Pay-as-you-go Configuration of Entity Resolution

Ruhaila Maskat, Norman W. Paton, and Suzanne M. Embury

School of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, UK
(maskatr@cs.man.ac.uk,npaton,suzanne.m.embury)@manchester.ac.uk

Abstract. Entity resolution, which seeks to identify records that represent the same entity, is an important step in many data integration and data cleaning applications. However, entity resolution is challenging both in terms of scalability (all-against-all comparisons are computationally impractical) and result quality (syntactic evidence on record equivalence is often equivocal). As a result, end-to-end entity resolution proposals involve several stages, including *blocking* to efficiently identify candidate duplicates, *detailed comparison* to refine the conclusions from blocking, and *clustering* to identify the sets of records that may represent the same entity. However, the quality of the result is often crucially dependent on configuration parameters in all of these stages, for which it may be difficult for a human expert to provide suitable values. This paper describes an approach in which a complete entity resolution process is optimized, on the basis of feedback (such as might be obtained from crowds) on candidate duplicates. Given such feedback, an evolutionary search of the space of configuration parameters is carried out, with a view to maximizing the fitness of the resulting clusters. The approach is pay-as-you-go in that more feedback can be expected to give rise to better outcomes. An empirical evaluation shows that the co-optimization of the different stages in entity resolution can yield significant improvements over default parameters, even with small amounts of feedback.

1 Introduction

Entity resolution is the task of identifying different records that represent the same entity, and is an important step in many data integration and data cleaning applications [11, 21]. A single entity may come to be represented using different records for many reasons; for example, data may be integrated from independently developed sources that have overlapping collections (e.g., different retailers may have overlapping product lines), or a single organization may capture the same data repeatedly (e.g., a police force may encounter the same individual or address many times, in situations where it may be difficult to be confident of the quality of the data). As a result, diverse applications encounter situations in which it is important to ascertain which records refer to the same entity, to allow effective data analysis, cleaning or integration.

In practice, given large collections, it is impractical to perform a detailed all-against-all comparison, which is $O(n^2)$ on the number of records. As a result, entity resolution tends to involve three principal phases: (i) *blocking*, whereby pairs of records that are candidate duplicates are identified using inexpensive, approximate comparison schemes; (ii) *detailed comparison* in which a distance function compares properties of the candidate duplicates from blocking; and (iii) *clustering*, whereby the records that have been identified as candidate duplicates through blocking are grouped into clusters on the basis of the results of the distance function. There is lots of work on each of these phases; for example, Christen [4] provides a survey of indexing techniques that support blocking, and Hassanzadeh *et al.* [14] compare clustering algorithms that take as input the results of blocking[1].

In an ideal world, off-the-shelf entity resolution techniques would be applied, with minimal manual intervention, to generate dependable results. However, in practice, the quality of the result of entity-resolution techniques is often crucially dependent on configuration parameters in all of *blocking*, *detailed comparison* and *clustering*. Setting these parameters directly is not straightforward for human experts; the impact of changes to parameters such as thresholds may be wide-ranging and difficult to predict, and there may be subtle inter-dependencies between parameters.

The importance of configuration of entity resolution has been widely recognised, and there are several results on different aspects of the problem. For example, blocking algorithms may use a subset of the fields in a record as the basis for comparison with other records, and apply a function that, given a record, generates one or several blocking keys, each of which gives rise to an index entry; such functions depend on the data to which the algorithm is to be applied, and manual tuning is often both laborious and difficult [4]. Indeed, several results have been reported that seek to automate (often by learning from training data) suitable parameter values for blocking schemes [3, 12, 26]. It is a similar story for clustering, where algorithms generally make use of thresholds, which also depend on the data that is being clustered. Although there has been some work on the tuning of comparison functions for pairwise matching (e.g. [13, 16]), we know of no previous work that seeks to co-optimize the complete lifecycle, from blocking to clustering. The most closely related proposal is probably Corleone [13], which also seeks to optimize the complete entity resolution process, but which differs in optimizing different aspects of an entity resolution pipeline in sequence rather than together, in emphasizing comparison functions rather than wider configuration parameters, and in focusing on pairwise comparison rather than clustering.

In this paper we investigate a pay-as-you-go approach to configuration of a complete entity resolution process. Given feedback on candidate duplicates, the

[1] Some entity resolution proposals carry out blocking and pairwise comparison, but stop short of clustering; this is fine up to a point, but clustering proposals are more comprehensive, in that they make the additional decisions as to which groups of candidate duplicates belong together.

approach explores the space of configuration parameters with a view to maximising the quality of the resulting clusters. Thus the entity resolution process is configured automatically in the light of feedback. The approach is pay-as-you-go, in that users can provide as little or as much feedback as they like, reviewing the results in the light of the feedback provided to date, and supplying additional feedback until they are satisfied. An empirical evaluation investigates the trade-off between the amount of feedback provided and the quality of the result.

In this paper, we use feedback that confirms or rejects candidate duplicates to configure the complete entity resolution process. Assume we have an entity resolution system $E(D, P)$, which given a collection of records (represented as tuples in this paper) in a data set D and a set of configuration parameters P (such as similarity thresholds), returns a set of clusters C, such that each cluster $C_i \in C$ is a subset of D. Given feedback on record pairs from D that indicates whether or not the records in the pair are actually duplicates, the problem is to identify parameters P that maximize the quality of the clusters C. Our approach is to use an evolutionary search [24] for parameter values that maximize cluster quality with respect to user feedback, for an existing, state-of-the-art, entity resolution proposal [5] that combines blocking and clustering. As such, this paper is an application of closed-loop evolutionary optimization [19] to entity resolution.

The contributions of the paper are as follows:

1. A generic approach to configuration of parameters for entity resolution that uses feedback on the correctness (or otherwise) of candidate duplicates. These parameters tune the blocking and clustering algorithms, and configure the distance function that is used to compare pairs of records.
2. A description of the application of that approach to produce a self-optimizing pay-as-you-go entity resolution system that uses an evolutionary search over the space of parameter values in which the fitness of alternative sets of parameters is assessed against feedback received on candidate duplicates from blocking.
3. An evaluation of the resulting platform with real world data sets, which shows substantial improvements in cluster quality, even with small amounts of feedback.
4. As the proposal can be considered to be computationally expensive, we describe and evaluate an approach that seeks to retain the results from (3) while also scaling to large data sets.

The paper is structured as follows: Section 2 provides the technical context for the remainder of the paper, introducing the terminology and prior results on which later sections build. Section 3 describes our approach to pay-as-you-go entity resolution, which is then evaluated in Section 4. An approach to addressing the computational overheads of the approach is presented in Section 5. Related work is discussed in Section 6, and conclusions follow in Section 7.

2 Technical Context

As discussed in the introduction, entity resolution proposals commonly involve two phases, blocking and clustering.

Blocking, given a data set D of type T, associates each element $d_i \in D$ with a set of other elements $M_i \subset D$ such that each element in M_i is a *candidate duplicate* for d_i. In practice, as surveyed by Christen [4], blocking typically yields an index $I : T \rightarrow \{T\}$, where the index may be based on a subset of the attributes of T or some form of string pattern such as n-grams. Requirements of blocking are that it should be efficient, and that M_i should contain most of (preferably all) the actual duplicates of d_i (and preferably not vast numbers of non-duplicates).

