustering of the problem instance. Therefore over all proposers we define $\pi(G'_i) \mapsto \{P_{i,0}, \ldots, P_{i,n-1}\}$ where $P_{i,j}$ is a unique partitioning for problem instance $i$ (potentially offered by more than one proposer) and thus $|P_i| \leq |\pi|$. The idea is that $\pi$ consists of a set of parameterized graph partitioning algorithms that cover a broad set of diverse algorithms with various configuration parameters.

Proposers used in SuperPart are summarized briefly below:

1. Identity: we pass through each problem instance $G'_i$ without modification.

2. Transitive Closure: given a threshold $t$ we select only edges with weight greater than $t$ in a given problem instance. We then return all connected components in the modified sub-graph.

3. Markov Cluster Algorithm (MCL): [26] A fast and scalable unsupervised clustering algorithm based on simulation of (stochastic) flow. MCL uses a non-negative stochastic column matrix where an entry $w_{ij}$ corresponds to the probability of moving from node $i$ to node $j$ in a random walk on the graph (and vice-versa). There are two primary parameters for MCL: The inflation rate controls whether more probable random walks are favored and the expansion rate controls the length of random walks. For this paper, we use a naïve implementation of the algorithm on a dense matrix.

4. Center [16] and Merge-Center [15]: both Center and Merge-Center produce a graph partitioning after a single pass through a sorted edge list. We parameterize these algorithms with a threshold $t$ via the same method applied to Transitive Closure as described above. The Center algorithm merges nodes into a cluster with a *center* of sufficient similarity to any unclustered node. Sorting edges by edge weight in descending order and starting with the heaviest-weight edge, we select one associated vertex to be a *center* and assign the other vertex to its cluster. Subsequent edges have the same behavior if both vertices are currently unassigned. If only one is assigned and is also a *center* then the other is assigned to its cluster. If both are assigned then the edge is skipped and no reassignment happens. The Merge-Center algorithm follows the same approach, however, it also allows for the merging of two clusters if their centers are similar.

5. K-Core: [4] for a threshold $t$, the K-Core algorithm yields sub-graphs of a problem instance such that each node in the sub-graph connects to at least $k$ other nodes in the same sub-graph after removing edges less than or equal to the threshold $j$. Singletons are returned as such.

Since $\pi$ is quite diverse, we expect the partitions $P_i$ to be diverse and hope that the ground truth clustering $Y_i$ is present in the $n$ proposals. We initially included synthetic proposals in the data used for training whenever a particular ground truth clustering was not present in one of the proposals, but found that this hurt overall system performance. The experimental results described in section 6 were only trained on proposals generated from problem instances, and thus if no proposer suggested the correct clustering, the model would have only negative examples for that problem instance.

## 4.2 Features

For each candidate partition $P_{i,j}$, we want to model the probability that $P_{i,j} = Y_i$ (*i.e.* the ground truth clustering). At this point we use a standard supervised learning approach by treating the problem as a binary classification task, where at training time, we compute a model $M = \psi(F, L)$ where $F$ is a $|P| \times f$ feature matrix and $L$ is our length $|P|$ ground-truth labeling of each proposer. $L$ is constructed by $L(P_{i,j}) = (P_{i,j} == Y_i)$. Since $P_i$ are unique proposals, we get zero or one true labels for each problem instance $G'_i$ and all other proposals are labeled false.

Our feature vector of length $f$ for each $P_{i,j}$ consists of two broad categories of feature values: indicator functions for each proposer and descriptive weighted graph measures. We consider each in the following sections.

### 4.2.1 Proposers as Features

By construction, for every partitioning $P_{i,j}$, there exists at least one proposer $\pi_p$ such that $\pi_p(P_i) = P_{i,j}$. The first category of feature values is simply a boolean vector of length $|\pi|$ indicating whether the corresponding proposer $\pi_p$ predicted this partitioning. In this way, every proposer listed in section 4.1 becomes a feature in $F$. Different proposers may be better suited for a particular problem domain or dataset. By including the binary indicator highlighting which proposer suggested a given partitioning, we allow the model to learn about the overall expectation of how each proposer will perform on this problem domain.

