Waldo: An Adaptive Human Interface for Crowd Entity Resolution*

Vasilis Verroios
Stanford University
verroios@stanford.edu

Hector Garcia-Molina
Stanford University
hector@cs.stanford.edu

Yannis Papakonstantinou
UC San Diego
yannis@cs.ucsd.edu

ABSTRACT

In Entity Resolution, the objective is to find which records of a dataset refer to the same real-world entity. Crowd Entity Resolution uses humans, in addition to machine algorithms, to improve the quality of the outcome. We study a hybrid approach that combines two common interfaces for human tasks in Crowd Entity Resolution, taking into account key observations about the advantages and disadvantages of the two interfaces. We give a formal definition to the problem of human task selection and we derive algorithms with strong optimality guarantees. Our experiments with four real-world datasets show that our hybrid approach gives an improvement of 50% to 300% in the crowd cost to resolve a dataset, compared to using a single interface.

1. INTRODUCTION

In entity resolution, we are given a set of records and our objective is to find which records refer to the same real-world entity. Hence, the final outcome is a clustering of records, where in each cluster we have all the records that refer to the same entity. In many cases, humans are much better than machine algorithms in detecting *matches* (record pairs referring to the same entity) in a set of records (e.g., customer records that include the customer's photograph). This observation drives the idea of crowd entity resolution, where machine algorithms and humans are combined to perform entity resolution.

Crowd entity resolution is a topic studied by a number of papers over the last few years [18, 19, 22, 27, 32, 34, 35, 36, 37, 38]. Moreover, crowd entity resolution has received a lot of attention in the industrial level as well. Major companies such as Google, Bing, and Facebook are using this approach to create summary records of entities in web search or to remove duplicate entries on their maps, while startups like Tamr [6] are curating large datasets by involving humans in the loop.

*This research was supported by the National Science Foundation under grants no. 1009916 and no. 1447943.

SIGMOD'17, May 14-19, 2017, Chicago, Illinois, USA

DOI: http://dx.doi.org/10.1145/3035918.3035931

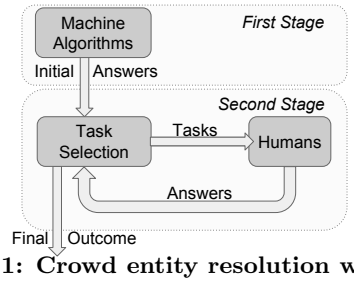


Figure 1: Crowd entity resolution workflow.

Figure 2: Multi-item interface.

The typical workflow for crowd entity resolution is depicted in Figure 1. A first stage produces similarity values for all pairs of records, using machine algorithms. In a second stage, a task selection algorithm determines record comparison tasks to ask humans, gets back the answers, and selects new tasks based on the answers and the machine similarity values. Usually, the task selection loop goes on until we run out of the time or money available for the overall entity resolution task.

A key design choice for human tasks is the number of records per task. There are two options: including just two records in each task (*pairwise* interface, e.g., papers [18, 22, 32, 34, 36, 38]) or including more than two records (*multi-item* interface, e.g., paper [35]). In a *pairwise* task, a human provides a YES/NO (match/non-match) answer, for the two records in the task, typically, via a radio button. In a *multi-item* (or k-item) task, the human assigns a label/color to each record: the human indicates that pairs of records with the same label are matches while the pairs with different labels are non-matches. For instance, in Figure 2, the human

assigns labels/colors in a 6-item task. To allow humans to provide fast answers, all records in the multi-item interface initially have different labels, so by simply pressing the "submit" button, a human indicates that there are no matches in a given task.

Each of the two interfaces has its own advantages and disadvantages. The multi-item interface, allows humans to provide answers for many pairs of records with a few clicks. For example, in Figure 2, a human just assigns the same label to the images on the upper left and upper right corners, to provide answers for all $\binom{6}{2} = 15$ pairs of records; one YES and 14 NOs. On the other hand, the *pairwise* interface is simpler and allows humans to answer questions more accurately since they can focus on two specific records each time. We quantify this trade-off between accuracy and efficiency in terms of cost and time, in Section 2.

A task selection algorithm that attempts to take advantage of both multi-item and pairwise tasks, needs to answer three main questions: ***a)*** how many pairwise and how many multi-item tasks should be used in each task selection step? ***b)*** which records are more appropriate for pairwise tasks and which records for multi-item tasks? ***c)*** which records should be grouped together in each multi-item and pairwise task?

Recent studies in crowd entity resolution use only one of the two interfaces: papers like [18, 22, 32, 34, 36, 38] use the pairwise interface for humans and reason about the most useful questions to ask in each step of the task selection loop. Another paper [35], chooses the multi-item interface, instead. In order to simplify the reasoning for which tasks to use humans for, the authors in [35] assume that humans are always right and, thus, entity resolution can be performed in a single task selection step by just covering with human tasks all "ambiguous" record pairs.

In this paper, we propose the *Waldo* approach (based on the popular 80's book series "Where's Waldo?") that combines the two interfaces to perform crowd entity resolution. Our approach detects "difficult" pairs of records that are resolved more efficiently using pairwise tasks and resolves the rest of the pairs using multi-item tasks (hence, answering question ***b*** above). Moreover, we study the problem of *optimal grouping* of records into pairwise and multi-item tasks given an available budget for each task selection step. The solution to this problem is the set of pairwise and multi-item tasks that maximize the number of "useful" questions for pairs of records and has an overall cost that does not exceed the available budget for that step. Hence, via this solution we answer questions ***a*** and ***c*** above. We formally define the notion of difficult pairs, useful questions and the Optimal Grouping Problem in Sections 5 and 6.

The main contributions of our paper are the following:

1. We present the key observations that drive Waldo, in Section 2. In particular, we observe that there is a trade-off between cost and accuracy for the multi-item and pairwise interfaces. Moreover, this trade-off is different for different pairs of records, as the error rate in human answers varies significantly among pairs, especially when the multi-item interface is used.

2. We present the Waldo architecture, which enables trading off between pairwise and multi-item interfaces, in Section 3.

3. We discuss how human answers are used to resolve a pair of records as a match or non-match, in Section 4.

4. We discuss Waldo's approach in detecting difficult pairs of records, in Section 5.

5. We propose a novel formulation for the Optimal Grouping Problem that accurately captures our key observations (as our experiments verify). We present two approximation algorithms for this problem: a) a constant approximation ratio algorithm that has a $O(n^k)$ time complexity, for k records included in a multi-item task and n records in the dataset and b) a k-approximation algorithm with a $O(n^2 logn)$ time complexity. In addition, we propose a lightweight heuristic that works very well in practice.

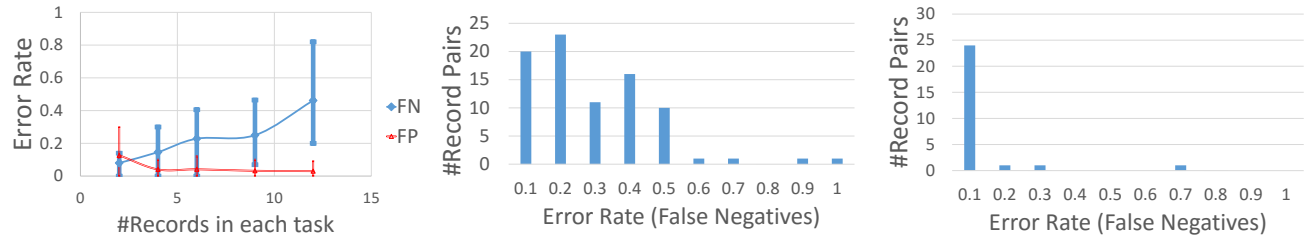
6. We compare Waldo and the different algorithms we propose to baselines and alternatives proposed recently that make use of a single interface, using four real datasets, in Section 7. We find that our algorithms for selecting which records to include in each multi-item task give an up to 17x improvement, in the cost to fully resolve a dataset, over the random approach (that randomly picks which records to place in each multi-item task). In addition, we find that *a)* the improvement of our overall approach compared to using only pairwise tasks can reach up to a factor of 3 in datasets with few "difficult" pairs and *b)* can reach up to a factor of 3 in datasets with many "difficult" pairs, compared to using only multi-item tasks.

2. BASIC INTERFACE COMPARISON

In this section, we compare the pairwise and multi-item interfaces and illustrate the trade-off between accuracy and cost as we increase the number of records we include in a task for humans, via a number of experiments with images of athletes. Our experiments point out that there is not a single *optimal* task size (in terms of number of records): for some pairs of records it is better to have a task involving just the two records, while for other pairs it is more efficient to include them in tasks with additional records.

The human tasks use the interface of Figure 2. Initially, all images in a task have different labels. Hence, the human assigned the task can indicate that all images refer to different athletes, by simply clicking a "submit" button for the task. If the human sees one or more images showing the same athlete, she "assigns" the same label to all those images through the drop-down list of each image. (Another alternative would be to use an empty default label for each image and "force" humans to manually assign labels to all images, before submitting a task.) Our design choice of having different labels for each record as a default, is motivated by the fact that, in most real-world datasets, the number of pairs of records that refer to the same entity is far less than the pairs referring to different entities. Thus, we expect that this design choice significantly reduces the amount of work for humans, compared to forcing humans to provide an explicit label for every single record in each task. (In platforms like Amazon Mturk, the spammers' percentage, which would exploit the "direct" submit function, is negligible when worker requirements are set.)

The dataset [1] used in the experiments of this section, consists of images of athletes (each image is focused on a single athlete) from ten different sports. In the first experiment, we measure the error rate as we increase the number of athlete images included in a task. Each task is generated based on three parameters: the number k of images included



(a) False Negative/False Positive Rates (b) #Pairs per FN rate - 9-item tasks (c) #Pairs per FN rate - 2-item tasks

Figure 3: Error rates and distribution of error rates for different task sizes.

in the task, a number N of pairs of images, and a number M of triplets of images, where each pair or triplet refers to a single entity. For example, for $k = 4$, $N = 1$, and $M = 0$ we would generate a task with one pair of images being a match and the other two images referring to different athletes. We pick 59 such combinations of k, N, and M values (see technical report [7]) and we generate ten tasks for each combination, by randomly choosing images that satisfy N and M, from the same sport. We assign each generated task to ten workers.

In Figure 3(a), the x-axis shows the number of images, k, included in a task and the y-axis shows the error rate: one curve refers to the false negative (FN) rate (i.e., the percentage of answers that give a different label to two records that show the same athlete) and the other curve to the false positive (FP) rate. Along with the average error rate per pair, Figure 3(a) also shows the 10-th and 90-th percentiles; at least 30 pairs refer to each point in the two curves.

One can see that as we increase the number of items in the multi-item interface the FN rate steeply increases. On the other hand, the FP rate slightly decreases from two to four records and stays even for more records in a task.

The explanation for the steep increase in the FN rate is simple: when multiple items are presented at the same time to a crowd worker, it is more likely for her to miss some matches. In addition to the limits in human attention, in the multi-item interface we have to use smaller-size images so that the overall area covered by all images in a task is within space limits. (We discuss in detail the space limits trade-offs for images and relational tuples in the technical report [7]). On the contrary, the FP rate's slight decrease from two to more records per task, is due to two factors: a) there are more pairs of records that are obvious non-matches (than pairs that are obvious matches) and are easy to "spot" even when presented in a task with 12 images. b) having a different label per image as a default, biases humans to "leaving" a different label for most images; at least for pairs of images that are not obvious matches.

In the same experiment, we also observe that in tasks with fewer items, the worker effort per record pair is higher. In Figure 4, the x-axis is the same as before, and on the y-axis we plot the average amount of time needed per record pair answer along with the 10-th and 90-th percentiles. For example, if a worker needs one minute for a 6-item task, the time per record pair is 4 seconds: 60 seconds divided by the $\binom{6}{2} = 15$ pairs of records included in the task. The time per record pair decreases steeply from two to four records and halves from four to six records.

Deciding between which interface to use becomes more complicated as we realize that the cost-accuracy tradeoff, illustrated in Figures 3(a) and 4, is different for different pairs of records. In particular, the FN and FP rates are not the same for all pairs of records: for some pairs of records we have to spend far less multi-item tasks to reach to a final match/non-match answer, compared to other "difficult" pairs where we get "mixed" answers from humans. For example, we get mixed answers when two athletes have similar characteristics (e.g., hair color) and their faces are not clearly visible in the images, like the athletes in the lower left and lower right images, in Figure 2. This effect for multi-item tasks is more pronounced for the FN rate, as the percentiles in Figure 3(a) suggest.