Clustering, given a data set D and an index I from blocking, returns a set of clusters C, such that each cluster $C_i \in C$ is a subset of D.

In this paper we do not exhaustively re-review related work on blocking [4] or clustering [14]; the contribution of this paper is on pay-as-you-go configuration of entity resolution, and not on entity resolution techniques *per se*. As such, we do not develop a new entity-resolution proposal, but rather demonstrate our approach on an existing state-of-the-art proposal [5] that is described in this section. We have chosen this proposal because: (i) the blocking phase, in employing a q-gram based hashing scheme, is using an approach that has been shown to be effective in a recent comparison [4]; (ii) the clustering algorithm meets all the generic requirements from [14], in that it is *unconstrained* (the number of clusters is not known in advance), *unsupervised* (clusters can be produced without training data) and *disjoint* (there is no overlap in the membership of clusters); (iii) the clustering algorithm is incremental, enabling new data to be incorporated as it becomes available; (iv) the full entity resolution process, from blocking, through pairwise comparison to clustering is included within a single proposal; and (v) the approach has been shown to be scalable in empirical studies [5]. Although we present our pay-as-you-go configuration approach in the context of this particular proposal, the overall approach is not specific to this technique, and could be applied to configure other approaches.

2.1 Blocking

Blocking, given a data set D of type T, creates an index $I : T \rightarrow \{T\}$ that, given a tuple $d_i \in D$, can retrieve a set of elements $M_i \subset D$ such that each element in M_i is a *candidate duplicate* for d_i. A *candidate duplicate* in this context is a record for which there is some evidence for its equivalence, but where the quality of the comparison may have been traded off for speed. The central questions for blocking are: *what type of index to use* and *how to construct index keys*. In the algorithm of Costa *et al.* [5], hash indexes are used, that associate each tuple d_i with all other tuples that have identical keys, where the keys use an encoding scheme that captures syntactic similarities between the tuples.

Specifically, each tuple may be associated with several keys, each generated using different hash functions, following Algorithm 1. Given a tuple t and configuration parameters P, HASH returns a collection of keys for t. The configuration

Algorithm 1: HASH(Tuple t, Parameters P)

```
1 for i ← 1 to P.numKeys do
2     r ← {h_k|a_k ∈ t, h_k ← min{H_i^1(g)|
3         g ←Q-GRAM(a_k, P.q)}}
4     key_i = ""
5     for j ← 1 to P.keyComponents do
6         key_i ← key_i++min{H_j^2(h)|h ∈ r}
7 return < key_1, ..., key_{P.numKeys} >
```

parameters provide: the number of hash keys to be generated (*P.numKeys*), the number of components in each hash key (*P.keyComponents*) and the size (*P.q*) of q-grams to use for approximate matching. Costa *et al.* assume the presence of two collections of hash functions, H_i^1 and H_j^2, that carry out first and second level hashing (i.e. second level hashing is applied to the result of first level hashing, as described below). These collections of hash functions can be used to generate several keys for each tuple within blocking. HASH proceeds as follows:

- For each of the keys to be generated (line 1) a representation r is created that contains, for each attribute a_k of t, the minimum value obtained when H_i^1 is applied to the q-grams of a_k (lines 2, 3). Thus a single hash code is generated to represent each attribute in t, in such a way that the probability that two values will be assigned the same hash value increases with the overlap between their q-grams.
- The key generated for each tuple is obtained by concatenating (using ++) together *P.keyComponents* elements from the representation r (line 5). The specific component that contributes to the jth position in the key is the minimum value obtained when H_j^2 is applied to each of the elements in r (line 6).
- The result of the function is a collection of *P.numKeys* keys (line 7).

To take an example, assume we have two tuples representing person data, each with attributes for forename, surname, street and city: t_1 = *¡David, Cameron, 10 Downing Street, London¿* and t_2 = *¡Dave, Cameron, Downing Street, London¿*. Assume that the representation r produced in (line 2) for each of the tuples is as follows $r_{t_1} = < h_1, h_2, h_3, h_4 >$, and $r_{t_2} = < k_1, k_2, k_3, k_4 >$, where each h_i and k_j is a hash code. As the surname and city attributes have identical values, $h_2 = k_2$ and $h_4 = k_4$. As the forename and street attributes are similar, and thus have several q-grams in common, there is a good likelihood but no guarantee that $h_1 = k_1$ and that $h_3 = k_3$. Assume that the number of key components is 2, and that the key constructed for t_1 is $H_1^2(h_1)$++$H_2^2(h_2)$. There is then a reasonable likelihood that the first component of the key for t_2 will be the same as for t_1 and a very strong likelihood that the second component of the key will be the same, although it is possible that either or both will be different and thus that the index keys of the tuples will not match.

Algorithm 2: PopulateClusters(Clusters C, Index I, Set<Tuple> NewTuples, Parameters P)

```
1 for t ∈ NewTuples do
2     N ←kNearestNeighbors(t, I, P)
3     c ←MostLikelyCluster(N, C, t, P)
4     if c == null then
5         newCluster ←new Cluster(t);
6         C ← C ∪ newCluster
7     else
8         c ← c ∪ t
9 Return C
```

In constructing the index there is a trade-off: the more index entries there are for a tuple (i.e. the higher is *P.numKeys*), the more the recall of blocking (the fraction of the correct tuples that are returned) will rise while its precision (the fraction of the returned tuples that are correct) will fall. By contrast, increasing the number of key components increases the precision of blocking but reduces recall. An empirical evaluation of these features is provided in the original paper on the technique [5].

2.2 Clustering

Given an optional set of existing clusters C, an index from blocking I, a set of tuples $NewTuples$ and some configuration parameters P, PopulateClusters updates the clusters C to take into account the presence of the $NewTuples$. As a result, this algorithm can be used to cluster an entire data set in one go, or incrementally to cluster data as it becomes available. The top level of the algorithm is given in Algorithm 2, which proceeds as follows:

- For each new tuple t (line 1), its K nearest neighbors are retrieved as N. The function kNearestNeighbors (line 2) returns up to $P.K$ entries from the index I that are the closest to t according to a Distance function (described below), and also above a *similarityThreshold* from P.
- The most likely cluster for t is identified by MostLikelyCluster (line 3), which returns a cluster from the clusters of the tuples in N, following a voting procedure involving the neighbors in N. In essence, each neighbor of t, $n_t \in N$, adds a contribution $\frac{1}{Distance(t,n_t)}$ to the score of its cluster, and the new tuple t is considered to be a candidate to be returned by MostLikelyCluster whenever its score from voting exceeds a *membershipThreshold* from P.
- Where a cluster is identified by MostLikelyCluster, t is added to this cluster (line 8), otherwise a new cluster is created with t as its only member (lines 5 and 6).

Algorithm 3: EntityResolution(Set<Tuple> Data, Parameters P)

```
1 I ← new Index();
2 for d ∈ Data do
3 |   Keys =Hash(d, P)
4 |   for k ∈ Keys do
5 |   |   I.Insert(k, d)
6 C ← new Clusters();
7 PopulateClusters(C, I, Data, P)
8 return C
```

The Distance function, given two tuples s and t, computes a weighted sum of the n-gram distance (denoted *dist*) of the attributes, so for two tuples $s =< a_{s,1}, ..., a_{s,n} >$ and $t =< a_{t,1}, ..., a_{t,n} >$, their distance is $w_1 \times dist(a_{s,1}, a_{t,1}) + ... + w_n \times dist(a_{s,n}, a_{t,n})$, where the weights are from P.