### 4.2.2 Weighted graph metrics as features

The second category of features in $F$ are real or Boolean-valued metrics that describe a partitioning. These features are engineered in order to provide signal to the classification model about different characteristics of clusterings. Different problem domains have different definitions of what is a good clustering. Even within the same dataset, a ground truth might contain a heterogeneous set of labeled truth clusters. These descriptive features allow the model to associate detectable patterns and characteristics of candidate proposals with the ground truth's definition of a good clustering. There are multiple sub-categories of these metrics. First, consider a (problem instance, proposal) pair: $(G'_i = (V'_i, E'_i), P_{i,j})$, which for simplicity in this section we will simply refer to as $(G = (V, E), P)$. The first sub-category of descriptive metrics omits all edges in $E$ that cross partitions within $P$ and omits all edges with a weight below a threshold $t$. More precisely, we define $\text{Consistent}(v_i, v_j) = (P(v_i) == P(v_j))$ (*i.e.* both vertices are in the same partition). Then, we define a partitioned, thresholded graph, $G'$ as follows: $G' = G(V, \{(x, y) \in E \mid \text{Consistent}(x, y) \wedge s(x, y) \geq t\})$. We compute the following metrics on $G'$:

- Biconnected: given a set of thresholds to test, the feature value is the maximum threshold at which each partition is biconnected [18] (ignoring singleton partitions, which are partitions with only a single element). Only edges that have weights greater than the threshold being tested are considered when testing biconnectivity. We test thresholds $j$ in the interval $j \in [0.1, 0.9]$ incrementing by 0.1 and return the maximum $j$ as a single feature.
- Diameter: the diameter of a graph is the longest shortest-path distance between any two nodes in the graph [13]. In the context of a partitioned, thresholded proposal, each partition has a possibly different diameter. We emit one feature value for various diameters, $d$. For each $d$, we compute the maximum edge threshold $w$ such that every partition has diameter $\leq d$. In our experiments we use $d \in [1, 2, 3, 4]$ yielding 4 different features. Diameter characterizes the density of partitions and captures whether many low weight edges affect shortest paths.

The second sub-category of descriptive features also operates on a (problem instance, proposal) pair but unlike the first sub-category does not drop cross-partition edges. That is these metrics are computed on $G' = G(V, \{(x, y) \in E \mid s(x, y) \geq t\})$.

- Coverage[10]: the ratio of within-partition edges to the total number of edges in a problem instance. If all edges are within-partition edges, then the coverage of the problem instance is 1.0. We use a single, preselected threshold of 0.5 to filter edge weights for coverage calculations.
- Performance [10]: the ratio of intra-partition edges plus missing cross-partition edges to the total number of possible edges in the problem instance. In our experiments, we use a single preselected threshold of 0.5 to filter edge weights before calculating performance.

Additionally, we have descriptive features on the original, unthresholded (problem instance, proposal) pair $(G(V, E), P)$:

- Size (in $|V|$) of the problem instance
- Minimum weight edge internal to any partition
- Mean and variance in weight for edges internal to any partition
- Maximum weight edge crossing two partitions
- Mean and variance in edge weights crossing two partitions
- Size (in $|V|$) of the largest partition. This includes both the *absolute* size and the *relative* size, which is the size of the largest partition divided by $|V|$
- The total number of proposed partitions
- The total number of edges less than a selected threshold internal to any partition. This feature is calculated for a number of preselected threshold values in the interval $[0.1, 0.9]$ incrementing by 0.1.
- The total number of edges greater than a selected threshold crossing any two partitions. This feature is calculated for a number of preselected threshold values in the interval $[0.1, 0.9]$ incrementing by 0.1.