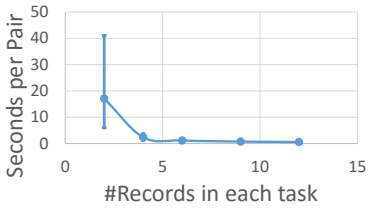


Figure 4: Time per record pair answer.

To illustrate further, we plot in Figure 3(b) the FN rate on the x-axis and the number of record pairs for which we get a specific FN rate on the y-axis, when we include them in 9-item tasks. For example, for an x value of 0.4, there are 16 pairs of images, for which 4 out of 10 answers are wrong. We see that there is an almost uniform distribution on FN rates between 0.0 and 0.5. On the contrary, in the same plot for 2-item tasks, in Figure 3(c), we see that almost all pairs have an FN rate less than 0.1. This observation indicates that combining the two interfaces can be beneficial, since it may be more cost-effective to use the pairwise interface for the pairs with high error rates and for the rest of the pairs to use the multi-item interface.

In particular, the approach we propose in this paper starts by using multi-item tasks to get "fast" answers for a large number of record pairs. Still, for the "difficult" pairs, we get mixed answers from the multi-item tasks. Waldo identifies those pairs (in Section 5 we formally define the term "difficult pair" and we describe how to detect such pairs) and employs pairwise tasks to increase the chances of reaching to a correct final answer, on each such pair. In the next section, we present an overview of our approach and we discuss further how to combine pairwise and multi-item tasks.

3. APPROACH OVERVIEW

Waldo's task selection algorithm is applied in a loop, where, in each iteration, the algorithm selects tasks for humans to answer, based on initial machine answers and human answers from previous iterations.

Initially, machine algorithms generate answers for all record pairs: each answer is a single number expressing the probability of the corresponding pair being a match. For example, consider two customer records, r_1, r_2, with the first and last name of each customer. A logistic regression classifier can use the first and last names in r_1, r_2, and classify the two records as a match with a probability of 0.9.

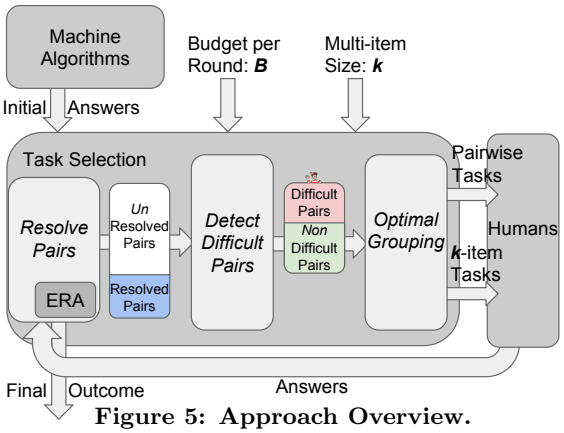


Figure 5: Approach Overview.

Each iteration starts with the "Resolve Pairs" stage (Figure 5). The input to this stage is the initial machine answers along with the answers from multi-item and pairwise tasks, from all previous iterations. The output is a classification of all pairs of records into "Resolved" and "UnResolved" ones. A pair is resolved when the probability of being a match (or a non-match), given all answers for that pair, is greater than a threshold. In the end of this stage, an entity resolution algorithm (ERA) is applied. The ERA can change the label of an UnResolved pair to Resolved. For example, an ERA that applies the transitive relation may infer that an UnResolved pair (a, c) is a match, if pairs (a, b) and (b, c) are resolved as matches. We discuss in detail this stage, in Section 4.

The unresolved pairs and human answers from previous iterations, are the input to the "Detect Difficult Pairs" stage. The output is a classification of the unresolved pairs into "Difficult" and "NonDifficult" pairs. This classification is based on the expected (monetary) cost to resolve a pair via the pairwise interface and via the multi-item interface. Difficult pairs are the ones with a higher expected cost via the multi-item interface, i.e., for a difficult pair it is more cost-effective to use pairwise tasks. We focus on difficult pairs detection, in Section 5.

The "Optimal Grouping" stage selects a set of multi-item and pairwise tasks using the label difficult/non-difficult for each unresolved pair along with the human answers from previous iterations. Difficult pairs are included in pairwise tasks and non-difficult pairs in multi-item tasks (although a few difficult pairs may get unintentionally included in multi-item tasks). All multi-item tasks have a fixed size of k records, which is given as input ($k = 6$ in Figure 2). Setting the same size for all multi-item tasks is sufficient: for each dataset, there is an optimal multi-item task size for "regular" (NonDifficult) pairs, while for Difficult pairs we must rely on pairwise tasks. (The optimal multi-item task size can be found using experiments like the ones in Section 2.) The overall monetary cost of the selected tasks is constrained by a budget B, also given as input. (In practice, a value for B can be computed using techniques like the ones proposed in papers [33] and [23].) Due to the budget constraint, not all of the unresolved (difficult and non-difficult) pairs are included in pairwise and multi-item tasks. Hence, the Optimal Grouping stage evaluates the feasible sets of pairwise and multi-item tasks using a utility function, and selects the set of tasks with the maximum utility. This utility-maximization objective under the budget constraint formulates the Optimal Grouping Problem, discussed in Section 6.

4. RESOLVE PAIRS

Pair resolution is performed via direct resolution (Section 4.1) or via an ERA (Section 4.2). The notation used is as follows:

- m ($\overline{m}$): the event of two records being a (non) match, i.e., (not) referring to the same entity.
- $priors[p]$: the prior probability of being a match, for a pair of records p; given by the respective machine answer.
- $A^{\mathrm{M}}[p] = (x, y)$: the multi-item answers collected for a pair of records p. That is, for all the multi-item tasks that include both of those records, we count the number of times the two records got the same label (YES answers), y, and the number of times they got a different label (NO answers), x. For answers from the pairwise interface on a pair p, we use the notation $A^{\mathrm{P}}[p] = (x, y)$.
- p_{thr}: probability threshold for resolving a pair.

4.1 Direct Resolution

INPUT
A^{M}, A^{P}, p_{thr}, $priors$
OUTPUT
Each pair is labeled as resolved (match/non-match) or unresolved.

Pairs resolution is based on the following rule:

Pair Resolution Rule
A pair p is resolved when $Pr(m|\mathcal{I}) > p_{thr}$ (match) or $Pr(\overline{m}|\mathcal{I}) > p_{thr}$ (non-match), where $\mathcal{I} = \{A^{\mathrm{M}}[p], A^{\mathrm{P}}[p], priors[p]\}$ denotes all evidence for pair p.

That is, a pair is resolved once the probability of being a match (non-match) given the answers from multi-item and pairwise tasks is greater than the probability threshold p_{thr}.

The main technical difficulty in pairs resolution, is the lack of the exact error rate for humans. As we illustrated in Section 2, different pairs have different error rates. Thus, we need to use the answers so far, for a specific pair, to infer the probability of each error rate for that pair and then integrate over all possible error rates, to compute the probability of the pair being a match. This is a significant difference compared to previous work in crowd entity resolution (e.g., papers [32, 36, 38]). However, this is not the main technical contribution of the paper and we decided to move this discussion in Appendix A. Here, we just illustrate pair resolution via a simple example where the error rates for a pair are known:

`EXAMPLE 1.` *Consider a pair p, and assume $p_{thr} = 98\%$ and $priors[p] = 50\%$. We consider a simple scenario here where we only have multi-item task answers. Consider the case where $A^{\mathrm{M}}[p] = (0, 2)$, i.e., there are only two YES answers for pair p. In addition, assume the false negative and positive rates are both 0.1. In this case, $Pr(m|A^{\mathrm{M}}[p]) = \frac{Pr(A^{\mathrm{M}}|m)}{Pr(A^{\mathrm{M}}|m)+Pr(A^{\mathrm{M}}|\overline{m})} = \frac{0.9*0.9}{0.9*0.9+0.1*0.1} \approx 0.9878 > 0.98$ and, hence, pair p is resolved as a match.*

4.2 ERA Resolution

In each iteration, after direct resolution, we apply an ERA that may resolve additional pairs of records. For example, an ERA that applies the transitive relation, will infer that records a and c are a match, when pairs (a, b) and (b, c) are resolved as matches via direct resolution. On the other hand,

if a pair (a, b) has been resolved as a match but a pair (b, c) has been resolved as a non-match, the transitive-relation ERA will infer that (a, c) is also a non-match. Another possibility for a transitive-relation ERA is to apply only the "positive" transitive relation that infers matches, but avoid to infer a non-match (i.e., in the second case above, the ERA would not infer (a, c) as a non-match). Yet another possibility is to take into account all answers between, say, a pair resolved as a match, (a, b), and a second pair also resolved as a match, (c, d), to infer if all 4 records refer to the same entity (e.g., see SCC [32]).

Waldo considers the ERA as a blackbox: any algorithm can be plugged in and resolve additional pairs. Hence, in the main technical problem, discussed in Section 6, we do not include ERA details. The reason for this decision is simple: we wanted to focus on the most general and interesting version of the problem of combining multi-item and pairwise tasks to perform entity resolution.

5. DETECT DIFFICULT PAIRS

Before discussing our definition of difficult pairs let us give the motivation and intuition behind this definition. First, consider a simple heuristic rule for detecting difficult pairs: if after 6 answers for a pair via the multi-item interface, the majority is less than, say, 5, the pair is considered difficult and we should switch to the pairwise interface for that pair. For example, if for a pair p_1 we had 4 NOs and 2 YESes, we would switch to the pairwise interface. Would that be a good decision? It mainly depends on the cost of a pairwise task compared to the cost of a multi-item task. If the cost of a pairwise task was relatively low, switching to the pairwise interface for p_1, would be a good decision. On the other hand, if the pairwise task cost was high, then we may want to wait for more answers for p_1. Hence, such a simple heuristic rule would not suffice: we need a method that takes into account the cost of each type of task.

We extend our notation with the parameters for the (monetary) cost of tasks:

- c^{M}: the (monetary) cost of a single multi-item task.
- c^{P}: the (monetary) cost of a single pairwise task.

The input and output of this stage are as follows:

INPUT
A^{M}, A^{P}, p_{thr}, *priors*, c^{M}, c^{P}
OUTPUT
Each (unresolved) pair is labeled as difficult or non-difficult.

The definition for difficult pairs is given by:

Difficult Pair Rule
A pair is identified as *difficult* when $\mathbb{C}^{\mathrm{M}} > \mathbb{C}^{\mathrm{P}}$.

We denote by $\mathbb{C}^{\mathrm{M}}$ the expected cost to resolve a specific pair using multi-item tasks and by $\mathbb{C}^{\mathrm{P}}$ the expected cost to resolve the same pair using pairwise tasks.

Just like in the pair resolution stage, the main technical difficulty for computing the expected costs, is the lack of the exact human error rate, for each pair of records. Again, we have to infer the probability of each error rate, for a pair, to compute the expected cost to resolve the pair, using multi-item or pairwise tasks. We discuss the details in Appendix B.

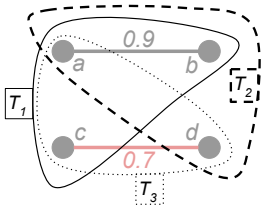


Figure 6: Intuition behind the utility function.

6. OPTIMAL GROUPING PROBLEM

In this section, we define the Optimal Grouping Problem and we propose three algorithms for this problem. We first give the motivation for the problem formulation, in Section 6.1. In Section 6.2, we describe the problem formulation and provide the details for the utility function used in this formulation. In Sections 6.3 and 6.4, we give two simple algorithms for the Optimal Grouping Problem that have strong optimality guarantees. To arrive to these approximation algorithms and prove their optimality guarantees, we show that the utility function is submodular and then we combine key insights that are specific to our problem with tools from submodular optimization. The most challenging proofs to the lemmas and theorems of this section can be found in Appendix D (the rest are included in the technical report [7]). Finally, in Section 6.5, we describe a simple heuristic, which is a greedy adaptation of the constant-factor approximation algorithm of Section 6.3, and works very well in practice (as our experiments indicate).