Algorithm 3 provides the top-level pseudo-code of EntityResolution that, given some *Data* to be clustered and control parameters P, shows how Hash and PopulateClusters can be used together to create a new data clustering.

2.3 Configuration Parameters

The algorithms described above make a range of different decisions during both blocking and clustering, where these decisions can be tuned using configuration parameters; the parameters are summarized in Table 1. *The hypothesis that this paper sets out to test is that these parameters can be optimized in the light of feedback to improve the quality of clustering for records for which no feedback has yet been obtained.* Although several of these parameters are strategy-specific, blocking and clustering strategies tend to make similar types of decision, so we suggest that these provide a representative collection of tuning parameters for a study on parameter tuning.

3 Pay-as-you-go Clustering

3.1 Overview

Automatic entity resolution is challenging because duplicate records typically manifest representational heterogeneities and data level inconsistencies that it is difficult for computer systems to unravel. As discussed in general terms in Section 1, and as detailed for a specific entity resolution strategy in Section 2.3, entity resolution techniques make decisions that are often guided by configuration parameters, the most effective settings for which may be data specific, and which may be challenging to set manually. This section details our approach to optimizing parameter setting, in the light of feedback, where the incremental collection of feedback allows a pay-as-you-go approach.

Parameter	Description	Type	Optimization Range
numKeys	Number of keys per tuple in HASH	Integer	$1 \leq numKeys \leq 9$
keyComponents	Number of values contributing to a key in HASH	Integer	N/A – $keyComponents = 1$
q	Size of q-gram in HASH	Integer	N/A – $q = 3$
H_i^1	Index into collection of first level hash functions	Integer	$1 \leq H_i^1 \leq 100$
H_i^2	Index into collection of second level hash functions	Integer	$1 \leq H_i^1 \leq 100$
K	Nearest neighbors returned by KNEARESTNEIGHBORS	Integer	$1 \leq k \leq 20$
similarityThreshold	Maximum distance between in KNEARESTNEIGHBORS candidate duplicates	Float	$0 \leq similarityThreshold \leq 1.0$
membershipThreshold	Voting threshold in MOSTLIKELYCLUSTER	Float	$0 \leq membershipThreshold \leq 1.0$
w_i	Attribute weights in DISTANCE, such that there is one weight per attribute	Float	$0 \leq w_i \leq 1.0$

Table 1. Entity resolution algorithm parameters.

We cast the search for effective parameters as an optimization problem. Given an objective function of the form FITNESS*(Clusters C, Feedback F)* that indicates the quality of the set of clusters C in the light of the feedback F, the problem is to search for configuration parameters that yield clusters that maximize FITNESS. Rather than using a model of the problem to estimate the quality of a candidate solution (as, for example, is standard practice for query optimization), the search takes place over the space of possible parameter settings, and the FITNESS is computed using clusters produced by running ENTITYRESOLUTION with candidate parameters over real data and feedback. Such an approach is followed because there is no known model for predicting the quality of the result of the entity resolution process given its parameters.

For entity resolution, we assume that feedback takes the form of annotations on pairs of records that are candidate duplicates, that confirm or refute that a pair of records are truly duplicates; such feedback is also obtained in [31, 33]. We then require a fitness measure for the outcome of the entity resolution process, the clusters, that builds on this feedback. Our fitness function uses the fraction of the feedback that has been correctly clustered as an estimate of the fitness that results from the use of a set of configuration parameters. In deriving this fraction, we use the notation FC to denote counts derived from the feedback and the clustering, where F and C can each take the values M or U. The first character, F, represents the status of a pair of records in the Feedback: an M indicates that a pair of records match (i.e. they are duplicates); and a U indicates that a pair of records is unmatched (i.e. they are not duplicates). The second character, C represents the status of a pair of records in the clusters: an M indicates that a pair of records match (i.e. are duplicates); and a U indicates that a pair of records is unmatched (i.e. are not duplicates). Thus the notation:

- *MM* denotes the number of records that are matched in the feedback that are also matched in the clusters.
- *UU* denotes the number of records that are unmatched in the feedback that are also unmatched in the clusters.
- *MU* denotes the number of records that are matched in the feedback that are unmatched in the clusters.
- *UM* denotes the number of records that are unmatched in the feedback that are matched in the clusters.

The FITNESS of a clustering in the context of some feedback is then defined as:

$$\frac{MM+UU}{MM+MU+UU+UM}$$

Intuitively, this is the fraction of the feedback which is correctly represented in the clusters. We note that this notion differs from existing fitness measures for clusters, but that existing measures tend either: (i) to assume access to the ground truth (e.g. [14]), of which we have access to only a portion via feedback; or (ii) to relate to general properties of clusters such as inter- and intra-cluster distance (e.g. [23, 27]) that may not be good indicators of how appropriately values have been assigned to clusters.

3.2 Evolutionary Search

This section describes how an evolutionary search, in particular a genetic algorithm, can be used to optimize the configuration parameters of the entity resolution technique from Section 2. We chose to employ a genetic algorithm because: (i) the search space is large, precluding the use of an exhaustive search; (ii) the search acts over a heterogeneous collection of configuration parameters, which are nevertheless easily accommodated in genetic algorithms; (iii) a genetic algorithm often takes less time to complete a task than other search methods [25], which is important in our context because of the high cost of clustering which is used to support fitness evaluation; and (iv) fitness evaluation can be easily parallelized in an evolutionary search.

We continue with some terminology [24]. An evolutionary search maintains an evolving *population* of interim solutions (individuals), which is subject to changes inspired by genetics. In our case, each member of the population consists of a collection of values for configuration parameters that control how entity resolution is carried out (see Table 1); the overall approach should be able to be applied to other entity resolution strategies using a population that captures their configuration parameters in place of those in Table 1). A *parent* is a member of a population that has been chosen to participate in the production of the next population (or generation) by way of *mutation* and *crossover* operators. A *mutation* introduces a random change to an individual member of the population, whereas a *crossover* combines features from two parents to produce a new member of the next population. A *fitness* function models requirements of the search problem, and informs the selection of parents, although to ensure diversity in the population not only the fittest solutions act as parents.