## 4.3 Supervised Learning

Given the feature matrix $F$ and labels $L$, we compute the model $M = \psi(F, L)$ where $\psi$ is a supervised learning algorithm for binary classification that produces some reasonably calibrated score in [0..1], such as a probability. In our experiments presented in section 6, we use the Random Forest model implemented in Apache Spark[1] with hyper parameters of 100 trees with a max depth of 2. At run time, we pick the proposal with the maximum score from the classification model: $\hat{P}_{i,j} = \text{argmax}_j\,(M(P_{i,j}))$. The score for the proposal is the measure of confidence on the clustering.

In an alternate scenario, where you are only interested in computing a confidence score on an existing clustering, you can go through the same process as illustrated in Figure 5, but instead of picking the maximum score, you choose the score corresponding to the proposal with the indicator that is suggested by the *Identity* proposer (see section 4.2.1). Since the Identity proposer is the *current* clustering, this is a measure of confidence of the existing graph. In this case, it is important to calculate the proposals from all other proposers, because the resulting $P_{i,j}$ that contains the Identity might also have indicators for other proposers. This is the case when multiple proposers already agree with the current identity partitioning and their presence in the feature vector may provide signal to the model that it is a high quality clustering.

# 5. DATASETS

As discussed in section 1, the Graph Partitioning phase constitutes a distinct phase of the record linkage process. However, progress in the field has been hampered by a lack of benchmark datasets that would facilitate direct comparison among competing approaches. Existing datasets suffer from one or more of the following defects:

- **Lack of a ground truth partitioning**. Ground-truth clearly facilitates using a dataset as a benchmark. Often it is expensive to label clusters and therefore labels are omitted or extremely limited in size. Alternatively, ground truth clusters are synthetically created and may not represent real-world problems [14].
- **Dataset requires complex pre-processing**. In the case of record linkage datasets, this often means that users of the dataset must first compute pairwise similarity measures from the raw attributes. This opens up the possibility that two results may differ due to the quality of the similarity measure $s(x, y)$ rather than the quality of the graph partitioning algorithm.
- **Ground-truth clusters are too small**. If too large a fraction of the ground-truth positive examples consist of equivalence classes of size 2, then Graph Partitioning is a trivial problem.
- **Graph is not weighted**. Although SuperPart works on both weighted and unweighted graphs, weighted graphs are more complex and more interesting.

To remedy this problem, we present the SuperPart graph partitioning repository[2]. Every dataset in this repository

[1]https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-regression

[2]https://github.com/rreas/amzn-supervised-clustering

| Dataset | Components | Ground-Truth Clusters | Records |
|---|---|---|---|
| General (Train) | 86 | 213 | 886 |
| General (Test) | 101 | 175 | 1604 |
| IC-Random (Train) | 254 | 437 | 3511 |
| IC-Random (Test) | 239 | 436 | 3014 |
| IC-Stratified (Train) | 257 | 490 | 2649 |
| IC-Stratified (Test) | 243 | 443 | 2424 |

Table 1: Size summary of 3 new, real-world datasets released as part of this paper.

contains a ground-truth partitioning, is presented in the form of a weighted graph, and presents data where the ground truth clustering has a substantial number of clusters consisting of three or more records. For simplicity and (in some cases) confidentiality reasons, these datasets are presented as abstract weighted graphs. Two files are provided for each data set:

1. The *edges* files contain the `component_id`, `record_id1`, `record_id2` and a pairwise similarity `score` for each pair.
2. The *labels* files contain the ground truth `record_id` and `partition_id` for each item in the dataset. That is, they contain a mapping of a record to a ground-truth cluster.

Note that all `record_id`'s are randomized and do not correspond to any concept used by Amazon internally or externally. `component_id` is an optional column that is similarly randomized and is included for convenience. We maintain the invariant that there is no sharing of `record_id`'s and there are no edges that cross two `component_id`'s. Conceptually a component can be thought of as a set of records that were collectively human-reviewed. The judge was tasked with assigning equivalence classes within that set.