6.1 Intuition and Motivation

Each set of multi-item and pairwise tasks that can be selected in each iteration, has a "utility". We define utility as the expected number of "useful" questions between the pairs of records included in a set of tasks, where "useful" are the questions that help us identify matches. Before formally defining utility (Section 6.2.1), we give the intuition and motivation via the following example:

`EXAMPLE 2.` *Consider the four records, a, b, c, and d, depicted by corresponding nodes in Figure 6. All pairs except (a, b) and (c, d) have been resolved as non-matches. Given the evidence so far, pair (a, b) is a match with probability* 0.9 *and is considered non-difficult. Pair (c, d) has a* 0.7 *match probability and is considered difficult. Unresolved pairs are depicted by edges, where the weight of each edge is the match probability of the corresponding pair. Next, consider the multi-item tasks T_1, T_2, T_3, of size $k = 3$, depicted by triangular shapes enclosing the records included in the task. The utility of each multi-item and pairwise task depends on the weight of the edges enclosed in the task: the more unresolved pairs within the task and the higher the weights on those pairs, the higher the utility of the task. For instance, the utility of T_1 depends on the weight between a and b, which is the only unresolved pair enclosed in T_1. Moreover, the utilities of tasks T_1 and T_2 are not independent because both tasks share pair (a, b). For instance, the answer to T_1 may be enough to resolve (a, b), and, hence, the answer to T_2 would be redundant. When multiple tasks share the same pairs, the utility of each task decreases based on the probability of the answers on those pairs becoming redundant once other tasks provide the answers. As for the utility of task T_3, it is zero, because the only unresolved pair (c, d) included in T_3, is a difficult pair: the weight of difficult pairs contributes only to the utility of pairwise tasks.*

As Example 2 illustrates, the utility of a set of tasks has three properties: (i) the more matches that can be detected, the higher the utility, (ii) the utility decreases as more pairs are being shared by the tasks, and (iii) difficult pairs contribute only to the utility of pairwise tasks.

The importance of property (ii) is obvious, while property (iii) is motivated by our decision to resolve difficult pairs via pairwise tasks. Regarding property (i), finding matches early accelerates the entity resolution process, since an ERA can infer the relationship between additional pairs of records, using the matches detected already, without asking the crowd (e.g., papers [36] and [34]).

6.2 Problem Definition

The input and output of the Optimal Grouping stage are:

INPUT
$\mathcal{D}$, $\mathcal{E}$, A^{M}, A^{P}, p_{thr}, *priors*, c^{M}, c^{P}, B
OUTPUT
A set of multi-item tasks $\mathcal{T} = \{T_1, \ldots T_t\}$ and a set of pairwise tasks $\mathcal{P} = \{P_1, \ldots P_p\}$, to be issued to humans.

where $\mathcal{D}$ are the difficult pairs and $\mathcal{E}$ the non-difficult ones, from the difficult pair detection stage, and B is the budget for the current task selection round.

The selection of tasks $\mathcal{T}$ and $\mathcal{P}$ is based on the following problem formulation:

PROBLEM 1. **Optimal Grouping Problem**

$$\max_{\mathcal{P},\mathcal{T}} \quad EU(\mathcal{P},\mathcal{T}) \tag{1}$$

$$s.t. \quad |\mathcal{P}| * c^{\mathrm{P}} + |\mathcal{T}| * c^{\mathrm{M}} \leq B \tag{2}$$

Equation 2 states that the sum of costs from pairwise and multi-item tasks should not exceed the budget for the current round. Next, we formally define the utility function EU (Expected Useful questions) of Equation 1.

6.2.1 Utility Function *EU*

Each $T_i \in \mathcal{T}$ is a set of k records and each $P_i \in \mathcal{P}$ is a set of 2 records. Note that both sets $\mathcal{T}$ and $\mathcal{P}$ are actually multisets, since each set T_i and P_i can be included multiple times in $\mathcal{T}$ and $\mathcal{P}$, respectively.

We refer to the pairwise tasks $P_i \in \mathcal{P}$ as *questions*. The multi-item tasks $T_i \in \mathcal{T}$ also imply a multiset of (pairwise) *questions*:

DEFINITION 1. **Question multiset $Q(\mathcal{T})$ of multi-item tasks $\mathcal{T}$**

$$Q(\mathcal{T}) = \{(r_1, r_2) : \exists T_i \in \mathcal{T}, r_1 \in T_i \wedge r_2 \in T_i\}$$

Each question $q \in Q(\mathcal{T})$ is assigned with a sequence number, $q.seq$: if the multiplicity of q is, say, 3 (i.e., the same pair of records appears three times in $Q(\mathcal{T})$), then the first instance of q has $q.seq = 1$, the second instance has $q.seq = 2$, and the third $q.seq = 3$; without assuming a specific order of the different instances. Each question $q \in \mathcal{P}$ is also assigned with a sequence number based on q's multiplicity in $\mathcal{P}$, the same way as in $Q(\mathcal{T})$.

EXAMPLE 3. *Consider again the four records, a, b, c, d and the multi-item tasks $\mathcal{T} = \{T_1, T_2\} = \{\{a, b, c\}, \{a, b, d\}\}$, from Example 2. Then, $Q(\mathcal{T}) = \{(a, b), (b, c), (a, c), (a, b), (b, d), (a, d)\}$. Moreover, there are two instances of question (a, b), denoted by q_1 and q_2. Their sequence numbers are: $q_1.seq = 1$ and $q_2.seq = 2$.*

The objective in Equation 1 states that we want to find the tasks maximizing the expected number of "useful" questions:

DEFINITION 2. **Useful Question:** *A question $q \in \mathcal{P} \cup Q(\mathcal{T})$ is* useful *when: (a) the pair of records in q is a match, (b) the pair in q has* not *been resolved by the answers in previous rounds and the answers to questions for the same pair with a smaller sequence number than $q.seq$, and (c) q is detected as difficult in case $q \in \mathcal{P}$ and q is detected as* non-*difficult in case $q \in Q(\mathcal{T})$.*

Note that the three conditions in Definition 2 match the three properties discussed in Section 6.1. To illustrate further, consider the following example:

EXAMPLE 4. *Consider again the parameters from Example 1 and questions q_1, q_2, for pair (a, b), from Example 3. In addition, assume that (a, b) is identified as* non-*difficult and that we have a single YES multi-item task answer, from previous rounds. For question q_1 to be* useful*, it suffices pair (a, b) to be a match. For question q_2 to be useful, (a, b) must be a match and the answer to q_1 should be NO. (Otherwise, pair (a, b) would have been already resolved by q_1 based on the parameters used in Example 1.)*

In our formulation, we use the indicator random variable U_q, which is 1, if $q \in \mathcal{P} \cup Q(\mathcal{T})$ is useful, and 0 otherwise. The objective function in the Optimal Grouping Problem is denoted by $EU(\mathcal{P}, \mathcal{T})$ (expected number of useful questions):

$$EU(\mathcal{P},\mathcal{T}) = \mathbb{E}\{\sum_{q \in \mathcal{P} \cup Q(\mathcal{T})} U_q\}$$

The next two lemmas simplify the computation of EU.

LEMMA 1.

$$EU(\mathcal{P},\mathcal{T}) = \sum_{q \in \mathcal{P} \cup Q(\mathcal{T})} Pr(U_q = 1)$$

LEMMA 2.

$$\forall q \in Q(\mathcal{T}),\ Pr(U_q = 1) = Pr(m_q | A^{\mathrm{M}}[q]) * \sum_{(x,y) \in W(q)} p_m(x, y)$$

where *a)* m_q is the event of the pair in q being a match, *b)* $p_m(x, y)$ is the probability of reaching point (x, y), i.e., x NO answers and y YES answers, given that the pair is a match, *c)* F is the set of (x, y) points where the pair in q is resolved (see Appendix A for the formal definition of $p_m(x, y)$ and F), and *d)* the set of points $W(q)$ is defined as:

$$W(q) = \{(x, y) : x + y = |A^{\mathrm{M}}[q]| + q.seq - 1 \wedge (x, y) \notin F\} \tag{3}$$

Note that Lemma 2 refers to the case where $q \in Q(\mathcal{T})$. In case $q \in \mathcal{P}$, the same lemma applies with the equivalent definition for the sets of points F and $W(q)$.

Next, we discuss three algorithms for the Optimal Grouping Problem; their formal description is given in Appendix C.

6.3 Greedy Algorithm

In this section, we present an algorithm that uses the submodularity of the EU function; we prove this property in Theorem 1. The most well-known result for submodular functions is the typical greedy algorithm [28] that gives a $(1 - \frac{1}{e})$ approximation guarantee. This approximation guarantee holds only when all items to be selected have the same cost. However, in our problem, pairwise and multi-item tasks have a different cost. The Optimal Grouping Problem falls into the category of increasing submodular function maximization, under a cost constraint, when the items to be selected have different costs. There is a modified greedy al-

gorithm [25] for this category of problems, but it gives a lower approximation ratio of $\frac{1}{2}(1-\frac{1}{e})$.

The greedy algorithm presented here, takes into account the special structure of the Optimal Grouping Problem and gives an approximation guarantee of $(1-\frac{1}{e})$, i.e., twice the ratio of the general case algorithm [25]. We call this algorithm *Greedy*. The key insight of Greedy is that it generates one solution with only multi-item tasks and one solution with only pairwise tasks and, then, it combines the two solutions, to achieve the improved approximation ratio. Hence, Greedy runs in two stages: Stage 1 deals with multi-item tasks and Stage 2 deals with pairwise tasks and the combination of the two solutions.

In Stage 1, Greedy generates a list of multi-item tasks, such that the aggregate cost of all the tasks in the list does not exceed budget B. To generate this list, Greedy appends to the list the multi-item task that increases the overall utility the most, in each step, taking into account the tasks appended in previous steps. To find that task, Greedy examines all possible $\binom{n}{k}$ tasks, in each step, where n is the overall number of records. In Stage 2, Greedy first generates a fixed number of pairwise questions for each difficult pair. Then, it sorts the questions in descending order based on $Pr(U_q = 1)$, i.e., the probability of a question being useful. In the last step of Stage 2, the list of multi-item tasks from Stage 1 is "combined" with the sorted list of pairwise questions, to produce the final outcome: Greedy selects a prefix of the multi-item list and a prefix of the pairwise list, so that the utility of all tasks in the two prefixes is maximized and the total cost is within budget B.

The time complexity of *Greedy* is $O(\lfloor \frac{B}{c^{\mathrm{M}}} \rfloor * n^k)$. Note that finding the best multi-item task to append in each step of Stage 1, is, by itself, an NP-hard problem; by reduction from the Densest k-Subgraph [13]. Note also that the values for both k and n can be quite small. First, the value for k will always be small due to the human attention limitations. Second, there are very effective partitioning methods that can split a dataset into smaller sets of records: usually, machine answers in real datasets are sufficient to identify buckets of records, where there are no matches between buckets. (We discuss such a partitioning method in the technical report [7].) Hence, *Greedy* can be applied on each such bucket separately. Furthermore, examining the $\binom{n}{k}$ tasks in each iteration can be performed in parallel. The theoretical results for Greedy are summarized in the two following theorems:

THEOREM 1. *Consider a sequence of budgets*

$$\langle B_j = j * c^{\mathrm{M}} \mid j \in \mathbb{Z}^+ \rangle$$

For any $i \in [1, |L_{\mathrm{GR}}|]$

$$EU(\emptyset, L_{\mathrm{GR}}[0:i]) \geq (1-\frac{1}{e}) EU(\emptyset, \mathcal{T}^*_{B_i})$$

where L_{GR} *is the outcome of Stage 1 (*$L_{\mathrm{GR}}[0:i]$ *are the first* i *multi-item tasks in the list) and* $\mathcal{T}^*_{B_i}$ *the set of multi-item tasks that maximizes EU for a budget constraint* B_i.

THEOREM 2. *Greedy is a* $(1-\frac{1}{e})$*-approximation algorithm for the Optimal Grouping Problem. That is,*

$$EU(\mathcal{P}_{\mathrm{GR}}, \mathcal{T}_{\mathrm{GR}}) \geq (1-\frac{1}{e}) EU(\mathcal{P}^*, \mathcal{T}^*)$$

where $\mathcal{P}_{\mathrm{GR}}, \mathcal{T}_{\mathrm{GR}}$ *are the tasks returned by Stage 2 and* $\mathcal{P}^*, \mathcal{T}^*$ *is the optimal solution to the Optimal Grouping Problem.*

6.4 Edges Algorithm

Our second approximation algorithm for the Optimal Grouping Problem, named *Edges*, is a more lightweight algorithm compared to Greedy. Let us illustrate Edges key insight, using a simple example. Assume $k = 4$ and a budget B enough for only one multi-item task. Edges will first select the two most useful questions (i.e., highest $Pr(U_q = 1)$), between, say, pair (a, b) and pair (c, d). Then, it will generate a multi-item task, T_E, with records a, b, c, and d. Let us assume that $Pr(U_q = 1) = u_{max}$ for both questions between (a, b) and (c, d). In the worst case, *a)* all pairs between a, b, c, and d are resolved, except pairs (a, b) and (c, d), i.e., $Pr(U_q = 1) = 0$ for all other pairs in T_E and *b)* there is an optimal multi-item task T^* where for all pairs between the records in T^*, there is a question with $Pr(U_q = 1) = u_{max}$. Hence, in the worst case, the overall utility of T_E is $2 * u_{max}$, while the overall utility of T^* is $6 * u_{max}$. That is, the utility of T_E is the $\frac{1}{3} = \frac{1}{k-1}$ of the utility of T^*.