Algorithm 4: GeneticSearch(Set<Tuple>Data, Feedback F)

```
1  population ← initial collection of new random individuals
2  fitness ← initial assignment of 0 for each individual
3  bestIndividual ← null
4  bestFitness ← 0.0
5  n ← number of elite individuals to be retained
6  t ← tournament size
7  c ← crossover rate
8  m ← mutation rate
9  counter ← 0
10 while counter < generation size do
11     for i ← 1 to populationsize do
12         Clusters ← CLUSTER(Data, population[i])
13         fitness[i] ← FITNESS(Clusters, F)
14         if bestFitness < fitness[i] then
15             bestIndividual ← population[i]
16             bestFitness ← fitness[i]
17     nextPopulation ←the n fittest individuals in population
18     count ← 0
19     while count < populationsize/2 do
20         parent_a ← TournamentSelection(population, t)
21         parent_b ← TournamentSelection(population, t)
22         children ← Crossover(parent_a, parent_b, c)
23         nextPopulation = nextPopulation ∪ Mutate(children, m)
24         count = count + 1
25     population = nextPopulation
26     counter = counter + 1
27 Return bestIndividual
```

An evolutionary search starts from a set of random solutions that constitutes the initial population. The following steps are then repeated, as detailed in Algorithm 4, until a termination condition (e.g., the number of generations) is satisfied:

- **Fitness evaluation** assigns a fitness to each element in the population (lines 13-16). In our case, the fitness of an individual (i.e. a collection of values for configuration parameters) is obtained by generating a collection of clusters C using those configuration parameters, and then by using the definition of Fitness from Section 3.1 to assess these clusters in the light of the user feedback F.
- **Elite capture** maintains a set of the best solutions found to date (line 17); the elites then have a role in the construction of future generations, and have the effect of focusing the search on promising solutions. This focusing can reduce diversity in the population, but is used here because the high cost of fitness evaluation means that we cannot afford too many generations.

- **Parent selection** chooses from the current generation those that should be used as parents for the next generation (lines 20-21). We apply tournament selection, in which the likelihood of an individual being a parent is proportional to its fitness.
- **Crossover** is a binary variation operator on two parents that brings together properties of different individuals (line 22). We use one-point crossover, which splits both parents at a randomly selected element and exchanges their tails. The crossover is applied with a probability c, called the *crossover rate*.
- **Mutation** is a unary variation operator that manipulates a single parent, and is used to maintain diversity in a population (line 23). Each of the values in our population is numeric, and we use *gaussian convolution* to mutate these values in such a way that the values in the children are likely to be similar to those of their parents, but periodically are substantially different [24]. The mutation is applied with a probability m, called the *mutation rate*.

4 Evaluation of pay-as-you-go clustering

This section describes the experiments carried out to evaluate the effectiveness of the parameter optimization strategy described in Section 3, given different amounts of user feedback. The purpose of the evaluation is to investigate the extent to which the strategy can improve on default configurations, and the amount of feedback required to obtain such improvements.

4.1 Experimental Setup

Datasets We use three real datasets made available by University of Leipzig, for which the ground truth is provided with the dataset, that have been widely used in other studies of the performance of entity resolution techniques (e.g. [21, 22, 30, 31]):

- *DBLP-ACM*[2] is a data set containing bibliographic records from 2 data sets. There are a total of *12,051,595* pairs of records, of which *1083* represent the same entity. An example record is: *<672969, "An Effective Deductive Object-Oriented Database Through Language Integration","Maria L. Barja, Norman W. Paton, Alvaro A. A. Fernandes, M. Howard Williams, Andrew Dinn","Very Large Data Bases",1994>*.
- *Abt-Buy*[3] is a data set containing *2173* product records from 2 data sets. There are a total of *2,359,878* pairs of records, of which *1097* represent the same entity. An example record is: *<6493, "Denon Stereo Tuner - TU1500RD", "Denon Stereo Tuner - TU1500RD/ RDS Radio Data System/ AM-FM 40 Station Random Memory/ Rotary Tuning Knob/ Dot Matrix FL Display/ Optional Remote", $375.00>*.

[2] dbs.uni-leipzig.de/file/DBLP-ACM.zip
[3] dbs.uni-leipzig.de/file/Abt-Buy.zip

– *Amazon-Google*[4] is a data set containing *4598* product records from 2 data sets. There are a total of *10,527,166* pairs of records, of which *1,300* represent the same entity. An example record is: *<http://www. google.com/base/feeds /snippets/12244614697089679523, "production prem cs3 mac upgrad,adobe cs3 production premium mac upgrade from production studio premium or standard,adobe software",805.99>*.

Techniques We evaluate several different approaches to entity resolution, all of which generate clusters using the strategy described in Section 2. These approaches differ along three principal dimensions: (i) whether or not optimization takes place, as described in Section 3; (ii) when optimization does take place, whether only weights, or both weights and parameters participate in the search space; and (iii) whether or not the feedback is applied directly to tuple pairs in a way that overrides the score from the distance function used during clustering. In (iii), where a tuple pair is known from feedback to represent the same entity it can be given the minimum distance of *0*, and where a pair is known from feedback to represent different entities it can be given the maximum distance of *1*.

Approach	**Optimization**	**Parameters Optimized**	**Score Change**
Baseline	No	Not Applicable	No
NOSC	No	Not Applicable	Yes
WOO	Yes	w_i in Table 1	No
WAPO	Yes	All of Table 1	No
POSC	Yes	All of Table 1	Yes

Table 2. Approaches compared.

The approaches evaluated are as follows, where their key features are summarized in Table 2:

– *Baseline:* this method makes no use of feedback, and simply applies the entity resolution strategy from Section 2 directly. As such, the baseline has been subject to manual configuration. The weights w_i used by the DISTANCE function in Table 1 are problem-specific, and were set to values that give higher values to properties that seem likely to be of greater significance for matching, specifically: DBLP-ACM – (title: 0.8, authors: 0.4, venue: 0.6, year: 0.7); AbtBuy – (product_name: 0.6, description: 0.8, price: 0.2); and Amazon-Google – (product_name: 0.8, description: 0.4, manufacturer: 0.6, price: 0.2). The other parameter values in Table 1 were set as follows: $numKeys = 9$, $keyComponents = 1$, $q = 3$, H^1 and H^2 are chosen at random, $K = 10$,

[4] dbs.uni-leipzig.de/file/Amazon-GoogleProducts.zip

$similarityThreshold = 0.25$, $membershipThreshold = 0.5$. Where specific values have been given, these were always based on positive experiences with those values in practice, so the baseline should be representative of what can realistically be achieved based on general experience.

- *No-optimization score change (NOSC):* this method applies the entity resolution strategy from Section 2 without parameter optimization, but with scores changed to reflect feedback that confirms or refutes that tuple pairs represent the same entity. This is to allow us to understand the effectiveness of a strategy in which the feedback is used to change scores but not to inform the configuration of the entity resolution process.
- *Weights-only optimization (WOO):* this method uses the optimization method from Section 3, but only searches over the weights used in the distance function (w_i in Table 1). This is to allow us to understand the relative impacts of optimization of the distance function and the optimization of the control parameters.
- *Weights-and-parameters optimization (WAPO):* this method uses the optimization method from Section 3, and searches over not only the weights but also the control parameters in Table 1.
- *Post-optimization score change (POSC):* this method uses the optimization method from Section 3, and searches over all the parameters in Table 1, but *after* the optimization scores are changed to reflect feedback that confirms or refutes that tuple pairs represent the same entity, and the dataset is reclustered with the parameters from the optimization and the changed scores[5]. This it to allow us to understand if optimization of the weights and parameters is sufficient to bring about appropriate clustering of the records on which feedback has been obtained.

We are unable to carry out a direct head-to-head comparison with related work, such as that discussed in Section 6, as there is no other proposal that covers parameter setting and distance function tuning, from blocking to clustering.