All released datasets are derived from the products domain. We include the following datasets in the repository:

- **General**: this dataset includes randomly sampled products from the Amazon catalog that are likely, as determined by an internally developed model, to be part of an incomplete or incorrectly grouped family. Human labelers group these products into clusters following specific guidelines. The ground truth labels correspond to the results of this audit. In addition to uncovering the correct ground truth clustering we would like to use a confidence score to prioritize existing groups or sets of groups for review by auditors.
- **Inconsistent-Random Sample**: we use the same sampling process as with the general dataset, however, here we focus just on finding product families that should be split. The ground truth corresponds to a partitioning within each existing grouping (or the entire group when consistent). Anecdotally, we expect this to be an easier task. We collect and report results on whether we can uncover the ground truth clustering as well as whether we can successfully prioritize audits of groups that are known to be inconsistent.
- **Inconsistent-Stratified Sample**: again, we focus on the simpler task of identifying groups to split. However, we take a stratified sample across the entire product catalog.

For the General dataset described above we must identify a method for problem instance detection. Here, we have used an internal, pairwise model to generate scores in the interval $[0, 1]$ between many pairs in the products catalog. Using these scores and applying transitive closure at a threshold of 0.5 we find clusters that contain many existing product families and thus disagree with the current product grouping. Any conflicting groupings are then labeled with ground-truth clusterings. The Inconsistent datasets already have a natural problem instance detection method. For these we simply select individual product families and look for those that should be partitioned into at least two families. Table 1 summarizes the released datasets.

# 6. RESULTS

Using the datasets explained in section 5 we follow a general experimental setup:

1. **Create problem instances**: over the output of pairwise matching (step 3 from the introduction description), we apply a Problem Instance Detector (PID) to generate problem instances. In the case of the General dataset this is simply generating connected components of the graph by considering only edges with a weight $\geq 0.5$. For the Inconsistent datasets, a natural domain-specific partitioning already exists and there is no need for taking the transitive closure over edges.
2. **Calculate a weighted graph**: using pairwise models developed internally we complete the graphs by scoring all of the pairs in each problem instance (as described in section 3.2). The edge scores are the same as provided in the released datasets. In the case of the General dataset, we use the same model to generate edges for connected component detection and so this step is simply a filtering of available edges.
3. **Generate partitioning proposals**: we apply the Proposers listed in section 4.1 at selected thresholds to generate candidate partitionings for each problem instance. Edge weights are in the interval $[0, 1]$, and we select thresholds $0.5 \leq t \leq 0.9$ in 0.1 increments and then filter to edges with weight $> t$. For K-Core clustering we use these same thresholds, however, with an additional parameter of core order set to either 2 or 3. For MCL we use a single set of parameters: inflation rate of 1.4, expansion rate of 2 and run for at most 200 iterations.

We compare end-to-end performance of SuperPart against a hold-out set of the ground truth cluster labels using the BCubed clustering quality metric [2]. Without SuperPart,

one must find the right algorithm and the right hyperparameters. In order to benchmark against the best-case scenario, we present results comparing SuperPart to a number of popular unsupervised clustering algorithm in Table 2. In each case, we present the best results over all tested hyperparameter options. We list the best hyperparameters to illustrate that the best performing results for existing unsupervised algorithms occur at different threshold levels for different datasets.

For the Inconsistent datasets described above, we additionally measure Area Under the Precision-Recall Curve for a binary classification task of deciding whether an input component should be split. In practice, the splitting can be done by auditors, or we may consider making changes directly based on the output of SuperPart, if the confidence score produced for a new partitioning is high enough.