Edges also runs in two stages, where the second stage is identical to the second stage of Greedy. In Stage 1, Edges starts by generating a vector of values for each non-difficult pair p: in position i of the vector, the value stored is $Pr(U_q = 1)$, for a question q on p with $q.seq = i$. Then, all vectors are merged into one list which gets sorted in descending order. In successive steps, Edges generates multi-items tasks until the cost of all tasks generated exceeds budget B. In particular, in each step, Edges removes from the sorted list, the $\lfloor \frac{k}{2} \rfloor$ entries with the highest $Pr(U_q = 1)$, to create a multi-item task with the records referenced by the selected entries (if less than k records are referenced, we randomly add records to the task).

The time complexity of Edges is $O(n^2 logn)$, where n is the overall number of records, while the approximation ratio of Edges is $\frac{1}{k}$. Using Edges key insight and the way multi-item and pairwise tasks are combined in Stage 2 of Edges, we prove the $\frac{1}{k}$ approximation ratio in the following two theorems.

THEOREM 3. *Consider a sequence of budgets*

$$\langle B_j = j * c^{\mathrm{M}} \mid j \in \mathbb{Z}^+ \rangle$$

For any $i \in [1, |L_{\mathrm{E}}|]$

$$EU(\emptyset, L_{\mathrm{E}}[0:i]) \geq \frac{1}{k} EU(\emptyset, \mathcal{T}^*_{B_i})$$

where L_{E} *is the outcome of Edges Stage 1 (*$L_{\mathrm{E}}[0:i]$ *denotes the first* i *multi-item tasks in the list) and* $\mathcal{T}^*_{B_i}$ *the set of multi-item tasks that maximizes EU for a budget constraint* B_i.

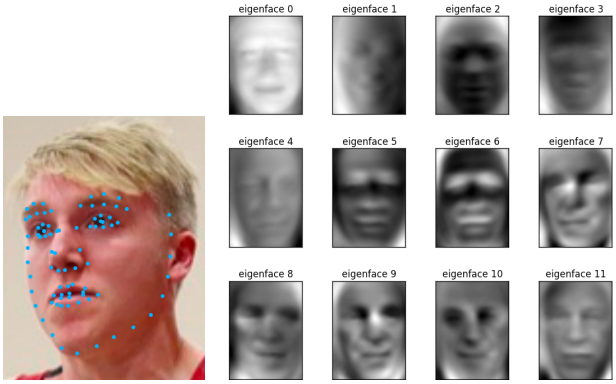
THEOREM 4. *Edges is a* $\frac{1}{k}$*-approximation algorithm for the Optimal Grouping Problem. That is,*

$$EU(\mathcal{P}_{\mathrm{E}}, \mathcal{T}_{\mathrm{E}}) \geq \frac{1}{k} EU(\mathcal{P}^*, \mathcal{T}^*)$$

where $\mathcal{P}_{\mathrm{E}}, \mathcal{T}_{\mathrm{E}}$ *are the tasks returned by Stage 2 and* $\mathcal{P}^*, \mathcal{T}^*$ *is the optimal solution to the Optimal Grouping Problem.*

6.5 2Greedy Algorithm

Our third algorithm, *2Greedy* ("too Greedy"), is the greedy adaptation of *Greedy*: instead of examining all $\binom{n}{k}$ possible k-item tasks, in each step, 2Greedy *greedily* constructs a k-item task. In particular, 2Greedy starts by adding to the task the pair, say, (a, b), referring to the highest $Pr(U_q = 1)$. Then, it finds and adds the record c that increases the overall utility the most, based on pairs (a, c) and (b, c). The same greedy rule is applied to add records, until k records are added in the current task. Stage 2 remains the same as in Greedy.



(a) Landmarks (b) Eigenfaces

Figure 7: Face landmarks and eigenfaces in the *AllSports* dataset.

2Greedy does not come with theoretical guarantees. Nevertheless, one of our key findings in the experimental evaluation is that 2Greedy is very effective in practice, due to the structure of real datasets: in most cases, 2Greedy is just slightly worse than Greedy.

7. EXPERIMENTAL EVALUATION

In our experiments, we compare Waldo with the two general approaches studied in previous research on crowd entity resolution: using only multi-item or pairwise tasks. In fact, for the only-pairwise approach, we use the strategy proposed in [36], adjusted to handle human errors. (Papers [34] and [18] are also based on the same key insight with [36].) We also compare Waldo with three approaches that combine multi-item and pairwise tasks in a "static" way (e.g., using half of the budget for multi-item tasks and the other half for pairwise tasks). In addition, we compare algorithms Greedy, Edges, and 2Greedy, with the naive algorithm that randomly selects which records to include in each multi-item task. We keep the discussions in following sections brief; all details can be found in our technical report [7].

7.1 Datasets, Parameters and ERA

7.1.1 Datasets

In our experiments, we used four datasets: *AllSports* [1], *Products* [5], *Cora* [3], and *Cars* [2]:

AllSports: the dataset consists of 200 images of athletes from ten different sports. Performing entity resolution using only image processing algorithms in this dataset is particularly difficult. We tried Google Photos for this dataset, to find out that only a small portion of the images were clustered into entities (low recall), while for most clusters there were photos referring to many different entities (low precision). The initial machine answers are generated using two image processing methods for face comparison: face landmarks and eigenfaces. The face landmarks are key points detected on a face image (e.g., nose tip). We used Face++ [4] to detect the faces on each image and to detect the landmarks of each face (Figure 7(a) depicts an example of landmark detection by Face++). Using the detected landmarks, we extracted features for each image (e.g., ratio of eye centers' distance to mouth width). The eigenfaces approach generates a low-dimensional representation of face images using Principal Component Analysis (PCA). We used the LFW dataset [24] as the training set for PCA. Figure 7(b) depicts the first 12 eigenfaces produced by PCA. Machine answers are generated using logistic regression, the eigenfaces and landmarks feature representation, and a subset of 67 athlete images from *AllSports* as a training set (the positive/negative examples are the pairs between those 67 images). We discuss the details in our technical report [7].

Name	Description	Default
B	budget in each task selection iteration	\$0.4
c^{M}	cost of a multi-item task	\$0.05
c^{P}	cost of a pairwise task	\$0.02
k	number of records in a multi-item task	6
CPM	confusing pairs % in matches	30%
$CPnM$	confusing pairs % in non-matches	15%
EnC	error rate on non-confusing pairs	0.1
ECk	error rate on confusing pairs (k-item)	0.5
ECp	error rate on confusing pairs (pairwise)	0.2

Table 1: Parameters used in experiments.

Products: the dataset consists of 2000 product records. The machine answers are generated using a histogram approach (see paper [38]), where the similarity value for each pair is the average between *a)* the Jaro distance [40] of the product names and *b)* the ratio of the lowest to highest price.

Cora: the dataset consists of 2000 paper publications with three main fields: the title and authors of the paper, and the conference/journal where the paper was published. The initial machine answers are generated using a histogram approach, where the similarity value for each pair of records is the average Jaro distance of all fields.

Cars: This dataset consists of both images and text. In particular, the dataset consists of 600 records extracted from *autotrader.com*: each record contains the image, make and model of the car in an ad. To make things more challenging, we extracted the model only for 50% of the records. Two records refer to the same entity when they show a car of the same make, model, and generation, e.g., two 6th generation corvettes are considered the same entity even if one is a coupe and the other a convertible. The initial machine answers are generated using the text information, i.e., the make and model fields (e.g., for 2 cars of the same make and model, we compute the probability of the cars being the same generation and, hence, the same entity, using statistics).

7.1.2 Parameters

Table 1 summarizes the parameters of the experimental setting. In the experiments presented here, in each iteration, we spend a budget B of 40 cents. We also tried other values for budget B, but we did not notice any significant differences regarding our key findings. In addition, the cost c^{M} of any multi-item task (independently of its size k) is 5 cents and the cost c^{P} of a pairwise task is 2 cents. (By keeping the same cost c^{M} for any size k, we examine different values for the normalized cost-per-pair, i.e., cost $c^{\mathrm{M}}/\binom{k}{2}$.)

In our experiments, we used both emulated human answers (on real datasets) and real answers from mTurk (again, on real data). With answer emulation we have more control over the experimental setting and we are able to explore and draw conclusions about the behavior of the different approaches in different scenarios. Real answers let us verify that such conclusions also hold in a more realistic scenario. (Note that real answers' experiments have a significant cost, so it would not be practical to run every scenario using real answers. For example, in our case, we had to issue 20000 multi-item tasks; see below.)

In human answer emulation, there are five parameters controlling the emulation. Pairs of records are split into "con-

fusing" and "non-confusing". Confusing pairs are those with a high error rate, while non-confusing have a low error rate. (Depending also on the cost parameters and size k, confusing pairs usually end up being detected as difficult.) The error rate for non-confusing pairs is the same for pairwise and multi-item tasks and is given by the Error-non-Confusing (*EnC*) parameter. (For example, for $EnC = 0.1$ we synthetically generate each non-confusing-pair answer with a 0.1 probability the generated answer to be wrong.) The error rate for confusing pairs is given by the *ECk* parameter, in case of k-item tasks, and the *ECp* parameter, in case of pairwise tasks. The last two parameters are the Confusing-Percent-Matches (*CPM*) parameter, which gives the percentage of confusing pairs for matches, and the Confusing-Percent-non-Matches (*CPnM*) parameter, which gives the percentage of confusing pairs for non-matches. (For example, a $CPM = 30\%$ means that 30% of the pairs that are matches, are confusing.)

Note that the default values for the answer emulation parameters are consistent with the results in Section 2. For example, for the default *CPM*, *EnC*, *ECk* and k values of Table 1, the expected false negative (FN) rate for 6-item tasks is $0.5 * 30\% + 0.1 * 70\% = 0.22$. As Figure 3(a) shows, the FN rate for $k = 6$ is exactly 0.22.

In our real human answers' experiments, we focused on the *AllSports* and *Cars* datasets. We ran those experiments in two modes: *cached* and *live*. The experiments presented here are all run in *live* mode: all tasks selected by an algorithm, in one round, are directly sent to mTurk and the algorithm resumes once all tasks are completed. (See the technical report for the *cached* mode experiments for *AllSports*; the cached answers are included in the dataset material [1].)

7.1.3 Entity Resolution Algorithm

In the experiments presented here, we used a simple transitive relation algorithm (the same one with papers [34, 36]). This algorithm applies the transitive relation both for matches and non-matches. For example, if pair (a, b) is a match and pair (b, c) is also a match, the algorithm infers that (a, c) is a match as well. On the other hand, if (a, b) is a match and (b, c) is a non-match, the algorithm infers that (a, c) is a non-match.

7.2 Experiments

In the first experiment presented here, we compare Greedy, Edges, 2Greedy, and *Random*. The Random approach randomly selects the records to include in each multi-item task and uses the same second stage with the other three, to select a number of pairwise tasks for difficult pairs. The experiment is run on the *AllSports* dataset using answer emulation, with the default parameter values besides *CPM*, which is set to 10%, and *CPnM*, which is set to 5%. In this first experiment, we use lower values than the default for *CPM* and *CPnM* (i.e., assume lower percentages of confusing pairs) because the main focus is on the multi-item task selection (which is even more critical when the percentage of confusing pairs is low).

The results are depicted in Figure 8(a): the y-axis shows the F1 score after each task selection iteration (x-axis). F1 is given by $\frac{2*p*r}{p+r}$, where p is the precision and r is the recall.

Greedy is the most effective algorithm and 2Greedy is very slightly worse. The cost of reaching an F1 score of 0.9 via Greedy and 2Greedy is half the cost of reaching an F1 of 0.9 via Edges: Greedy and 2Greedy need around 100 iterations to reach 0.9 while Edges needs 200 iterations. Even more impressive is the improvement over the naive Random algorithm: Random needs around 1700 iterations to reach 0.9 (not shown in Figure 8(a)), hence, there is a 17x improvement when using Greedy or 2Greedy and a 8x improvement when using Edges.