Scope of optimization Note that where the optimization is referred to as acting on the control parameters in Table 1, in fact for three of these parameters (q, *keyComponents* and *numKeys*), a sensitivity analysis identified that specific values ($3, 9$ and 1 respectively) always yielded the best results for our data sets, and thus these do not participate in the optimization.

[5] Note that in principle, it is also possible to have some form of *pre-optimization score change* strategy, which changes scores to reflect feedback that confirms or refutes that tuple pairs represent the same entity *before* optimization takes place. However, in practice, such an approach was found to be ineffective for optimizing configuration parameters because the score changes in themselves tend to be sufficient to ensure appropriate clustering of the feedback. Thus changes to configuration parameters tend to have little impact on fitness, and are not necessarily helpful for informing the clustering the records for which feedback has not been obtained.

Evolutionary search The optimization was implemented using the ECJ evolutionary computing system[6]. Following experimentation with different values, it was ascertained that using a *population size* of *50* and a generation size of *70* yielded results that were both stable and close in quality to those obtained with more searching; as time consuming entity resolution takes place during optimization, it is important that the search can converge on appropriate answers after a modest number of generations.

Feedback generation The experiments use feedback that is generated algorithmically from the ground truth. In essence, the feedback is a subset of the ground truth, which assigns an annotation *match* or *unmatch* to pairs of candidate duplicates. The approach makes few specific assumptions about how the feedback is obtained, although we do need both *match* and *unmatch feedback*; it is possible that targeted selection of feedback of the form discussed in Section 6 could produce improved results, but for now we use a straightforward approach to sample from the ground truth.

To generate *match* feedback, matching pairs are selected at random from the ground truth. Generating plausible *unmatch* feedback requires more care. In any data set, there are likely to be many more *unmatched* pairs than *matched* pairs, and in practice it makes little sense to ask users if a randomly selected pair of records match, as generally they will not match and there will be no reason to believe that they do. So, to generate *unmatch* feedback, the following steps take place: (i) generate the collection of *blocked pairs* by running the blocking algorithm described in Section 2 – *blocked pairs* should contain only pairs for which there is some evidence of syntactic similarity; and (ii) select pairs at random from the *blocked pairs* that have the annotation *unmatch* in the ground truth. In the experiments, we have identical amounts of *match* and *unmatch* feedback.

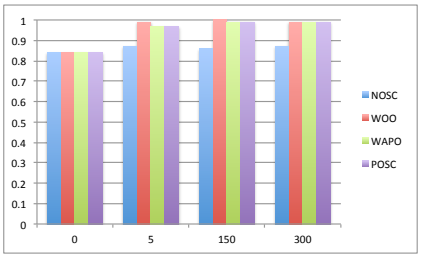
There is no fully accepted way of selecting data for feedback for entity resolution; indeed identifying the most appropriate data on which to obtain feedback is a research topic in its own right. The approach to feedback generation used in the experiment shares features with other work in the literature. For example, in investigating interaction with crowds for entity resolution, CrowdER [31] uses machine-based techniques to identify candidates, on which feedback is then obtained. This is analogous to our use of the blocking algorithm to identify candidates for *unmatch* feedback.

Evaluating Clusters The following formula is used to compute the fitness of clustering results with respect to the ground truth:

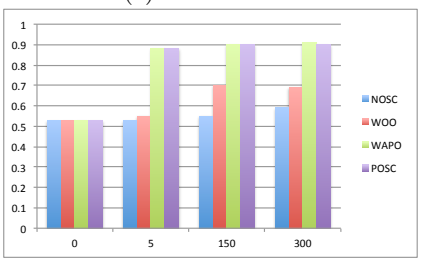
$$\left(\frac{M_G M}{M_G M + M_G U} + \frac{U_G U}{U_G U + U_G M}\right) \times \frac{1}{2}$$

where:

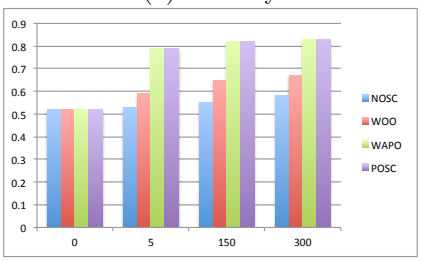
[6] cs.gmu.edu/ẽclab/projects/ecj/



(a) DBLP-ACM



(b) Abt-Buy



(c) Amazon-Google

Fig. 1. Experiment 1: cluster quality for different levels of feedback for all 4 methods. The results for no feedback are the baseline for the experiment.

- $M_G M$ denotes the number of records that are matched in the ground truth that are also matched in the clusters.
- $U_G U$ denotes the number of records that are unmatched in the ground truth that are also unmatched in the clusters.
- $M_G U$ denotes the number of records that are matched in the ground truth that are unmatched in the clusters.
- $U_G M$ denotes the number of records that are unmatched in the ground truth that are matched in the clusters.

This calculates the fraction of correctly clustered record pairs over all record pairs, giving equal importance to the correct clustering of the matching and unmatching portions of the result. This approach is adopted in a context where there are generally many more unmatched pairs than matched pairs, to ensure that the counts of unmatched pairs do not swamp those for matched pairs. The

formula used for evaluating the clusters in the experiments is similar to that used for estimating the fitness of clusters in Section 3.1. The main difference is that for evaluating clusters we have access to the ground truth, whereas the fitness function used when creating the clusters in Section 3.1 only has access to the feedback, which is an approximation of the ground truth. The overall approach of computing fractions of correctly clustered data in the evaluation is standard (e.g. [14]).

4.2 Results

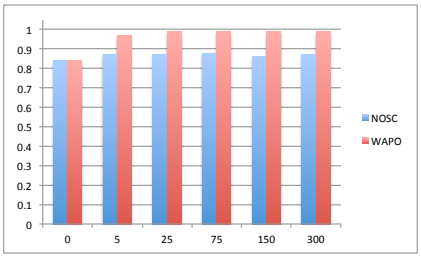
Experiment 1

Comparison of different approaches, for three different levels of feedback. The aim of this experiment is to understand the relative performance of the different methods from Table 2, so that subsequent experiments can investigate the most promising approaches in more detail. The experiment involves all three datasets (DBLP-ACM, AbtBuy and AmazonGoogle), and three different levels of feedback. Each item of feedback represents the annotation of a single candidate pair as *match* or *unmatch*. As the data sets have different sizes, as detailed in Section 4.1, the amounts of feedback experimented with range from annotations involving less than 1% of the records in each data set, up to individual annotations on around 6% of the records in DBLP-ACM and AmazonGoogle, and 14% of the records in AbtBuy.