For this classification task, we also compare our method to a heuristic baseline. The baseline is calculated as follows: first, components are partitioned by transitive closure at threshold $t$. Next, we select all edges that cross between two partitions and calculate the mean edge weight $w$. Since any edge weighted greater than or equal to $t$ connects a component of the graph, we have $w < t$. Then, the baseline is equal to $(1 - 1/t * w)$. Intuitively, when $w$ is close to $t$ the baseline score is near 0 and when $w$ is close to 0 then the baseline approaches 1. When scoring a component that is not partitioned by transitive closure the baseline is set to 0. We use $t = 0.9$ for experiments presented here when calculating the baseline.

Results are presented comparing SuperPart against the baseline described above as well as against a third, naïve approach of just taking the average intra-partition edge weights. Average edge weight is a typical, initial approach to quantify the quality of a clustering. In Table 3 along with the precision-recall curve (Figure 4) for SuperPart, results are presented for the Inconsistent-Random dataset.

## 7. DISCUSSION

In this work we present SuperPart, a novel supervised approach to graph clustering. As noted above, this method provides a number of material improvements to the problem. First, SuperPart outputs clusters with an associated *confidence* value, reflecting the model's belief that the partitioning is correct. Since this model is trained on ground truth labels, this measure can capture complex, even idiosyncratic, patterns specific to the domain. Confidence values are particularly useful in industrial settings, where it is commonly the case that portions of your output are audited and corrected by humans. Given the large size of industrial datasets, it is infeasible to review every system decision. A highly calibrated quality measure, tailored to the domain, allows for optimization of work assignment to human auditors.

Second, SuperPart is extensible and uses any past or future clustering approach in a principled way that avoids tedious manual algorithm selection and tuning for your problem domain. [14] illustrates that different partitioning algorithms provide the best accuracy depending on the problem. To that end, SuperPart can incorporate additional new or existing clustering algorithms in a straightforward manner by including them in the proposal generation step (section 4.1).

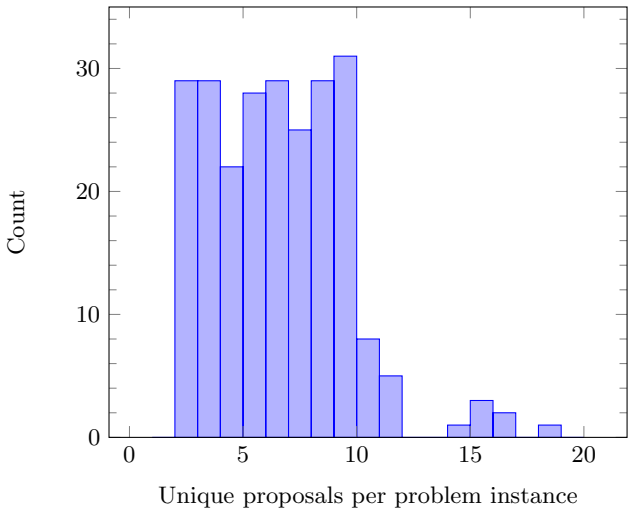


Figure 6: Histogram of unique proposals per problem instance in the Inconsistent-Stratified test dataset

In the context of record linkage, the two phase approach of creating problem instances (section 3) and partitioning the problem instances (section 4) fits naturally, because the input graph is a large, very sparse, similarity matrix and the threshold for creating problem instances is relatively easy to pick in such a way as to maximize recall. The pairwise model often produces calibrated scores, such as probabilities, which have natural operating points. In other graph clustering domains, it may be more difficult to create problem instances in such a way as to not propagate false negative error in the first step. We have not yet experimented with more sophisticated approaches to creating problem instances.

Most of the literature on graph clustering describes unsupervised methods to solve the problem. This is due to the use of graph clustering as a data exploration technique and a lack of ground truth clusterings available. In the context of industrial record linkage, it is feasible to collect ground truth cluster labels. As mentioned previously, the human auditor workflow is already a part of many record linkage scenarios where data quality is a critical concern. This is a natural place to collect human labeled clusterings that can be used to train and iteratively improve the supervised model. We find that the definition of a match and the desired characteristics of equivalence classes vary greatly across the various scenarios in which we use record linkage at Amazon. Additionally, even within a particular scenario, the ground truth clusters are not homogeneous. One benefit of SuperPart is its ability to pick different algorithms to cluster different parts of the similarity graph. Figure 6 illustrates the diversity of proposals in the Inconsistent-Stratified dataset. All problem instances have at least two different proposals and many have over 10 proposals.

