Automated Selection of Multiple Datasets for Extension by Integration

Yael Amsterdamer
Bar-Ilan University

Moran Ben-Yehuda
Bar-Ilan University

Abstract

Organizations often seek to extend their data by integration with available datasets originating from external sources. While there are many tools that recommend *how* to perform the integration for given datasets, the selection of *what* datasets to integrate is often challenging in itself. First, the relevant candidates must be efficiently identified among irrelevant ones. Next, relevant datasets need to be evaluated according to issues such as low quality or poor matching to the target data and schema. Last, jointly integrating multiple datasets may have significant benefits such as increasing completeness and information gain, but may also greatly complicate the task due to dependencies in the integration process.

To assist administrators in this task, we quantify to what extent an integration of multiple datasets is valuable as an extension of an initial dataset and formalize the computational problem of finding the most valuable subset to integrate by this measure. We formally analyze the problem, showing that it is NP-hard; we nevertheless introduce heuristic efficient algorithms, which our experiments show to be near-optimal in practice and highly effective in finding the most valuable integration.

1 Introduction

Data scientists and analysts often face information needs that extend beyond the original data and require integrating it with external datasets, which we term *extension by integration*. The rise in number of online data sources increases the potential availability, volume and coverage of relevant data. Yet, identifying which datasets are most valuable for integration is often a complicated task: each newly added dataset may include important information, but at the same time may also introduce errors and uncertainties. Moreover, it may be worthwhile enriching an initial dataset with *multiple, complementary* datasets. The challenge then is that the value of each potential dataset is dependent on the choice of other datasets. Some datasets may indeed be complementary to each other, while some information in a given dataset may be redundant if we have already integrated another one, etc.

For example, assume Tamara is a data analyst working with a dataset of products. To gain further insights on why certain products had high or low sales, Tamara wishes to extend her data with additional information. For instance, for each manufacturer, add details about the company; for each manufacture country, add details on the economy of that country; and so on. Tamara finds a

relevant relation regarding companies, but when she integrates it with the product dataset, she discovers the result contains mostly NULL values. Apparently, the added relation only contained data on companies from a certain region. Integrating datasets about companies in other regions improves the outcome and the number of NULL values decreases. Then, Tamara discovers that she has many overlaps in her dataset. She does not wish to pay for the use of redundant data. What datasets should she retain? Which ones are the most suitable in terms of schema, coverage, quality?

To this end, we formalize and study in this paper the problem of automatically selecting multiple datasets to be integrated with a given dataset. The formalization is based on quantifying the cost and gain associated with datasets selection. Intuitively, the gain is based on the number of acquired table cells that are (estimated to be) truthful and correctly integrated with the initial dataset. Our model weighs different such facets, where weights are obtained from existing data integration tools that measure the uncertainty of integration in different ways. The cost of integration may be explicitly specified as a price per candidate dataset, (e.g., the price of data usage rights). Another cost factor is a decrease in quality as a result of the integration. This decrease is the counterpart of our gain notion, accounting for acquired data values that are either incorrect or incorrectly integrated. Another cause of quality decrease could be the increase of incompleteness (NULL values), either due to incompleteness of the candidate dataset or due to low coverage of the newly introduced attributes with respect to the original data. The model is detailed in Section 2.

Our model leads to the problem of finding an optimal extension. We show that the problem is FP^{NP}-complete, via a reduction from Set Cover to the corresponding decision problem (Section 3.1). Still, we are able to provide PTIME heuristics (Section 3.2) that are experimentally shown to be effective in practice. The general scheme of our solution is greedy, choosing at each point the next best match; yet, the function that we optimize is non-convex, and therefore we exhaustively integrate as many relations as possible, and then backtrack and return the best integration among the intermediate results. Crucially, we develop optimizations that identify cases where the marginal contribution of a given relation is bounded, allowing to significantly prune candidates for integration. The solution is detailed in Section 3.