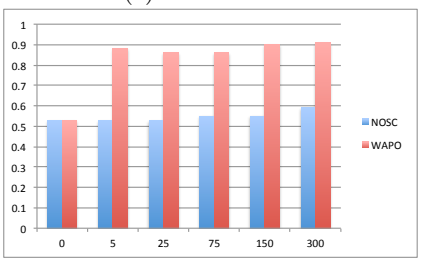
The results are in Figure 1(a) for DBLP-ACM, in Figure 1(b) for AbtBuy and in Figure 1(c) for AmazonGoogle. Where there is no feedback, this corresponds to the *Baseline* method described in Section 4.1. The following can be observed: (i) For NOSC, the feedback yields limited improvements in all cases – the local edits to scores are applied in the context of the default weights and parameter settings, and result in only small scale improvements to clusters. (ii) For WAPO and POSC, for all data sets, even a small amount of feedback yields a substantial improvement in performance. (iii) For the optimization based strategies, WOO, WAPO and POSC, the performance has leveled off by the time there are 150 items of feedback; other feedback amounts are considered in Experiment 2. (iv) The performance of WAPO is much better than for WOO in Abt-Buy and Amazon-Google, showing that there is benefit to be derived from including control parameters as well as weights in the optimization; several proposals from the literature focus on the optimization of comparison functions without also considering control parameters (e.g. [13, 15]). (v) The performance of POSC is very similar to that of WAPO, showing that WAPO generally clusters the data items for which there is feedback correctly, without the additional evidence that comes from changed scores.

Experiment 2

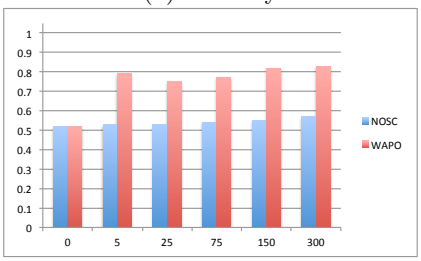
Comparison of selected approaches for different amounts of feedback. The aim of this experiment is to understand the rate at which the quality of a clustering



(a) DBLP-ACM



(b) Abt-Buy



(c) Amazon-Google

Fig. 2. Experiment 2: cluster quality for fine grained levels of feedback for NOSC and WAPO. The results for no feedback are the baseline for the experiment.

can be expected to improve as the amount of feedback collected grows. Thus this experiment is critical in terms of the cost-effectiveness of the pay-as-you-go approach. The experiment involves all three datasets (DBLP-ACM, AbtBuy, AmazonGoogle) and varying levels of feedback. The experiment focuses on NOSC and WAPO: NOSC as it represents a baseline in which the feedback is used but there is no optimization, and WAPO because it emerged as a strong proposal from Experiment 1.

The results are in Figure 2(a) for DBLP-ACM, in Figure 2(b) for AbtBuy and in Figure 2(c) for AmazonGoogle. The following can be observed: (i) In WAPO, small amounts of feedback yield results that are close to the results obtained with much larger amounts of feedback; this is a positive result, as it suggests that the approach is cost-effective in terms of feedback collection. For AbtBuy, 5 items of feedback involves around 0.2% of the records, and for DBLP-ACM

and AmazonGoogle, 5 items of feedback involves around 0.1% of the records. (ii) The highest quality measure obtained is always quite high, but varies between data sets. This is because in some data sets the syntactic evidence on similarity is stronger than in others; for example there are rarely inconsistencies in titles between the same paper in DBLP-ACM, but product descriptions for the same product are often significantly different in AbtBuy and AmazonGoogle. (iii) In AbtBuy and AmazonGoogle, for WAPO the result quality is slightly less good for *25* and *75* items of feedback than for *5* items of feedback. This is not especially surprising: (i) in all cases the feedback represents a small fraction of the ground truth, which means that the estimated fitness is subject to fluctuations based on the specific data for which feedback has been obtained; and (ii) the search for solutions is not exhaustive, and thus the search itself gives rise to variations in result quality.

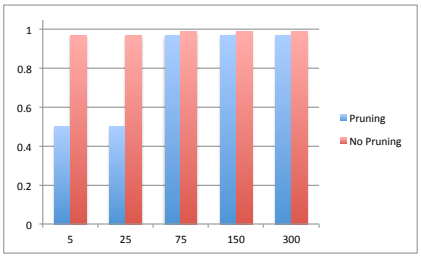
5 Scaling the approach

An issue with the approach in Section 3 is the high cost of evaluating the fitness function for the evolutionary search, which involves repeatedly applying the entity resolution technique from Section 2 on the data set for each candidate solution. Where there is a population size of p and g generations, this leads to $p \times g$ runs of the clustering algorithm. Although the search can readily be parallelized using platforms such as map/reduce [8] or Condor [29] so that the fitness of every element in the population at a generation is computed in parallel (indeed, our implementation runs using Condor on a campus grid), this can still be both resource intensive and lead to substantial elapsed times of broadly $g \times c$, were c is the cost of clustering. This section makes a proposal for reducing the cost of exploring the space of candidate solutions, and evaluates its effectiveness both in terms of runtime and cluster quality.

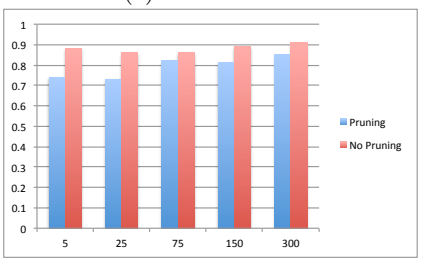
5.1 Clustering on Pruned Data Sets

With a view to reducing response times and resource usage, here we describe an approach to reducing c by clustering only the portion of the dataset on which feedback has been obtained. Recall from Section 3.1 that the fitness of a clustering is estimated from the fitness of the feedback (which represents the available subset of the ground truth). Thus although changes to the weights and parameters of the clustering apply to every item in the data set, the estimated fitness depends only on the clustering of the pairs for which we have feedback. Thus, with a view to reducing the computational cost that results from clustering records about which we have no feedback, clustering is run over a pruned dataset that consists of the records on which we have feedback, which is defined as follows:

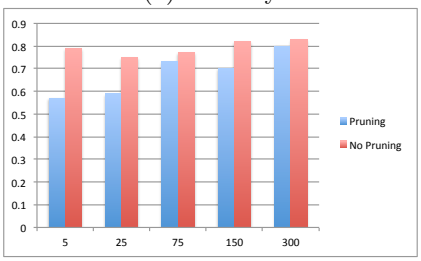
$$prunedDataset = \{r | r \in dataset, hasFeedback(r)\}$$



(a) DBLP-ACM



(b) Abt-Buy



(c) Amazon-Google

Fig. 3. Experiment 3: cluster quality for different levels of feedback using pruned and non-pruned data sets for WAPO.

where *dataset* is the collection of records to be clustered, and *hasFeedback* is true for a record if there is any feedback on that record.

Thus when pruning is used, during optimization the fitness of a candidate configuration is an estimate in two senses: (i) the fitness is based only on the feedback, and not on the ground truth, which is not available in practice; and (ii) the fitness is calculated over clusters that involve only a subset of the complete dataset, in contrast with Section 4.2. In reporting the results of the experiments, cluster quality is reported in terms of the ground truth for a clustering of the complete dataset, where that clustering was obtained using the configuration parameters obtained when optimizing using the pruned dataset.

Table 3 indicates the number of records in the pruned data sets for different amounts of feedback. Where the feedback amounts are small, the number of records is typically twice the number of items of feedback, as each item of feed-

Table 3. Pruned data set size for different feedback amounts.

Dataset	**Total Number**	**Feedback Amount**				
	of Records	5	25	75	150	300
DBLP-ACM	4910	10	50	150	292	572
AbtBuy	2173	10	50	146	278	535
Amazon Google	4589	10	50	149	295	578

back involves the relationship between two records. As the amount of feedback grows, there is a growing likelihood that a new item of feedback will involve at least one record for which there is already some feedback, and hence the number of records becomes less than twice the amount of feedback.