As the results in Table 2 show, SuperPart successfully selects partitionings from underlying proposers and achieves BCubed F1 inline with or slighly above the best proposers' results. On the Inconsistent-Stratified dataset, SuperPart modestly improves the BCubed F1 against the most competitive alternatives but notably did not require any manual effort to determine which proposers would yield the best result for any given problem instance.

| Dataset | SuperPart | Merge-Center | Center | Transitive Closure | Markov | K-Core |
|---|---|---|---|---|---|---|
| IC-Random | 0.925 | 0.927 (t=0.8) | 0.577 (t=0.6) | **0.928 (t=0.8)** | 0.879 | 0.921 (t=0.7) |
| IC-Stratified | **0.910** | 0.908 (t=0.5) | 0.676 (t=0.5) | 0.908 (t=0.5) | 0.887 | 0.907 (t=0.5) |
| General | **0.953** | **0.953 (t=0.6)** | 0.380 (t=0.8) | **0.953 (t=0.6)** | 0.952 | 0.949 (t=0.6) |

Table 2: BCubed results for SuperPart and unsupervised algorithms on new datasets processed after CPA application. SuperPart has no threshold to select and we show the best found threshold values for all other partitioning algorithms for each dataset.

| Dataset | SuperPart | Baseline | Avg. Edge |
|---|---|---|---|
| IC-Random | **0.898** | 0.839 | 0.807 |
| IC-Stratified | **0.851** | 0.838 | 0.845 |

Table 3: AUPRC results on the binary classification task of predicting whether a group is inconsistent. Inconsistent groups need to be split, usually by human review. SuperPart exceeds the baselines on both datasets.

For some record linkage problems at Amazon, we use cluster confidence to prioritize which work is reviewed by human auditors. As a proxy measure of the usefulness of SuperPart's clustering confidence scores, we compared a SuperPart model trained on these labels to an internal baseline described in section 6. Against relevance labels assigned by human auditors, SuperPart's AUPRC improved the baseline by 1.55% and 7.03% on two datasets respectively (see Table 3).

By including both descriptive features (section 4.2.2) and proposers-as-features (section 4.2.1), we allow the model to learn the right balance of expectation that a particular proposer will suggest the right clustering versus the ability for descriptive characteristics to indicate a good clustering. This balance may vary from dataset to dataset. On the Inconsistent-Stratified dataset, the top three features by feature importance [11] include performance (0.274 importance) and two proposers: Transitive closure with $t = 0.5$ (0.234 importance) and Merge-Center with $t = 0.5$ (0.121 importance). On the Inconsistent-Stratified dataset the top three are all proposers: Transitive closure at two thresholds ($t = 0.8$ with 0.109 importance and $t = 0.7$ with 0.094 importance) and Merge-Center with $t = 0.6$ (0.933 importance).

In the future, we will explore more sophisticated methods of problem instance creation in order to expand the utility of this method to the more general graph clustering problems. Picking a simple transitive closure threshold has worked surprisingly well in record linkage problems where the graphs are sparse and the clusters are small, but in a more general context, picking these thresholds introduces errors in the first step from which the second step can't recover. Another promising direction for future work is learning graph embeddings to project the similarity graph into a continuous vector space. Such an approach may reduce our reliance on numerous, hand-curated descriptive features, which each contribute their own signal and noise to the supervised proposal picking model.