We have implemented our algorithms, employing further optimizations based on locality sensitive hashing (LSH) to speed-up each pairwise integration. We experimentally evaluated the quality of our solutions using multiple benchmarks. Our results show that our solutions are highly effective in balancing the integration gain and cost tradeoff: in any case where a brute-force computation of the optimal solution was feasible, the results of our algorithms were either equal or very close to it. The results of our algorithms were much better than those of baseline alternatives that we have explored. In terms of execution time, our solutions scale well beyond problem sizes that the brute-force can handle, and are feasible even when there are many candidate matches. The implementation and experiments are detailed in Section 4.

The paper is organized as follows. We provide a formal model for extension by integration and for gain and cost, which serve to define our problem, in Section 2. In Section 3, we provide hardness proofs as well as a solution scheme and optimizations. We implement our solution and conduct an experimental study in Section 4. We discuss related work in Section 5 and conclude in Section 6.

Products

Prod	Manu.	Country	Sales
GreatPad X4000	BCnD	South Africa	50M
GreatPad Y600	BCnD	South Africa	60M
Superb Vital	Macron	NULL	40M
Smarterbook Elite	Netter	Saudi Arabia	10M
Smarterbook Emerge	Netter	Saudi Arabia	5M

Table 1: Example input relation Products

CA (Companies in Africa)

Company	Located	Category	Rev.
Avocado	Ethiopia	Technology	115.8
BCnD	S. Africa	NULL	324.1
Macron	NULL	IT	155.3
Transact	Senegal	Finance	87.6
XYnZ	Tunisia	NULL	252.2

MEIT (Middle Eastern IT)

Name	Country	Revenue
Macron	Egypt	155
Netter	UAE	32
Opportune	Qatar	79
Promot	Israel	35
QueenTech	Jordan	28

Table 2: Example candidates for extending Products

2 Preliminaries

We start by recalling the standard relational model, extended by a notion of correctness probability. Formally, a relation R_i is associated with a finite set of attributes $\text{sort}(R_i) = U^i = \{U^i_1, U^i_2, \ldots, U^i_{|U^i|}\}$ $\subseteq$ U for some universal (possibly infinite) attribute set U. We identify the relation R_i with an instance, such that R_i is a finite set of tuples of the form $t : U^i \rightarrow \text{Dom} \cup \{\text{NULL}\}$, where Dom is some universal domain of values and NULL $\neq \in$ Dom represents a missing value. Each R_i has a key $\text{key}(R_i)$. To simplify definitions, we assume a single key consisting of a single attribute, w.l.o.g. U^i_1, with unique, non-NULL values. We note, however, that our solution supports multiple keys. $\text{P}^{\text{correct}}(R_i)$ denotes an a-priory probability of an error in any non-NULL value in R_i. We use $\mathcal{R}$ to denote a set of relations. We discuss below, in Section 4.1, preprocessing of datasets, in particular for estimating the probability of an error and finding keys.

2.1 Dataset Extension by Integration

Given an initial relation, R_0, we next consider its extension by integrating additional relations to it from a collection of candidates. As details below, our framework does not aim to redefine basic operations in integration but to make an effective use of them as black-boxes (see Section 5 for possible implementations). Since most tools for data integration operate *pair-wise*, comparing two relations at-a-time, we will obtain an integration of multiple datasets through a sequence of pair-wise integration operations.[1] Our model supports *asymmetry* in the sense that we distinguish the initial relation from the one integrated to it, and thus allow the order of integration operations to affect the integration

[1] Given an initial dataset with multiple relations, we can repeat the process for each relation.

$\mathcal{I}_{\text{res}}$(**Products**, **MEIT**)

Prod (Products)	Manu. (Products)	Country (Products)	Sales (Products)	Revenue (MEIT)
GreatPad X4000	BCnD	South Africa	50M	NULL
GreatPad Y600	BCnD	South Africa	60M	NULL
Superb Vital	Macron	Egypt	40M	155
Smarterbook Elite	Netter	Saudi Arabia	10M	32
Smarterbook Emerge	Netter	Saudi Arabia	5M	32

Table 3: Example integration result of Products and MEIT.

result. This could be used, e.g., to give precedence to the values integrated earlier, and specifically to the input relation's values.