5.2 Evaluation of Pruning

This section empirically evaluates the effectiveness of the pruning approach in terms of runtime performance and the quality of the clusters produced. The purpose of the evaluation is to investigate the extent to which pruning can improve on the runtime required for a complete clustering, and the extent to which this reduced runtime is accompanied by reduced cluster quality.

Although the fitness function is applied only to records for which there is feedback, the pruning of records for which there is no feedback can be expected to impact on the clustering of the records for which there is feedback, for example because such records now have different neighbors. As such, it is important to evaluate the results of the pay-as-you-go entity resolution strategy to ascertain: (i) the extent to which effective weights and parameters are obtained when optimization takes place over the pruned data sets; and (ii) the performance improvement in terms of clustering times.

Experiment 3

Comparison of cluster quality for optimization using pruned data sets. The aim of this experiment is to understand the extent to which the quality of clusters produced for a given level of feedback is affected by optimization over pruned data sets. Thus this experiment is important in terms of the scalability of approach. The experiment involves all three datasets (DBLP-ACM, AbtBuy and Amazon-Google) with varying levels of feedback. The experiment focuses on WAPO as it has been shown to perform well, and because it depends for its performance on the effectiveness of the optimization.

The results are reported in Figure 3. The following can be observed: (i) In all cases, where there are 75 or more items of feedback, representing a single item of feedback on 2% to 4% of the records, the results obtained using pruned data sets significantly improve on the default parameters, for which the results were reported in Experiment 1. (ii) In all cases, there is a gap between the results

Table 4. Average runtimes (seconds) and speedup obtained using pruned data set with 300 items of feedback.

Dataset	Runtime: Full	Runtime: Pruned	Speedup
DBLP-ACM	300	5	60
AbtBuy	378	14	27
Amazon Google	9360	300	31

with and without pruning, which narrows as more feedback is obtained. The gap would be expected to narrow, as increased amounts of feedback leads to larger and more representative data sets being used in the assessment of the fitness of candidate solutions. (iii) For the smaller amounts of feedback, specifically *5* and *25* items, the gap between the pruned and non-pruned cases can be large, and the results using pruning are worse than the default parameters in DBLP-ACM. This is explained by the clustering taking place on tiny data sets, which are unrepresentative.

This experiment illustrates that there is an interesting trade-off between the two kinds of payment in pay-as-you-go entity resolution. Payment in the form of user feedback provides a more rapid return on investment when fitness is evaluated by clustering the complete data set. However, clustering the complete data set within the search is computationally expensive. By contrast, with the pruned data sets the cost of clustering can be significantly reduced, but good results can only be obtained where there is more feedback.

Experiment 4

Comparison of clustering times between original and pruned data sets. The aim of this experiment is to understand the impact of pruning on the runtime of clustering. The experiment uses all three data sets, and focuses on WAPO as it has been shown to be an effective strategy.

In practice, clustering times on complete data sets can vary quite significantly between runs (up to about a factor of 3), as different configurations lead to different collections of neighbors, etc. As such, to give a summary of the improvements that can be expected, Table 4 reports the average speedup obtained, averaged over 5 runs, using the pruned data set in place of the complete data set.

The results show that clustering with the pruned data set is several orders of magnitude faster than with the complete data set; even more impressive results can be obtained for more intensive pruning. This shows that pruning can be used with large data sets to very substantially reduce the cost of fitness function evaluation.

6 Related Work

This section discusses some of the most relevant related work, focusing on *learning/optimization for entity resolution* and *crowdsourcing for entity resolution*, both of which seek to improve the results of the entity resolution process on the basis of feedback or training data. In discussing these areas, although there are proposals that follow a pay-as-you-go approach, there is no other proposal that covers parameter setting and distance function tuning, from blocking to clustering.

We do not review the wider literature on entity resolution, where there are existing surveys on the topic as a whole [11, 20], on blocking [4] and on clustering [14].

In relation to *learning/optimization for entity resolution*, there has been work to inform both blocking strategies and more detailed comparison rules. Although not specifically following a pay-as-you-go approach, several relevant proposals have been made that learn or tune blocking schemes, for example using training data to inform the selection of properties that participate in blocking functions (such as [3, 26]), or by tuning similarity thresholds (e.g. [2]). More detailed rules for comparing values have also been learned using genetic programming [6], for example for identifying links between linked open data resources [15]. In addition, as entity resolution potentially acts over huge data sets, research has been carried out into the use of active learning for obtaining the most suitable training data [1, 16]. Such research complements the work in this paper; in this paper the focus is on co-optimizing blocking, comparison and clustering, and insights from work on each of these individual stages can help to identify opportunities for their combined optimization.

In our work, in order to evaluate the fitness of a candidate set of parameters, the full entity resolution process needs to be re-run for each candidate. Such an approach, known as closed-loop optimization, has been widely used in other domains to automatically set configuration parameters for physical systems; for example, Knowles [19] describes the use of evolutionary search techniques for tasks as diverse as optimizing instrument setup in analytical biochemistry, and improving chocolate production. This practice of searching for suitable parameters by running the actual system has also been employed for computing systems. For example, in iTuned [10], experiments are generated that investigate the effect of different system parameters on overall performance, in a context, like ours, where the development of an analytical cost model is not obviously a practical proposition. Furthermore, in reinforcement learning [18], learning is intrinsically closely associated with an ongoing process, with the normal behaviour of a system interleaved with learning steps that follow a trial-and-error approach. For entity resolution, de Freitas *et al.* [7] combine active learning with reinforcement learning, where the latter is used to evaluate the confidence of different committee members that are making detailed comparison decisions.

In relation to *crowdsourcing for entity resolution*, there is work on *identifying the most suitable data on which to crowdsource feedback* and on *applying the feedback to inform entity resolution decisions.*

For *identifying the most suitable data on which to crowdsource feedback*: CrowdER [31], focuses on the grouping of candidate entity pairs into tasks with a view to obtaining the required information while minimizing the number of crowd tasks, and thus the expense; and several proposals have investigated the identification of the pairs of records that are likely to be most valuable for refining decisions relating to entity resolution (e.g. [16, 30, 32, 33]). For example, in Isele *et al.*, [16], feedback is sought using active learning on the record pairs on which candidate detailed comparison rules disagree the most.

Such research complements the work in this paper, which could be used alongside techniques for efficient collection of feedback to improve overall return on investment.

For *applying the feedback to inform entity resolution decisions*, several proposals consult the crowd for specific purposes. ZenCrowd [9] identifies pairs of instances in linked data using two levels of blocking to identify candidate pairs for confirmation by the crowd. A probabilistic *factor graph* accumulates evidence from different sources, from which a probability is derived that a candidate pair is correct. The Silk Link Discovery Workbench [16] uses the crowd to judge whether candidate pairs are duplicates, and refines the collection of detailed comparison rules on the basis of this feedback. Perhaps the work that is closest to ours in terms of ethos is Corleone [13], which seeks to provide *hands-off crowdsourcing* for entity resolution, whereby the whole entity resolution process is automated, obtaining input from the crowd as needed. To do this, Corleone uses crowdsourcing to learn blocking and refinement rules in turn, and also addresses issues such as when to terminate a crowdsourcing activity. Our work is similar, in including several entity resolution phases, but is somewhat broader in scope in that it also optimizes related system parameters and considers clustering as well as pairwise matching. In addition, the approaches are significantly different; Corleone tackles blocking and matching in sequence, whereas in this paper, blocking, the distance function and other system thresholds and parameters are optimized at the same time. The price paid to support this co-optimization is the need to run blocking and clustering repeatedly within the evolutionary search, although we have shown that the costs associated with this can be reduced using pruning. In addition, by co-optimizing system parameters, our methodology reduces the risk that important but non-obvious interactions between parameters are overlooked during the application of the pay-as-you-go methodology.