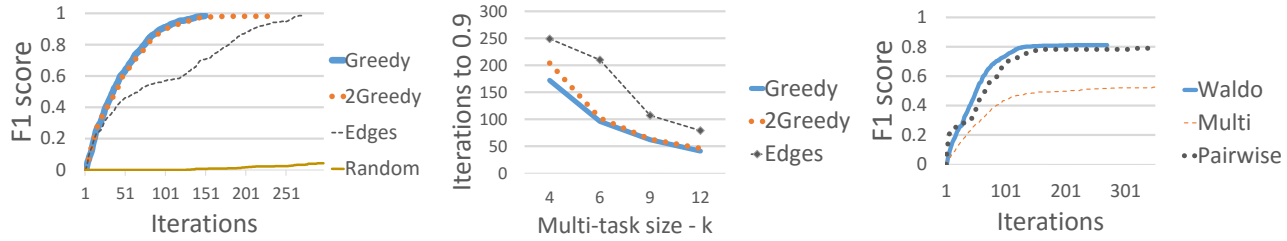
We also compared Greedy, Edges, and 2Greedy, for different values of k. The results are depicted in Figure 8(b). The y-axis shows the number of iterations needed to reach an F1 score of 0.9, for each k value (x-axis). There are two interesting findings from the experiments of Figure 8(b). First, Edges needs almost twice as many iterations as Greedy and 2Greedy do, for k greater or equal to 6. This points out that the constant approximation factor of Greedy makes a 2x difference in practice, compared to the k approximation factor of Edges. Second, 2Greedy is as effective as Greedy, except for one case: for $k = 4$, 2Greedy needs around 20 percent more iterations than *Greedy*. Incidentally, in this case where k is small, the computational cost of Greedy is not prohibitive and the choice of Greedy over 2Greedy may be preferable, when the number of records is not very large.

In general, in all four datasets used in our experiments, the effectiveness (i.e., obtaining a high F1 score with fewer tasks) of Greedy and 2Greedy is very similar in most cases. Nevertheless, there are cases where the structure of the dataset causes Greedy to be much more effective than 2Greedy. To illustrate, consider a very simple synthetic dataset that: a) consists of x entities, b) exactly $y\%$ of the entities have exactly two records referring to them, while the rest have a single record, and c) the prior probability of a match for two records that refer to the same entity is picked uniformly at random from $[1.0-\epsilon, 1.0]$ and picked uniformly at random from $[0.0, \epsilon]$ if the two records refer to different entities. It is not hard to see that as y and ϵ approach zero, the utility of the tasks created by Greedy will be $\frac{k}{2}$ times higher than the utility of the tasks created by 2Greedy.

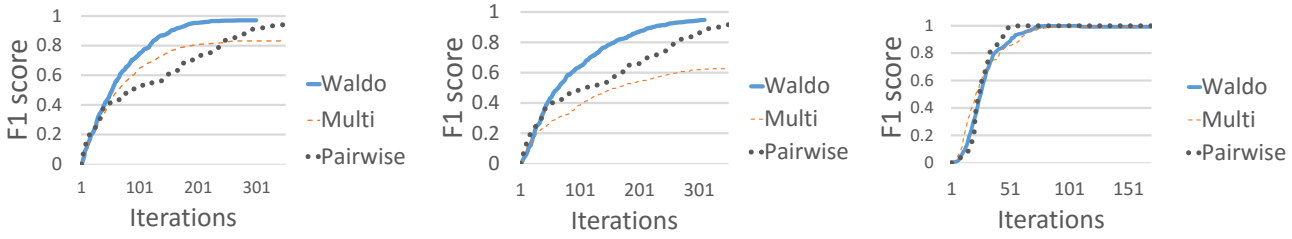
To confirm the difference between Greedy and 2Greedy in practice, we created a synthetic dataset with $x = 1000$, $y = 10\%$, and $\epsilon = 0.01$, and we ran experiments with the same parameters as in Figure 8(b). For $k = 4$, Greedy can reach an F1 score of 0.9 in 7 iterations while 2Greedy reaches 0.9 in 11 iterations, for $k = 6$, Greedy reaches 0.9 in 4 iterations while 2Greedy in 9 iterations, and for $k = 8$, Greedy reaches 0.9 in 3 iterations while 2Greedy in 8 iterations. Note that for the structure of this simple dataset, Edges shows the same effectiveness as Greedy.

Using a similar rationale, one can construct datasets where Greedy is substantially more effective than both 2Greedy and Edges. In practice, most real datasets (including the four datasets used here) do *not* have a problematic structure for 2Greedy. Hence, we use 2Greedy in the experiments that follow; taking also into account the lower complexity of 2Greedy.

Regarding the execution time for the experiments of Figure 8(b), each invocation of Greedy requires 0.23 seconds for $k = 4$, 2.98 seconds for $k = 6$, 9.62 seconds for $k = 9$, and 26.33 seconds for $k = 12$, on a 1.8 GHz Intel Core i5 processor. On the other hand, 2Greedy requires from 10 ms for $k = 4$ to 20 ms for $k = 12$ and Edges requires from 5 ms for $k = 4$ to 12 ms for $k = 12$. Clearly, Greedy requires an execution time that is orders of magnitude higher than the one for 2Greedy and Edges, however, the time for Greedy is



(a) $k = 6$ (b) $k \in \{4, 6, 9, 12\}$ (c) *Products* - $CPM = 30\%$



(d) *AllSports* - $CPM = 30\%$ (e) *AllSports* - $CPM = 50\%$ (f) *Cora* - $CPM = 50\%$

Figure 8: Comparison of (i) Greedy, Edges, and 2Greedy, and (ii) Waldo, Multi, and Pairwise.

still an order of magnitude lower than the time needed by humans to complete the tasks in each round. (As discussed in Section 6.3, in large datasets, partitioning can keep the number of records, on which Greedy is applied, sufficiently small, while there are very few real datasets where large multi-item tasks (e.g., $k > 9$) can be used; due to the limits in human cognitive abilities.)

Next, we compare Waldo (task selection is performed by 2Greedy) with using only multi-item tasks (Multi) or using only pairwise tasks (Pairwise). (That is, in Multi we consider all pairs as non-difficult while in Pairwise we consider all pairs as difficult.) The experiment is run on the *AllSports* dataset with the default parameter values.

The results are depicted in Figure 8(d). There is a substantial improvement when using Waldo compared to Multi and Pairwise: a 1.5x improvement over Multi and a 2x improvement over Pairwise, for the cost of reaching an F1 of 0.8. In addition, Waldo converges to a much higher F1 score compared to Multi: Waldo reaches to an F1 close to 1.0 after 200 iterations, while Multi converges to an F1 just above 0.8 as the confusing pairs cannot be resolved using multi-item tasks. Of course, the size of the gap between Waldo and Pairwise or Waldo and Multi can grow if one of the interfaces becomes less efficient (higher expenses or more errors) compared to the other. Waldo can dynamically adapt to each scenario, and use the interface that is most efficient in each case, for each pair.

Next we examine an extreme case with a large percentage of confusing pairs: we increase *CPM* to 50% and *CPnM* to 25%. The results are depicted in Figure 8(e). Since the confusing pairs mostly affect the multi-item tasks (*ECk* is 0.5 while *ECp* is 0.2), Multi becomes much worse than Pairwise and cannot reach an F1 higher than 0.65. Nevertheless, Waldo remains robust to the large percentage of confusing pairs and shows an important 1.6x improvement over Pairwise for the cost of reaching an F1 of 0.8. In addition, the improvement of Waldo over Multi increases greatly: the cost for reaching an $F1$ of 0.6 with Multi is now three times more than the cost with Waldo.

The effect of the confusing pairs also depends on the dataset and, in particular, the number/distribution of records per entity, as we illustrate in the next two experiments.

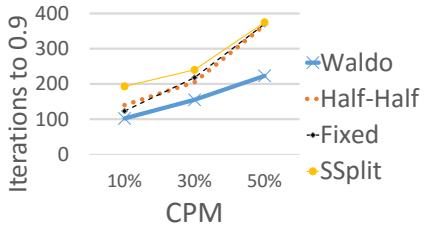


Figure 9: Waldo vs Half-Half and Fixed.

In the experiment of Figure 8(c) we use the *Products* dataset and the default parameter values. That is, *CPM* is set to 30% and *CPnM* to 15%, just like in Figure 8(d). Here, to be able to compare the results from the two datasets, we use only a subset of 200 records from *Products*. (In general, the same findings hold when the full dataset is used - in fact, in some scenarios the gains from applying Waldo increase when there are more records in the dataset.) Although the parameters are the same with the ones in Figure 8(d), we observe that the plot in Figure 8(c) looks more like the plot of Figure 8(e) than the plot of Figure 8(d): the Pairwise curve is close to the Waldo curve and Multi converges to a low F1 score, below 0.6. This can be explained by the number of records per entity: in the *Products* dataset there are exactly two records that refer to the same entity/product for the vast majority of entities, while in the *AllSports* dataset for the vast majority of entities there are more than two records. Having more records per entity can be beneficial: the ERA can infer matches through transitivity for confusing pairs without asking humans any questions for such pairs.

In the experiment of Figure 8(f), we use the *Cora* dataset and try the same stress test with Figure 8(e), by increasing *CPM* to 50% and *CPnM* to 25%. (Again, we used a subset of 200 records here, to compare with the experiment in Figure 8(e) - same findings hold when the full dataset is used.) Quite surprisingly, the results are very different from the ones in Figure 8(e) and all three approaches show almost the same, high, effectiveness. The reason is the power-law distribution for the number of records per entity in the *Cora* dataset, that causes the vast majority of matches to refer to a very small number of entities.

As the experiments of Figures 8(d) to 8(f) indicate, there is a clear advantage of combining multi-item with pairwise

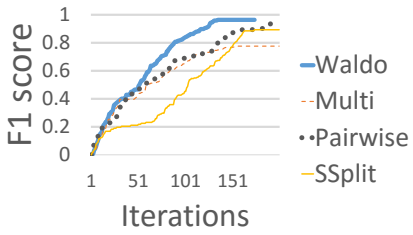


Figure 10: *AllSports* on mTurk answers.

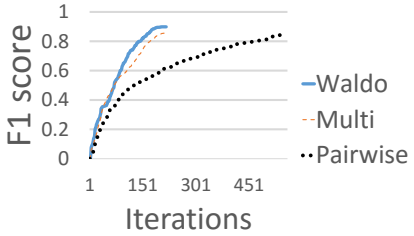


Figure 11: *Cars* on mTurk answers.

tasks, compared to using just one of the two types of tasks. In the next experiment, we investigate further to see if a "static" approach for combining multi-item and pairwise tasks can be just as effective as the sophisticated task selection method of Waldo. In fact, we compare Waldo to three static approaches: a) Half-Half uses half of the budget B in each iteration for multi-item tasks and the rest for pairwise tasks, b) Fixed switches to the pairwise interface for a pair of records only when a fixed limit of questions is reached for that pair, and c) SSplit statically selects a splitting point X and assigns each pair p to multi-item tasks if $priors[p] > X$ and pairwise tasks otherwise. Note also that SSplit groups pairs in multi-item tasks assuming the utility of each pair p is simply $priors[p]$ and first resolves the pairs in multi-item tasks, via crowd answers or transitivity, before resolving the pairs in pairwise tasks. We run the experiment on the *AllSports* dataset and we vary CPM from 10% to 50%. Percentage $CPnM$ is always set to $CPM/2$. The splitting point X for SSplit is based on CPM (e.g., if $CPM = 30\%$, the splitting point is on the 30-th percentile of $priors$).

The results are depicted in Figure 9, where the x-axis shows the CPM and the y-axis the number of iterations needed to reach an F1 score of 0.9. As the results indicate, Waldo gives a substantial improvement over the three static methods, which increases as CPM increases: Half-Half and Fixed require 30% more iterations to reach 0.9 for $CPM = 10\%$, 35% more iterations to reach 0.9 for $CPM = 30\%$, and 65% more iterations to reach 0.9 for $CPM = 50\%$. SSplit is the least effective method as it solely relies on $priors$ to decide which interface to use and how to group pairs in multi-item tasks, while other methods make better decisions using answers from the crowd.

As our experiments so far show, Waldo dynamically uses the most efficient interface and shows significant cost savings in many cases. Furthermore, Waldo never does worse than an other alternative (only in Figure 8(f) did we see Waldo slightly underperform). Regarding our 3 algorithms Greedy, 2Greedy, and Edges, they give a vast 17x improvement (8x in case of Edges) over the naive Random approach. In addition, the effectiveness of Greedy and Edges is aligned with their theoretical guarantees, while 2Greedy, which does not have theoretical guarantees, proves to be almost as effective as Greedy, in practice.

7.2.1 mTurk Answers

We now switch from answer emulation to real answers, to examine if the improvement of Waldo over Multi, Pairwise, and SSplit holds in an actual mTurk scenario as well. The first experiment is run on the *AllSports* dataset, for $k = 6$. Figure 10 depicts the results. Waldo is considerably better than Pairwise: Pairwise needs 50% more iterations to reach an F1 of 0.8 and, hence, it requires a monetary cost 50% higher than Waldo. Multi follows Pairwise closely up to a point, but converges to a much lower F1 (lower than 0.8), while SSplit is substantially worse than Multi in the first 100 iterations, yet, it manages to reach a higher F1, eventuallly.