Matching attributes Our system expects as input a black-box algorithm that matches attributes of two relations. We next formally define the input and output of this black-box.

Definition 2.1. *Given two relations, R_i and R_j, an* attribute matching algorithm *returns as output*

(i) *a function* $\text{attMatch}_{\text{R}_\text{i},\text{R}_\text{j}}(\cdot) : U^i \to U^j \cup \{\perp\}$ *maps every attribute in U^i to a matching attribute in U^j, or to $\perp$ if it has no match; and*

(ii) *a function* $\text{P}^{\text{attMatch}}_{\text{R}_\text{i},\text{R}_\text{j}}(\cdot) : U^i \to [0,1]$ *– an (estimated) likelihood for an attribute pair match to be correct.*

We say that R_i, R_j are matched if $\text{attMatch}_{\text{R}_\text{i},\text{R}_\text{j}}(\text{key}(R_j)) \neq \perp$.

Denote by $\text{attMatch}^{\text{-1}}_{\text{R}_\text{i},\text{R}_\text{j}}(U^j_l)$ *the inverse mapping, again assigning $\perp$ to unmatched attributes in U^j.*

The attribute matching defines the schema for an integrated relation $R_{i,j} = \mathcal{I}_{\text{res}}(R_i, R_j)$. Its attributes are $U_{i,j} := U^i \cup U^j - \{U^j_l \in U^j \mid \text{attMatch}^{\text{-1}}_{\text{R}_\text{i},\text{R}_\text{j}}(U^j_l) \neq \perp\}$, i.e., retaining one copy of each matched attribute.

Example 2.2. *Recall the datasets in Table 2, and let us number the attributes from left to right in each relation. Consider an integration between relations* Products *and* MEIT*, matching the attributes such that* $\text{attMatch}(U^{\text{Products}}_2) = U^{\text{MEIT}}_1$ *(manufacturer to name) and* $\text{attMatch}(U^{\text{Products}}_3) = U^{\text{MEIT}}_3$ *(country attributes). The attributes of the integration result, after removing the two matched attributes in* MEIT*, would be as in Table 3.*

Linking records Another input to our system is a black-box tool that matches between records of integrated relations, given the matched attributes. We define a weight for the linking based on the probability of matching the key attribute and the matching of values. Given two values, x and y, we denote by $\text{P}^{\text{valMatch}}(x, y)$ the likelihood of these values to have the same meaning.

We next define the result of record linking as well as its weight.

Definition 2.3. *Given two relations, R_i and R_j and an attribute matching function* $\text{attMatch}_{\text{R}_\text{i},\text{R}_\text{j}}$*, a* record linking algorithm *returns as output a function* $\text{link}_{\text{R}_\text{i},\text{R}_\text{j}} : R_i \to R_j \cup \perp$ *mapping each tuple in R_i to a matching tuple in R_j if exists.*

The linking between tuples $t \in R_i, t' \in R_j$ yields a linking result *tuple* $\text{res}^{\text{link}}_{\text{R}_\text{i},\text{R}_\text{j}}(t) = t''$ *defined as follows.*

- *When* $\text{attMatch}_{\text{R}_\text{i},\text{R}_\text{j}}(U^i_k) = \perp$ *then* $t''(U^i_k) = t(U^i_k)$*;*
- *If* $\text{attMatch}_{\text{U}^\text{i}_\text{k},\text{R}_\text{j}}(=)U^j_l$ *then if* $U^i_k(t) \neq \mathsf{NULL}$ *or* $t' = \perp$ *we set* $U^i_k(t'') = U^i_k(t)$*, and otherwise* $U^i_k(t'') = U^j_l(t')$*.*