# 8. CONCLUSION

In this paper, we describe a novel supervised approach to graph clustering called SuperPart. This method is an extensible approach for incorporating ground truth cluster-wise labels into clustering problems in the record linkage domain. SuperPart reduces the tedious, error-prone algorithm selection and tuning process and produces a useful clustering confidence score tailored to the specific problem domain. For auditing product families within Amazon, this improves the AUPRC by 11% compared to an internal baseline. Lastly, we contribute to the research community by releasing three Amazon real-world, anonymized datasets containing ground truth cluster labels and edge weights. These datasets can be used to benchmark future graph clustering work on real data, avoiding the over-reliance on synthetically created datasets.

# 9. REFERENCES


[1] N. Aghaeepour, G. Finak, H. Hoos, T. R. Mosmann, R. Brinkman, R. Gottardo, R. H. Scheuermann, F. Consortium, D. Consortium, et al. Critical assessment of automated flow cytometry data analysis techniques. *Nature Methods*, 10(3):228–238, 2013.

[2] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, 12(4):461–486, 2009.

[3] S. M. Ash and K. Ip-Lin. Embracing the sparse, noisy, and interrelated aspects of patient demographics for use in clinical medical record linkage. *AMIA Summits on Translational Science Proceedings*, 2015:425, 2015.

[4] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[5] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *International Journal on Very Large Data Bases (VLDB)*, 18(1):255–276, 2009.

[6] S. Chen, A. Borthwick, and V. R. Carvalho. The case for cost-sensitive and easy-to-interpret models in industrial record linkage. In *International Workshop on Quality in Databases*. VLDB, 2011.

[7] X. L. Dong and D. Srivastava. Big data integration. In *29th International Conference on Data Engineering (ICDE)*, pages 1245–1248. IEEE, 2013.

[8] A. J. Enright, S. Van Dongen, and C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research*, 30(7):1575–1584, 2002.

[9] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[10] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[11] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer Series in Statistics, New York, 2001.

[12] A. Gruenheid, X. L. Dong, and D. Srivastava. Incremental record linkage. *Proceedings of the VLDB Endowment*, 7(9):697–708, 2014.

[13] F. Harary. *Graph theory*. Addison-Wesley, Reading, MA, 1969.

[14] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2:1282–1293, 2009.

[15] O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. *International Journal on Very Large Data Bases (VLDB)*, 18(5):1141–1166, 2009.

[16] T. Haveliwala, A. Gionis, and P. Indyk. Scalable techniques for clustering the web. *Proceedings of WebDB*, 2000.

[17] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD Record*, volume 24, pages 127–138. ACM, 1995.

[18] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[19] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

[20] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 802–803. ACM, 2006.

[21] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[22] W. P. McNeill, H. Kardes, and A. Borthwick. Dynamic Record Blocking: Efficient Linking of Massive Databases in MapReduce. In *Quality in Databases*, 2012.

[23] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas. Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. *Proceedings of the VLDB Endowment*, 9(9):684–695, 2016.

[24] A. Saeedi, E. Peukert, and E. Rahm. Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution. *Advances in Databases and Information Systems*, pages 278–293, 2017.

[25] L. Shu, A. Chen, M. Xiong, and W. Meng. Efficient spectral neighborhood blocking for entity resolution. In *IEEE 27th International Conference on Data Engineering (ICDE)*, pages 1067–1078. IEEE, 2011.

[26] S. van Dongen. *Graph clustering*. PhD thesis, 2000.

[27] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, pages 219–232. ACM, 2009.

[28] D. T. Wijaya and S. Bressan. Ricochet: A family of unconstrained algorithms for graph clustering. In *International Conference on Database Systems for Advanced Applications*, pages 153–167. Springer, 2009.

[29] C. Wiwie, J. Baumbach, and R. Röttger. Comparing the performance of biomedical clustering methods. *Nature Methods*, 12(11):1033–1038, 2015.

[30] R. Xu and D. C. Wunsch. Clustering algorithms in biomedical research: A review. *IEEE Reviews in Biomedical Engineering*, 3:120–154, 2010.

[31] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.