Let us compare the results of Figure 10 with the results from the corresponding emulation experiment of Figure 8(d). (The results in Section 2 are on the *AllSports* dataset and, as discussed in Section 7.1.2, the default parameter values used in Figure 8(d) are aligned with the error rates observed in Section 2.) When comparing the results of Figure 10 with the results of Figure 8(d), it is interesting to observe that Waldo and Pairwise show an improved effectiveness in the real-answers setting, while Multi does not; in fact, Multi converges to a slightly lower F1 score. The reason for these differences in the real-answers setting, is the presence of some "problematic" images that are likely to form "confusing" pairs with many other images, in practice. (On the contrary, in the synthetic-answers setting confusing pairs are selected uniformly at random.) For example, consider the image of a soccer player with her head down so that her face is not entirely visible, but with her player number being visible. This image will form a confusing pair with all other images of the same player where the player number is not visible. Waldo intelligently chooses to use pairwise tasks that are more efficient for such confusing pairs and ends up being more effective in the real-answers setting.

The second mTurk answers' experiment is run on the 600 records of the *Cars* dataset. This time we increase the size of multi-item tasks, k, to 9; while keeping the default values for the tasks costs' c^{M} and c^{P}. As the results in Figure 11 show, there is a 3x improvement of Waldo over Pairwise to reach an F1 of 0.8; and a small improvement over Multi.

In this experiment, we observe a new interesting fact that was not evident in all the previous experiments presented here. While we use a large number of records in each multi-item task ($k = 9$), we observe that the accuracy of workers in multi-item tasks and pairwise tasks is quite close (this is why Waldo and Multi are way more effective than Pairwise). Workers that are car experts, can easily detect the cars referring to the same model and generation, even when each multi-item task contains 9 (small) images of cars. (We used the mTurk masters' requirement to ensure that only "car experts" participated in the experiment.) On the other hand, in the *AllSports* dataset, it is very difficult to detect which records refer to the same entities/persons, when a large number of (small) images are included in each multi-item task, no matter how "expert" a worker is in spotting out the same person in images. Clearly, the visual cues in the case of car models (e.g., shape of headlights) are much "stronger" for the experts, compared to the visual cues in the case of athletes. The takeaway here is that the type of entities in a dataset and the ability of humans to detect such entities when presented with a large number of records, drastically affects each approach. Waldo automatically detects when humans can be very accurate when presented with a large number of records, correctly relies heavily on

multi-item tasks in such a case, and issues pairwise tasks only for the few pairs where multiple errors occur.

7.2.2 Key Findings

(1) *2Greedy* does not provide any theoretical guarantees and can fail arbitrarily bad in theory. Nevertheless, in real datasets, *2Greedy* is as effective as *Greedy* in most cases, while being more lightweight.
(2) In cases where there are few confusing pairs in a dataset, using multi-item tasks is more effective than using pairwise tasks. On the contrary, when there are many confusing pairs using pairwise tasks is a better option. Waldo proves to be substantially better than using only multi-item or pairwise tasks in all cases.
(3) Naively combining multi-item and pairwise tasks (like Half-Half, Fixed, or SSplit do), is not sufficient to reach Waldo's effectiveness: Half-Half and Fixed require 30 to 65 percent more iterations than Waldo to reach an F1 of 0.9, in the experiments discussed here.
(4) Our experiments prove that our problem formulation accurately captures the right objectives, as the following findings indicate: *a)* the constant factor approximation algorithm Greedy is 17 times better than Random, while the $\frac{1}{k}$-factor approximation algorithm Edges is 8 times better. *b)* when compared to Multi and Pairwise, Waldo is the most effective approach in all cases and it is considerable better than the second best in many cases. *c)* Waldo's budget allocation (driven by the objective function of the Optimal Grouping formulation) is substantially more effective than the static allocations of Half-Half, Fixed and SSplit.

8. RELATED WORK

Over the last few years, crowdsourcing has become an important tool for data cleansing and integration. Many recent studies have proposed systems and frameworks [10, 11, 12, 19, 15, 37, 42, 44, 31] that combine algorithms and humans to perform data cleansing and integration.

One of the most critical tasks in data integration is entity resolution [9, 14, 17, 21, 29, 30, 40, 39]. In crowd entity resolution, most studies assume the use of the pairwise interface [8, 16, 18, 22, 34, 32, 38, 36, 43]. Very related to our paper are the studies in papers [18, 34, 36], where the approaches proposed use only pairwise tasks and the key insight is that finding matches first (before non-matches) accelerates the entity resolution process. Our Optimal Grouping problem formulation also uses this insight. The two important differences of our work with the studies [18, 34, 36] are: *a)* we do *not* assume that humans always give a correct answer and *b)* we combine pairwise with multi-item tasks. In fact, one of the fundamental observations driving our approach, is that the error rates of humans are different for different pairs of records: using the multi-item tasks for low error-rate pairs and using the pairwise tasks for high error-rate pairs is one of the key insights of Waldo.

Approaches using only pairwise tasks that take into account human errors have also been proposed [22, 32, 43]. One of the main objectives in these studies is the resolution of entities taking into account all (possibly erroneous) human answers. On the contrary, in our study, we combine multi-item and pairwise tasks and we focus on the selection of the set of tasks that best utilizes the available budget.

The use of the multi-item interface has also been studied in crowd entity resolution papers [27, 35]. Paper [27] compares the use of the multi-item interface to batching multiple pairwise YES/NO comparisons in the same task for humans. Another paper [35] that uses only the multi-item interface, assumes humans do *not* make mistakes, and operates in a single step by selecting a set of tasks that cover all ambiguous (based on machine evidence) pairs. On the other hand, we observe that although the multi-item interface is more cost-effective for many pairs of records, there are also pairs where the pairwise interface is more effective, and we study the optimal way to combine the two interfaces.

Finally, the multi-item interface has also been proposed in [20] for the slightly different problem of crowd-clustering [20, 26, 41], where the items to be placed in the same cluster do *not* have to refer to the same entity, but just be "similar" under human-decidable criteria.

9. CONCLUSION

We studied an approach of combining pairwise and multi-item tasks for Crowd Entity Resolution. Our key insight is that the available resources can be best utilized when certain "difficult" pairs of records are included in pairwise tasks while the rest of the pairs are resolved via multi-item tasks. We provided a formal definition for the problem of selecting the optimal set of multi-item and pairwise tasks within a budget constraint and we derived two algorithms with strong theoretical guarantees and one lightweight heuristic which is very effective in practice.

In our experimental evaluation, our Waldo approach proved to be the most cost-effective alternative in all scenarios examined. When compared to using only pairwise tasks, Waldo gave a substantial speedup in resolution in scenarios where there are few difficult pairs and when compared to using only multi-item tasks, Waldo showed a substantial improvement in scenarios where there are many difficult pairs. Our experiments have also shown that an important factor critically affecting performance, is the way the difficult pairs are distributed across a dataset. In practice, the difficulty of each pair of records for a human and how difficult pairs are distributed in a dataset cannot be known in advance. Waldo is able to timely and accurately estimate the difficulty of each pair and best utilize the resources through a sophisticated task selection.

10. REFERENCES

[1] Allsports. stanford.edu/~verroios/datasets/allsports-extended.zip.
[2] Cars. stanford.edu/~verroios/datasets/cars.zip.
[3] Cora dataset. people.cs.umass.edu/~mccallum/data/cora-refs.tar.gz.
[4] Face++. http://www.faceplusplus.com/.
[5] Products. http://dbs.uni-leipzig.de/file/Abt-Buy.zip.
[6] Tamr. http://www.tamr.com/.
[7] Waldo tech report, http://ilpubs.stanford.edu:8090/1137/.
[8] A. Abboura, S. Sahrl, M. Ouziri, and S. Benbernou. Crowdmd: Crowdsourcing-based approach for deduplication. In *Big Data (Big Data), 2015 IEEE International Conference on*, 2015.
[9] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014.
[10] A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD*, 2010.
[11] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi. Active sampling for entity matching. In *KDD*, 2012.

[12] M. Bergman, T. Milo, S. Novgorodov, and W.-C. Tan. Query-oriented data cleaning with oracles. In *SIGMOD*, 2015.
[13] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, 2010.
[14] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data*, 2007.
[15] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, 2015.
[16] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, 2013.
[17] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.
[18] D. Firmani, B. Saha, and D. Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.
[19] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.
[20] R. Gomes, P. Welinder, A. Krause, and P. Perona. Crowdclustering. In *NIPS*, 2011.
[21] A. Gruenheid, X. L. Dong, and D. Srivastava. Incremental record linkage. *PVLDB*, 7(9):697–708, 2014.
[22] A. Gruenheid, B. Nushi, T. Kraska, W. Gatterbauer, and D. Kossmann. Fault-tolerant entity resolution with the crowd. *CoRR*, 2015.
[23] D. Haas, J. Wang, E. Wu, and M. J. Franklin. Clamshell: Speeding up crowds for low-latency data labeling. *PVLDB*, 9(4):372–383, 2015.
[24] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.
[25] A. Krause and C. Guestrin. A note on the budgeted maximization of submodular functions. *Technical report, CMU, CMU-CALD-05-103, 2005.*
[26] J. Lee, H. Cho, J.-W. Park, Y.-R. Cha, S.-W. Hwang, Z. Nie, and J.-R. Wen. Hybrid entity clustering using crowds and data. *The VLDB Journal*, 22(5):711–726, 2013.
[27] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *PVLDB*, 2011.
[28] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 1978.
[29] T. Papenbrock, A. Heise, and F. Naumann. Progressive duplicate detection. *Knowledge and Data Engineering, IEEE Transactions on*, 27(5):1316–1329, 2015.
[30] V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4(4):208–218, 2011.
[31] Y. Tong, C. Cao, C. Zhang, Y. Li, and L. Chen. Crowdcleaner: Data cleaning for multi-version data on the web via crowdsourcing. In *ICDE*, 2014.
[32] V. Verroios and H. Garcia-Molina. Entity resolution with crowd errors. In *ICDE*, 2015.
[33] V. Verroios, P. Lofgren, and H. Garcia-Molina. tdp: An optimal-latency budget allocation strategy for crowdsourced MAXIMUM operations. In *SIGMOD*, 2015.
[34] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. In *VLDB*, 2014.
[35] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. In *VLDB*, 2012.
[36] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.
[37] S. Wang, X. Xiao, and C.-H. Lee. Crowd-based deduplication: An adaptive approach. In *SIGMOD*, 2015.
[38] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. In *VLDB*, 2013.
[39] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *IEEE Trans. on Knowl. and Data Eng.*, 25(5):1111–1124, 2013.
[40] W. Winkler. Overview of record linkage and current research directions. *Technical report, Statistical Research Division, U.S. Bureau of the Census, Washington, DC*, 2006.
[41] J. Yi, R. Jin, S. Jain, T. Yang, and A. K. Jain. Semi-crowdsourced clustering: Generalizing crowd labeling by robust distance metric learning. In *NIPS*, 2012.
[42] C. Zhang, L. Chen, Y. Tong, and Z. Liu. Cleaning uncertain data with a noisy crowd. In *ICDE*, 2015.
[43] C. Zhang, R. Meng, L. Chen, and F. Zhu. Crowdlink: An error-tolerant model for linking complex records. In *Workshop on Exploratory Search in Databases and the Web (ExploreDB)*, 2015.
[44] C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, 2013.

APPENDIX

A. PROBABILITIES' COMPUTATION

In this appendix, we discuss how to compute the probabilities involved in pair resolution (Section 4) and difficult pair detection (Section 5). In particular, we describe how to compute the probability of a pair being a (non) match given the answers for that pair, and the probability of reaching x NO and y YES answers for a specific pair of records.

Each pair of records has a different false negative/positive rate, e_m or $e_{\overline{m}}$. We start by assuming that we know the exact error rate for each pair of records and we then generalize into the case where the error rates need to be estimated. Moreover, we first focus in the case where we only have multi-item answers for the pair and we then discuss what changes when we have multi and pairwise answers.

When we are aware of the error rates e_m and $e_{\overline{m}}$ for a given pair p of records, we can compute the probability of the two records referring to the same entity, given the multi-item answers for that pair:

$$Pr(m|A^{\mathrm{M}}, e_m, e_{\overline{m}}) = \frac{Pr(A^{\mathrm{M}}|m, e_m)Pr(m)}{Pr(A^{\mathrm{M}}|m, e_m)Pr(m) + Pr(A^{\mathrm{M}}|\overline{m}, e_{\overline{m}})Pr(\overline{m})}$$

(To avoid clutter, we will be using A^{M}, A^{P} and $Pr(m)$, instead of $A^{\mathrm{M}}[p]$, $A^{\mathrm{P}}[p]$, and $priors[p]$.)