Denote $U = \text{key}(R_i)$*,* $U' = \text{attMatch}_{\text{R}_\text{i},\text{R}_\text{j}}(U) \neq \perp$ *and* $t' = \text{link}_{\text{R}_\text{i},\text{R}_\text{j}}(t)$*. The weight for the record link is defined as* $\text{W}^{\text{link}}_{\text{R}_\text{i},\text{R}_\text{j}}(t) :=$

$$\text{P}^{\text{attMatch}}_{\text{R}_\text{i},\text{R}_\text{j}}(U) \cdot \text{W}^{\text{correct}}(U,t) \cdot \text{W}^{\text{correct}}(U',t') \cdot \text{P}^{\text{valMatch}}(t(U), t'(U'))$$

where $\text{W}^{\text{correct}}(U,t)$ *is a weight assigned to reflect the correctness of* $t(U)$ *(we derive its formulation in Def. 2.6).*

Intuitively, our definition of an integrated tuple resembles a left outer join, using the values of both original tuples, and NULLs for the right tuple if not exists. For matched columns, we take the non-NULL value between the two tuples, and if both are non-NULL, we prefer the left tuple. The weight assigned to a link is an aggregation of our confidence in the attributes and value matches, as well as in the correctness of the individual tuples.

We define further two indicator functions that will be useful in the sequel, and that distinguish two types of values introduced via the integration.

- $t''(U^j_l)$ is a non-NULL value added through an unmatched column from R_j, denoted by the indicator function $\text{added}(U^j_l, t'', R_i, R_j) \in \{0,1\}$.
- $U^i_k(t'')$ is a non-NULL value obtained by replacing a NULL in R_i by a value from R_j for a matched attribute and record, denoted by the indicator function $\text{resolved}(U^i_k, t'', R_i, R_j) \in \{0,1\}$, since it corresponds to resolving a missing value.

Example 2.4. *Continuing Example 2.2, we now consider the linking of t, the third tuple in* Products *with t', the first in* Meit*, resulting in t'', the third tuple in Table 3. The first, second and fourth values were taken from t, the rest are from t'. In this case, the integration helped us resolve a* NULL *in* $t(U^{\text{Products}}_3)$ *(missing country), which we denote by* $\text{resolved}(U^{\text{Products}}_3, t'', \text{Products}, \text{Meit}) = 1$*. The last value of t'', 155, belongs to an attribute* $U^{\text{MEIT}}_3(t)$ *(Revenue) added to the integration result, denoted by* $\text{added}(U^{\text{MEIT}}_3, t'', \text{Products}, \text{Meit}) = 1$*.*

Now consider the weight of linking, $\text{W}^{\text{link}}_{\text{Products,Meit}}(t)$*. The Manu. column in* Products *is matched to a key of* MEIT*. Say* $\text{P}^{\text{attMatch}}_{\text{Products,MEIT}}(U^{\text{Products}}_2) = 0.95$*,* $\text{P}^{\text{correct}}(\text{Products}) = 1$*,* $\text{P}^{\text{correct}}(\text{MEIT}) = 0.8$ *and* $\text{P}^{\text{valMatch}}(\textit{Macron}, \textit{Macron}) = 1$*. The linking weight is then* $0.95 \cdot 1 \cdot 0.8 \cdot 1 = 0.76$*.*

As another example, note that for the first two tuples of Product*, no tuple of* MEIT *was matched and so their value for Revenue is* NULL*. Further note that in case of value disagreement, e.g., on the country of Netter (tuples 4-5 of* Products*, tuple 2 of* MEIT*), we use the value of the left relation (Saudi Arabia).* □

Overall integration result We now use the above definitions to obtain an overall definition of relation extension by integration.

Definition 2.5. *We define an integration as a tuple* $\mathcal{I} = (\text{attMatch}, \text{P}^{\text{attMatch}}, \text{P}^{\text{valMatch}}, \text{link}, \mathcal{I}_{\text{res}})$*, where* attMatch, $\text{P}^{\text{attMatch}}$ *are defined in Def. 2.1,* $\text{P}^{\text{valMatch}}$ *in the beginning of the* Record linking *par. and* link *in Def. 2.3. Given two relations* R_i, R_j*, we define* $\mathcal{I}_{\text{res}}(R_i, R_j) = \{t \mid \exists t' \in R_i, t = \text{res}^{\text{link}}_{\text{R}_\text{i},\text{R}_\text{j}}(t)\}$.