Note that the term *pay-as-you-go* has been applied with different meanings in relation to entity resolution. In this paper, as in the above literature on crowdsourcing, the payment takes the form of feedback on the results of an entity resolution process, whereas in [34] the payment is in the form of computational resource usage.

7 Conclusions

We now revisit the claimed contributions of the paper from the introduction:

1. *A generic approach to configuration of parameters for entity resolution that uses feedback on the correctness (or otherwise) of candidate duplicates.* We have described an approach that uses closed loop optimization to evaluate the effectiveness of alternative configuration parameters, simultaneously optimizing all stages of the entity resolution process; this is a potentially important feature of the approach, as there may be subtle relationships between parameters across the process.
2. *A description of the application of that approach to produce a pay-as-you-go self-optimizing entity resolution system that uses an evolutionary search over the space of parameter values.* We have applied the approach from (1) to automate the optimization of the parameters for the state-of-the-art approach described in Costa *et al.* [5]; the optimized parameters include weights within the distance function, indexing parameters and similarity thresholds.
3. *An evaluation of the resulting platform with real world data sets, which shows substantial improvements in cluster quality.* The evaluation shows not only that the technique can be effective, providing significantly improved clustering compared with default parameters, but also that effective results can be obtained with surprisingly modest amounts of feedback (a single item of feedback on less than 1% of the records in each of the experimental data sets). The experiments also showed that optimizing the control parameters and distance function together was more effective than optimizing only the distance function, a heretofore popular strategy.
4. *As the proposal can be considered to be computationally expensive, we describe and evaluate an approach that seeks to retain the results from (3) while also scaling to large data sets.* A modification of the approach is described in which the entity resolution process, rather than testing the fitness of candidate configurations over the complete data set, instead evaluates fitness over the (typically much smaller) portion of the data set for which feedback has been obtained. The experimental evaluation shows that more feedback is required to obtain stable and effective results than when fitness is evaluated over the complete dataset during the search. However, even when using the pruned data set, significant improvements over the default parameters have been obtained with feedback on a small percentage of the data (a single item of feedback on less than 4% of the records in each of the experimental data sets). Pruning on such amounts of feedback can reduce runtime costs of clustering by several orders of magnitude.

There are several possible areas for future work. Although positive results have been demonstrated with small amounts of feedback, it would be interesting to ascertain if active learning [28] could focus feedback collection on values that are still more effective, in particular in the case of the pruning strategy. In the application of the approach, the distance function is fairly straightforward; it would be interesting to investigate the co-optimization of richer distance functions (e.g. as in [6]) with other configuration parameters. Although the overall closed-loop optimization approach is, in principle, applicable for configuration of different entity-resolution algorithms, it would be interesting to demonstrate

this in practice, and to ascertain the extent to which different parameters in different settings can be tuned effectively in this way. Furthermore, it would be interesting to investigate the use of optimization techniques that are specifically targeted at problems with expensive black-box fitness functions [17].

Acknowledgement. Research on data integration at Manchester is supported by the EPSRC VADA Programme Grant, whose support we are pleased to acknowledge.

References

1. A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 783–794, 2010.
2. G. D. Bianco, R. Galante, C. A. Heuser, and M. A. Gonçalves. Tuning large scale deduplication with reduced effort. In *SSDBM*, page 18, 2013.
3. M. Bilenko, B. Kamath, and R. Mooney. Adaptive blocking: Learning to scale up record linkage. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 87–96, 2006.
4. P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.
5. G. Costa, G. Manco, and R. Ortale. An incremental clustering scheme for data de-duplication. *Data Mining and Knowledge Discovery*, 20(1):152–187, 2010.
6. M. de Carvalho, A. Laender, M. Goncalves, and A. Da Silva. A genetic programming approach to record deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(3):399–412, 2012.
7. J. de Freitas, G. L. Pappa, A. S. da Silva, M. A. Gonçalves, E. S. de Moura, A. Veloso, A. H. F. Laender, and M. G. de Carvalho. Active learning genetic programming for record deduplication. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pages 1–8, 2010.
8. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.
9. G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *VLDB J.*, 22(5):665–687, 2013.
10. S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. *PVLDB*, 2(1):1246–1257, 2009.
11. A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):1–16, 2007.
12. K. Goiser and P. Christen. Towards automated record linkage. In *Proceedings of the fifth Australasian conference on Data mining and analystics - Volume 61*, AusDM '06, pages 23–31, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.
13. C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD Conference*, pages 601–612, 2014.

14. O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proc. VLDB Endow.*, 2(1):1282–1293, Aug. 2009.
15. R. Isele and C. Bizer. Learning expressive linkage rules using genetic programming. *PVLDB*, 5(11):1638–1649, 2012.
16. R. Isele and C. Bizer. Active learning of expressive linkage rules using genetic programming. *J. Web Sem.*, 23:2–15, 2013.
17. D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.
18. L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res. (JAIR)*, 4:237–285, 1996.
19. J. D. Knowles. Closed-loop evolutionary multiobjective optimization. *IEEE Comp. Int. Mag.*, 4(3):77–91, 2009.
20. H. Kopcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197 – 210, 2010.
21. H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493, 2010.
22. S. Lee, J. Lee, and S.-w. Hwang. Scalable entity matching computation with materialization. In *20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 2353–2356. ACM, 2011.
23. C. Legány, S. Juhász, and A. Babos. Cluster validity measurement techniques. In *5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, AIKED'06, pages 388–393. World Scientific, 2006.
24. S. Luke. *Essentials of Metaheuristics*. Lulu, 2013.
25. Z. Michalewicz and D. Fogel. *How to solve it: modern heuristics*. Springer-Verlag New York Inc, 2004.
26. M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 440–445. AAAI Press, 2006.
27. E. Rendón, I. M. Abundez, C. Gutierrez, S. D. Zagal, A. Arizmendi, E. M. Quiroz, and H. E. Arzate. A comparison of internal and external cluster validation indexes. In *2011 American Conference on Applied Mathematics and the 5th WSEAS International Conference on Computer Engineering and Applications*, AMERICANMATH'11/CEA'11, pages 158–163. World Scientific, 2011.
28. B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.
29. D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
30. N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *Proc. VLDB Endow.*, 7(12):1071 – 1082, 2014.
31. J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.
32. S. Wang, X. Xiao, and C. Lee. Crowd-based deduplication: An adaptive approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1263–1277, 2015.
33. S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.
34. S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *Knowledge and Data Engineering, IEEE Transactions on*, 25(5):1111–1124, 2013.