For example, if $A^{\mathrm{M}} = (1, 1)$ and $e_m = 0.1$, then $Pr(A^{\mathrm{M}}|m, e_m) = 0.9 * 0.1 = 0.09$. The probability $Pr(\overline{m}|A^{\mathrm{M}}, e_m, e_{\overline{m}})$ is defined equivalently.

Let us now define the points (x, y), where after x NO and y YES multi-item answers, we resolve the pair:

$$F_d = \{(x, y) : Pr(m|A^{\mathrm{M}} \cup (x, y)) > p_{thr} \vee Pr(\overline{m}|A^{\mathrm{M}} \cup (x, y)) > p_{thr}\}$$

For some pairs of records it is possible that they will never get resolved, no matter how many questions are asked for them: there are pairs that, even humans, cannot recognize if they are a match or not. For such pairs we will keep getting mixed answers, even when using the pairwise interface, and the probability threshold p_{thr}, required to resolve a pair, will never be reached. This is why Waldo uses two upper thresholds f^{M} and f^{P}, for the maximum number of times a pair can be included in a multi-item and a pairwise task:

- f^{M}: the maximum number of times we can include a specific pair of records in a k-item task (a pair can still

$$p_m(x,y) = \begin{cases} p_m(x,y-1)(1-e_m) + p_m(x-1,y)e_m & : (x,y-1) \notin F \wedge (x-1,y) \notin F \\ p_m(x,y-1)(1-e_m) & : (x,y-1) \notin F \wedge (x-1,y) \in F \\ p_m(x-1,y)e_m & : (x,y-1) \in F \wedge (x-1,y) \notin F \\ 0 & : (x,y-1) \in F \wedge (x-1,y) \in F \\ 1 & : x=0 \wedge y=0 \end{cases}$$

$$p_{\overline{m}}(x,y) = \begin{cases} p_{\overline{m}}(x-1,y)(1-e_{\overline{m}}) + p_{\overline{m}}(x,y-1)e_{\overline{m}} & : (x,y-1) \notin F \wedge (x-1,y) \notin F \\ p_{\overline{m}}(x,y-1)e_{\overline{m}} & : (x,y-1) \notin F \wedge (x-1,y) \in F \\ p_{\overline{m}}(x-1,y)(1-e_{\overline{m}}) & : (x,y-1) \in F \wedge (x-1,y) \notin F \\ 0 & : (x,y-1) \in F \wedge (x-1,y) \in F \\ 1 & : x=0 \wedge y=0 \end{cases}$$

Figure 12: Recursive computation of $p_m(x,y)$ and $p_{\overline{m}}(x,y)$.

get unintentionally included in a k-item task even if the f^{M} limit has been reached).

- f^{P}: the maximum number of times we can include a specific pair of records in a pairwise task.

Next, we define the set of points (x,y) where we stop asking multi-item questions for a pair of records, because we reached threshold f^{M}:

$$F_o = \{(x,y) : |A^{\mathrm{M}}| + x + y = f^{\mathrm{M}} \wedge Pr(m|A^{\mathrm{M}} \cup (x,y)) < p_{thr} \wedge Pr(\overline{m}|A^{\mathrm{M}} \cup (x,y)) < p_{thr}\}$$

We denote by F the union of F_d and F_o.

To avoid clutter in the definitions of F_d and F_o, we used $Pr(m|A^{\mathrm{M}} \cup (x,y))$ instead of $Pr(m|A^{\mathrm{M}} \cup (x,y), e_m, e_{\overline{m}})$; however, we did assume fixed error rates e_m and $e_{\overline{m}}$.

The probability of reaching to x NOs and y YESes, is computed using the following two functions:

- $p_m(x,y)$ is the probability that we reach to point (x,y) given that the pair is a match; and A^{M} and e_m.
- $p_{\overline{m}}(x,y)$ is the probability that we reach to point (x,y) given that the pair is non-match; and A^{M} and $e_{\overline{m}}$.

Functions $p_m(x,y)$ and $p_{\overline{m}}(x,y)$ are computed using the recursive formulas in Figure 12.

Let us now generalize to the case where we do *not* know the exact error rates e_m, $e_{\overline{m}}$. Past answers, A^{M}, for a pair of records, define one distribution for e_m and one for $e_{\overline{m}}$:

$$Pr(e_m|A^{\mathrm{M}}, m) = \frac{Pr(A^{\mathrm{M}}|e_m, m)Pr(e_m|m)}{Pr(A^{\mathrm{M}}|m)} \quad (4)$$

$$Pr(e_{\overline{m}}|A^{\mathrm{M}}, \overline{m}) = \frac{Pr(A^{\mathrm{M}}|e_{\overline{m}}, \overline{m})Pr(e_{\overline{m}}|\overline{m})}{Pr(A^{\mathrm{M}}|\overline{m})} \quad (5)$$

$Pr(e_m|m)$ and $Pr(e_{\overline{m}}|\overline{m})$ are the prior distributions for the error rates and can be computed by using a training set of matches for $Pr(e_m|m)$ and a training set of non-matches for $Pr(e_{\overline{m}}|\overline{m})$. Moreover, in order to compute $Pr(A^{\mathrm{M}}|m)$ we need to integrate over e_m:

$$Pr(A^{\mathrm{M}}|m) = \int_0^1 Pr(A^{\mathrm{M}}|e_m, m)Pr(e_m|m)de_m \quad (6)$$

and to compute $Pr(A^{\mathrm{M}}|\overline{m})$ we need to integrate over $e_{\overline{m}}$:

$$Pr(A^{\mathrm{M}}|\overline{m}) = \int_0^1 Pr(A^{\mathrm{M}}|e_{\overline{m}}, \overline{m})Pr(e_{\overline{m}}|\overline{m})de_{\overline{m}} \quad (7)$$

Based on the computed distributions for e_m and $e_{\overline{m}}$, we can extend functions $p_m(x,y|e_m)$ and $p_{\overline{m}}(x,y|e_{\overline{m}})$, for the general case where e_m and $e_{\overline{m}}$ are *not* known:

$$p_m(x,y) = \int_0^1 p_m(x,y|e_m) * Pr(e_m|m, A^{\mathrm{M}})de_m \quad (8)$$

and

$$p_{\overline{m}}(x,y) = \int_0^1 p_{\overline{m}}(x,y|e_{\overline{m}}) * Pr(e_{\overline{m}}|\overline{m}, A^{\mathrm{M}})de_{\overline{m}} \quad (9)$$

When we detect that a pair of records is difficult, we switch to the pairwise interface for this pair. In this case, multi-item answers are used to compute the prior match probability for that pair, using Equations 6 and 7, and:

$$Pr(m|A^{\mathrm{M}}) = \frac{Pr(A^{\mathrm{M}}|m)Pr(m)}{Pr(A^{\mathrm{M}}|m)Pr(m) + Pr(A^{\mathrm{M}}|\overline{m})Pr(\overline{m})} \quad (10)$$

$$Pr(\overline{m}|A^{\mathrm{M}}) = \frac{Pr(A^{\mathrm{M}}|\overline{m})Pr(\overline{m})}{Pr(A^{\mathrm{M}}|m)Pr(m) + Pr(A^{\mathrm{M}}|\overline{m})Pr(\overline{m})} \quad (11)$$

Therefore, once we switch to the pairwise interface for two records, we *a)* "replace" the machine answer prior $Pr(\overline{m})$, with the priors from Equations 10 and 11, and *b)* use the exact same computations (Equations 4 to 9) with the multi-item only answers case, but we use only the pairwise answers A^{P} instead of the multi-item answers A^{M}; initially $A^{\mathrm{P}} = \emptyset$.

B. EXPECTED COSTS' COMPUTATION

Here, we describe how to compute the expected cost of resolving a pair of records (i.e., determining if the pair is a match or a non-match) via pairwise or multi-item tasks. The computation of those costs is used for the detection of difficult pairs, in Section 5.

As in the previous section, we start from the simple case where the error rates, e_m and $e_{\overline{m}}$, are known for the pair. Here, however, we start from the computation of the expected cost to resolution via the pairwise interface and we then discuss the computation of the expected cost via the multi-item interface.

A pair is resolved when it reaches a point in F_d or when it reaches a point in F_o, in case of pairwise answers. In case we reach a point in F_o with pairwise answers, it means that this (difficult) pair cannot be identified as a match or non-match even when using the pairwise interface; and there is no point in spending more money in trying to resolve it.

For pairwise answers, the expected cost to reach to a point in F_d given that the pair is a match, is:

$$C_d(e_m, m) = \sum_{(x,y) \in F_d} c^{\mathrm{P}} * (x+y) * p_m(x,y|e_m) \quad (12)$$

while the expected cost to reach to a point in F_d given that the pair is a non-match, is:

$$C_d(e_{\overline{m}}, \overline{m}) = \sum_{(x,y) \in F_d} c^{\mathrm{P}} * (x+y) * p_{\overline{m}}(x,y|e_{\overline{m}}) \quad (13)$$

Similarly, we can compute the expected cost to reach to a point in F_o given that the pair is a match:

$$C_o(e_m, m) = \sum_{(x,y) \in F_o} c^{\mathrm{P}} * (x+y) * p_m(x,y|e_m) \quad (14)$$

or given that the pair is a non-match:

$$C_o(e_{\overline{m}}, \overline{m}) = \sum_{(x,y) \in F_o} c^{\mathrm{P}} * (x+y) * p_{\overline{m}}(x,y|e_{\overline{m}}) \quad (15)$$

Overall, the estimated cost for a specific pair that has switched to the pairwise interface, given that the pair is a match, pairwise answers A^{P} and a fixed error rate e_m, is:

$$C(e_m, m) = C_d(e_m, m) + C_o(e_m, m) \quad (16)$$

Algorithm 1 *Stage 2*

```
1: L_GR := Greedy(E, B)
2: L_D := ∅
3: for each q ∈ D do
4:    for each seq ∈ [1, f^P] do
5:       q.seq := seq
6:       compute Pr(U_q = 1) using Lemma 2
7:       append q to L_D
8:    end for
9: end for
10: sort L_D based on Pr(U_q = 1), in desc order
11: for each i ∈ [0, |L_GR|] do
12:    L'_GR := remove the last i tasks from L_GR
13:    j := ⌊(i*c^M)/c^P⌋
14:    L'_D := the first j tasks in L_D
15:    if EU(L'_D, L'_GR) > EU(P_GR, T_GR) then
16:       P_GR, T_GR := L'_D, L'_GR
17:    end if
18: end for
19: return P_GR, T_GR
```

Algorithm 2 *Greedy - Stage 1*

```
1: n := the number of all records in the dataset
2: L_GR := ∅
3: while |L_GR| * c^M < B do
4:    for each T out of all possible (n k) tasks do
5:       if EU(∅, L_GR ∪ T) > EU(∅, L_GR ∪ T_max) then
6:          T_max := T
7:       end if
8:    end for
9:    append T_max to L_GR
10: end while
11: return L_GR
```

and given that the pair is a non-match:

$$C(e_{\overline{m}}, \overline{m}) = C_d(e_{\overline{m}}, \overline{m}) + C_o(e_{\overline{m}}, \overline{m}) \quad (17)$$

Using the error rates' distributions of Equations 4 and 5 and the cost formulas for fixed error rates from Equations 16 and 17, we can compute the general estimated cost, given that the pair is a match:

$$C(m) = \int_0^1 C(e_m, m) Pr(e_m | A^{\mathrm{P}}, m) de_m \quad (18)$$

and the general estimated cost given that the pair is not a match:

$$C(\overline{m}) = \int_0^1 C(e_{\overline{m}}, \overline{m}) Pr(e_{\overline{m}} | A^{\mathrm{P}}, \overline{m}) de_{\overline{m}} \quad (19)$$

The overall cost is given by the following formula:

$$\mathbb{C}^{\mathrm{P}} = C(m) Pr(m | A^{\mathrm{P}}) + C(\overline{m}) Pr(\overline{m} | A^{\mathrm{P}}) \quad (20)$$

where $Pr(m|A^{\mathrm{P}})$ and $Pr(\overline{m}|A^{\mathrm{P}})$ are computed using Equations 10 and 11.