2.2 Gain and Cost of Integration

We now formally define notions of gain and cost in our context, based on the correctness of data and operations captured by our model and measures of information gain and quality loss. We start by assigning weights to values based on our estimation of their correctness, in turn derived from the initial dataset correctness probabilities and from our estimation of the linkage and matches correctness.

Definition 2.6. *Let* $t \in R$ *be a tuple,* U *be an attribute of* R *such that* $t(U) \neq$ NULL.

- *If* R *is an initial relation (as opposed to the result of an integration), we define* $\text{W}^{\text{correct}}(U, t) = \text{P}^{\text{correct}}(R)$.
- *Else if* $\text{added}(U, t, R_i, R_j)$ *and let* $t = \text{res}^{\text{link}}_{\text{R}_\text{i},\text{R}_\text{j}}(t')$*, we compute* $\text{W}^{\text{correct}}(U, t) = \text{W}^{\text{link}}_{\text{R}_\text{i},\text{R}_\text{j}}(t') \cdot \text{W}^{\text{correct}}(U, \text{link}_{\text{R}_\text{i},\text{R}_\text{j}}(t'))$.
- *Else if* $\text{resolved}(U, t, R_i, R_j)$*, and let* $t = \text{res}^{\text{link}}_{\text{R}_\text{i},\text{R}_\text{j}}(t')$*, we compute* $\text{W}^{\text{correct}}(U, t) = \text{W}^{\text{link}}_{\text{R}_\text{i},\text{R}_\text{j}}(t') \cdot \text{P}^{\text{attMatch}}_{\text{R}_\text{i},\text{R}_\text{j}}(U) \cdot \text{W}^{\text{correct}}(\text{attMatch}_{\text{R}_\text{i},\text{R}_\text{j}}(U), \text{link}_{\text{R}_\text{i},\text{R}_\text{j}}(t'))$

Gain Our gain function is based on a notion of informativeness, by counting the expected number of (correct) values added/resolved by the integration process.

Definition 2.7. *Given two relations* R_i, R_j*, the gain from their integration is defined as*

$$\text{gain}(R_i, R_j) := \sum_{t \in \mathcal{I}_{\text{res}}(R_i, R_j)} \sum_{\substack{U \in U_{i,j} \\ t(U) \neq \text{NULL}}} \text{W}^{\text{correct}}(U, t)$$

Next, we consider three complementary factors of integration cost, concerning incompleteness, errors and fixed integration cost (e.g., monetary). All of these naturally depend on black-box integration solution; the $\mathcal{I}$ notation is omitted for brevity when clear from context.

Incompleteness cost The first cost factor is *completeness*, namely the number of NULL in the integration result, which is denoted by $\text{Cost}_{\mathsf{NULL}}(\mathcal{I}_{\text{res}}(R_i, R_j))$. During the integration process, NULL values may be included in integrated tables, but also introduced through added attributes and unlinked records, even if the integrated relation contains no NULLs. This is demonstrated by the following example.

Example 2.8. *Returning to our running example, for a simple case of increasing incompleteness by integration an incomplete relation, consider the integration of* CA *to* Products *in Table 2.* CA *includes three* NULL *values, that will also appear in the integration result, i.e., we have* $\text{Cost}_{\mathsf{NULL}}(\mathcal{I}_{\text{res}}(\mathsf{Products}, \mathsf{CA})) = 4$ *along with the single* NULL Products*. In contrast, relation* MEIT *from Table 2 contains no* NULL*s; but since the first two records of* Product *(BCnD) are not linked to any record in* MEIT*, they have* NULL *values in the newly-added attribute Revenue (see Table 3). Moreover, in this case* MEIT *resolves the* NULL *value in the third record of* Products*, resulting in* $\text{Cost}_{\mathsf{NULL}}(\mathcal{I}_{\text{res}}(\mathsf{Products}, \mathsf{MEIT})) = 2$

Error cost The *error cost* is defined as the expected number of erroneous (non-NULL) values in the integration result, and its formula is thus symmetric to the gain formula.