In case of multi-item answers, the expected cost to resolution, $\mathbb{C}^{\mathrm{M}}$, is computed the same way, using Equations 12 to 20. The only difference is with the cost computation for the case we reach a point in F_o. In particular, Equations 14 and 15, become:

$$C_o(e_m, m) = \sum_{(x,y) \in F_o} [\mathbb{C}^{\mathrm{P}} + c^{\binom{k}{2}} * (x + y)] * p_m(x, y | e_m) \quad (21)$$

$$C_o(e_{\overline{m}}, \overline{m}) = \sum_{(x,y) \in F_o} [\mathbb{C}^{\mathrm{P}} + c^{\binom{k}{2}} * (x + y)] * p_{\overline{m}}(x, y | e_{\overline{m}}) \quad (22)$$

($c^{\binom{k}{2}}$ is the cost-per-pair for a k-item task, i.e., $c^{\binom{k}{2}} = c^{\mathrm{M}} / \binom{k}{2}$.)

That is, if we reach a point f in F_o, we will pay the cost of reaching that point from the current point, $c^{\binom{k}{2}} * (x + y)$, plus the expected cost for resolving the pair using pairwise questions, $\mathbb{C}^{\mathrm{P}}$, from point f. Note that the expected cost

Algorithm 3 *Edges - Stage 1*

```
1: L := ∅
2: for each q ∈ E do
3:    for each seq ∈ [1, f^M] do
4:       q.seq := seq
5:       compute Pr(U_q = 1)
6:       append q in L
7:    end for
8: end for
9: sort L based on Pr(U_q = 1), in asc order
10: L_E := ∅
11: while |L_E| * c^M < B do
12:    E := remove the last ⌊k/2⌋ questions from L
13:    T := create a task with all the records referenced in E
14:    append T to L_E
15: end while
16: return L_E
```

Algorithm 4 *2Greedy - Stage 1*

```
1: Recs := the set of all records
2: L := ∅
3: for each q ∈ E do
4:    for each seq ∈ [1, f^M] do
5:       q.seq := seq
6:       compute Pr(U_q = 1)
7:       append q in L
8:    end for
9: end for
10: sort L based on Pr(U_q = 1), in asc order
11: L_2G := ∅
12: while |L_2G| * c^M < B do
13:    q_max := remove the last question from L
14:    T := the two records in q_max
15:    while |T| < k do
16:       for each r ∈ Recs \ T do
17:          if EU(∅, L_2G ∪ {T ∪ r}) > EU(∅, L_2G ∪ {T ∪ r_max}) then
18:             r_max := r
19:          end if
20:       end for
21:       append r_max to T
22:    end while
23:    append T to L_2G
24:    update L by removing the questions covered by T
25: end while
26: return L_2G
```

$\mathbb{C}^{\mathrm{P}}$ depends on which point in F_o we reached, as different points define different priors, $Pr(m|A^{\mathrm{M}})$, for the resolution using pairwise questions.

C. ALGORITHMS

In Algorithm 1, we give the algorithmic description of the second stage shared by Greedy, Edges, and 2Greedy. In Line 1, the first stage of Greedy (or Edges or 2Greedy) is invoked, which produces a sorted list of multi-item tasks. In Lines 2 to 10, a sorted list of pairwise tasks is generated. The last part of the algorithm (Lines 11 to 19) selects a subset of the produced multi-item tasks and a subset of the pairwise tasks, so that the budget constraint is not violated and the overall utility is maximized.

The first stage of *Greedy* is given in Algorithm 2. The algorithm mainly consists of a loop (Lines 4 to 8), that examines, in each step, all possible combinations of records to a single k-item task. The k-item task that increases the most the value of the objective function, EU, is appended to the list of tasks selected so far (Line 9). Once the cost of the selected tasks reaches budget B, the algorithm terminates.

The first stage of Edges is given in Algorithm 3. Algorithm 3 first generates all possible questions for non-difficult pairs in Lines 1 to 8, then sorts them based on $Pr(U_q = 1)$

(Line 9), and, in Lines 10 to 15, groups the questions with the highest $Pr(U_q = 1)$ into multi-item tasks. Intuitively, *Edges* assures that the $\lfloor \frac{k}{2} \rfloor$ out of the $\binom{k}{2}$ questions in each k-item task selected, have a high $Pr(U_q = 1)$.

The first stage of 2Greedy is given in Algorithm 4. The Lines 14 to 22 in Algorithm 4 refer to the Lines 4 to 8 in Algorithm 2: *2Greedy* constructs multi-item tasks by greedily adding records, instead of exhaustively searching the best tasks over all $\binom{n}{k}$ possible tasks.

D. PROOFS

LEMMA 2.

$$\forall q \in Q(\mathcal{T}),\ Pr(U_q = 1) = Pr(m_q|A^{\mathrm{M}}[q]) * \sum_{(x,y)\in W(q)} p_m(x,y)$$

where *a)* m_q is the event of the two records in q referring to the same entity, *b)* F is the set of points where the pair in q is resolved , and *c)* the set of points $W(q)$ is defined as:

$$W(q) = \{(x,y) : x + y = |A^{\mathrm{M}}[q]| + q.seq - 1 \wedge (x,y) \notin F\} \quad (23)$$

PROOF: Based on Definition 2,

$$Pr(U_q = 1) = Pr(U_q = 1|m_q, A^{\mathrm{M}}[q]) * Pr(m_q|A^{\mathrm{M}}[q])$$

In each point in $W(q)$, the pair in q is not yet resolved. Given that this pair is a match, q is useful if and only if we reach one of the points in $W(q)$. (When we reach a point in $W(q)$, the next answer to consider is the answer to q, based on the sequence numbers of questions.) Moreover, the events of reaching each of those points are disjoint: we can only reach to one of the points. Hence, the probability of q being useful given that the pair in q is a match, is the sum of the probabilities of reaching each of the points in $W(q)$:

$$Pr(U_q = 1|m_q, A^{\mathrm{M}}[q]) = \sum_{(x,y)\in W(q)} p_m(x,y) \qquad \square$$

THEOREM 1. *Consider a sequence of budgets*

$$\langle B_j = j * c^{\mathrm{M}} \mid j \in \mathbb{Z}^+ \rangle$$

For any $i \in [1, |L_{\mathrm{GR}}|]$

$$EU(\emptyset, L_{\mathrm{GR}}[0:i]) \geq (1 - \frac{1}{e}) EU(\emptyset, \mathcal{T}^*_{B_i})$$

L_{GR} is the outcome of Stage 1 of Greedy ($L_{\mathrm{GR}}[0:i]$ denotes the first i multi-item tasks in the list) and $\mathcal{T}^*_{B_i}$ the set of k-item tasks that maximizes EU for a budget constraint B_i.

PROOF: We first prove that EU is an increasing submodular function via Lemmas 3, 4, and 5. Then, we use a result [28] about greedy algorithms and the maximization of increasing submodular functions, to complete the proof.

LEMMA 3. *Consider two questions, $q, q' \in Q(\mathcal{T})$, on the same pair of records, with $q.seq > q'.seq$. Then, .*

$$Pr(U_{q'} = 1|m_{q'}, A) > Pr(U_q = 1|m_q, A)$$

where $A = A^{\mathrm{M}}[q] = A^{\mathrm{M}}[q']$, since both q and q' refer to the same pair of records.

Proof: Consider the sets of points $W(q)$ and $W(q')$, based on Equation 23. Note that any path to any point in $W(q)$ goes through a point in $W(q')$. Hence, the event of reaching any point in $W(q)$ entails the event of reaching a point in $W(q')$. Now, as discussed in the proof of Lemma 2, the events of reaching the points in $W(q)$ ($W(q')$) are disjoint, and

$$Pr(U_q = 1|m_q, A) = \sum_{(x,y)\in W(q)} p_m(x,y)$$

Therefore,

$$Pr(U_{q'} = 1|m_{q'}, A) > Pr(U_q = 1|m_q, A)$$

LEMMA 4. *The objective function $EU(\emptyset, \mathcal{T})$ is submodular:*

$$EU(\emptyset, \mathcal{T}_a \cup T) - EU(\emptyset, \mathcal{T}_a) \geq EU(\emptyset, \mathcal{T}_b \cup T) - EU(\emptyset, \mathcal{T}_b)$$

when $Q(\mathcal{T}_a) \subseteq Q(\mathcal{T}_b)$.

Proof: Consider the questions within multi-item task T, $Q(\{T\})$. Note that

$$EU(\emptyset, \mathcal{T}_a \cup T) - EU(\emptyset, \mathcal{T}_a) = \sum_{q\in Q(\{T\})} Pr(U_q = 1)$$

and

$$EU(\emptyset, \mathcal{T}_b \cup T) - EU(\emptyset, \mathcal{T}_b) = \sum_{q\in Q(\{T\})} Pr(U_q = 1)$$

Since $Q(\mathcal{T}_a) \subseteq Q(\mathcal{T}_b)$, the sequence number of each question in $Q(\{T\})$ is (potentially) greater when we append T to $\mathcal{T}_b$ compared to when we append T to $\mathcal{T}_a$. Hence, based on Lemma 3, for any $q \in Q(\{T\})$, $Pr(U_q = 1)$ is lower when we append T to $\mathcal{T}_b$ compared to when we append T to $\mathcal{T}_a$. It follows that:

$$EU(\emptyset, \mathcal{T}_a \cup T) - EU(\emptyset, \mathcal{T}_a) \geq EU(\emptyset, \mathcal{T}_b \cup T) - EU(\emptyset, \mathcal{T}_b)$$

LEMMA 5. *The objective function $EU(\emptyset, \mathcal{T})$ is increasing:*

$$EU(\emptyset, \mathcal{T}_a) \leq EU(\emptyset, \mathcal{T}_b)$$

when $Q(\mathcal{T}_a) \subseteq Q(\mathcal{T}_b)$.

Proof: Consider the questions in $Q(\mathcal{T}_a) \cap Q(\mathcal{T}_b)$. We can safely assume that each question in $Q(\mathcal{T}_a) \cap Q(\mathcal{T}_b)$, has the same sequence number when in $Q(\mathcal{T}_a)$ and when in $Q(\mathcal{T}_b)$ (e.g., if in $Q(\mathcal{T}_b)$ there are 2 questions referring on the same pair, while in $Q(\mathcal{T}_a)$ there is only one question referring on that pair, we can safely assume that in the intersection we include the question with the sequence number of 1). Since $Q(\mathcal{T}_a) \subseteq Q(\mathcal{T}_b)$:

$$EU(\emptyset, \mathcal{T}_a) = \sum_{q\in Q(\mathcal{T}_a)\cap Q(\mathcal{T}_b)} Pr(U_q = 1)$$

and

$$EU(\emptyset, \mathcal{T}_b) = \sum_{q\in Q(\mathcal{T}_a)\cap Q(\mathcal{T}_b)} Pr(U_q = 1) + \sum_{q\in Q(\mathcal{T}_b)\setminus Q(\mathcal{T}_a)} Pr(U_q = 1)$$

Thus,

$$EU(\emptyset, \mathcal{T}_a) \leq EU(\emptyset, \mathcal{T}_b)$$

Now that we showed EU is an increasing submodular function, we can use the result for the approximation ratio of greedy algortihms [28]. Consider any budget $B_i = i * c^{\mathrm{M}}$. Based on the result from [28], the aggregate utility of the first i multi-item tasks greedily appended in the list L_{GR}, is at least $(1 - \frac{1}{e})$ of the utility of the optimal solution containing i multi-item tasks, $\mathcal{T}^*_{B_i}$. $\square$

THEOREM 3. *Consider a sequence of budgets*

$$\langle B_j = j * c^{\mathrm{M}} \mid j \in \mathbb{Z}^+ \rangle$$

For any $i \in [1, |L_{\mathrm{E}}|]$

$$EU(\emptyset, L_{\mathrm{E}}[0:i]) \geq \frac{1}{k} EU(\emptyset, \mathcal{T}^*_{B_i})$$

where L_{E} is the outcome of Edges Stage 1 ($L_{\mathrm{E}}[0:i]$ denotes the first i multi-item tasks in the list) and $\mathcal{T}^*_{B_i}$ the set of k-item tasks that maximizes EU for a budget constraint B_i.

PROOF: Consider the worst case scenario for Edges Stage 1: *a)* k is an odd number and *b)* the optimal set of multi-item tasks, $\mathcal{T}^*_{B_i}$, has an overall utility of $i * \binom{k}{2} * u_{max}$, where u_{max} is the highest $Pr(U_q = 1)$ for any question q (i.e., there are i multi-item tasks where all the questions within each task have the highest $Pr(U_q = 1)$). Edges chooses the questions with the highest $Pr(U_q = 1)$, so the overall utility of $L_{\mathrm{E}}[0:i]$ is at least $i * \frac{k-1}{2} * u_{max}$ (since k is an odd number, there are $\frac{k-1}{2}$ questions in each task). Therefore,

$$EU(\emptyset, L_{\mathrm{E}}[0:i]) \geq \frac{1}{k} EU(\emptyset, \mathcal{T}^*_{B_i}) \qquad \square$$