Definition 2.9. *Given two relations R_i, R_j, the error cost of their integration is defined as*

$$\text{Cost}_{\text{err}}(\mathcal{I}_{\text{res}}(R_i, R_j)) := \sum_{t \in \mathcal{I}_{\text{res}}(R_i, R_j)} \sum_{\substack{U \in U_{i,j} \\ t(U) \neq \mathsf{NULL}}} (1 - \text{W}^{\text{correct}}(U, t))$$

Note that by definition, our weights are in $[0, 1]$ and resemble probabilities, hence subtracting from 1 in the above formula makes sense. In particular, by this definition, if $R = \mathcal{I}_{\text{res}}(R_i, R_j)$ and $U = U_{i,j}$ then $|R| \cdot |U| = \text{gain}(R) + \text{Cost}_{\mathsf{NULL}}(R) + \text{Cost}_{\text{err}}(R)$.

Fixed cost per integration Finally, in some cases a fixed cost is attached to each dataset integration. This could be the case, e.g., when the use of datasets is associated with monetary cost, or to penalize a larger number of integrated tables.

Definition 2.10. *Given two relations R_i, R_j, the* fixed cost *of their integration is defined by* $\text{Cost}_{\text{fixed}}(\mathcal{I}_{\text{res}}(R_i, R_j)) := \text{Cost}_{\text{fixed}}(R_i) + \text{Cost}_{\text{fixed}}(R_j)$*, where for each candidate dataset $R_i \in \mathcal{R}$ (not the result of an integration),* $\text{Cost}_{\text{fixed}}(R_i)$ *is given.*

Combining cost and gain We now combine the gain and cost functions together to obtain an optimization problem definition, as follows.

Definition 2.11. *Let R_0 be an initial dataset, $\mathcal{R} = \{R_1, \ldots, R_n\}$ a set of candidate datasets, $\mathcal{I}$ an integration and $\alpha, \beta, \gamma \in [0, 1]$ parameters. We define* `OPT-EXTENSION` *as the problem of finding a sequence $[R_{i_1}, R_{i_2}, \ldots, R_{i_k}]$ such that*

$$R = \mathcal{I}_{\text{res}}(\ldots \mathcal{I}_{\text{res}}(\mathcal{I}_{\text{res}}(R_0, R_{i_1}), R_{i_2}) \ldots, R_{i_m})$$

i.e., sequentially integrating them in order, maximizes

$$\text{score}(R, \alpha, \beta, \gamma) := \text{gain}(R) - (\alpha\,\text{Cost}_{\texttt{NULL}}(R) + \beta\,\text{Cost}_{\text{err}}(R) + \gamma \cdot \text{Cost}_{\text{fixed}}(R)) \quad (1)$$

For brevity, we omit α, β, γ from the score$(\cdot)$ input when their values are clear or irrelevant in the context.

3 Results

We next show a hardness result for our problem and then detail a heuristic scheme that effectively solves it in practice.

3.1 Hardness of `OPT-EXTENSION`

We first define the decision problem corresponding to `OPT-EXTENSION`.

Definition 3.1. *Let R_0 be an initial dataset, $\mathcal{R} = \{R_1, \ldots, R_n\}$ a set of candidate datasets, $\mathcal{I}$ an integration, $\alpha, \beta, \gamma \in [0, 1]$ parameters and Θ a target score. We define `DEC-EXTENSION` as the problem of deciding whether there exists a sequence $[R_{i_1}, R_{i_2}, \ldots, R_{i_k}]$ such that* $\text{score}(\mathcal{I}_{\text{res}}(\ldots \mathcal{I}_{\text{res}}(\mathcal{I}_{\text{res}}(R_0, R_{i_1}), R_{i_2}) \ldots, R_{i_m})) \geq \Theta$.

Proposition 3.2. *Given an initial relation R_0, a set of candidate relations $\{R_1, \ldots, R_n\}$ and parameters α, β, γ, and a polynomial-time integration $\mathcal{I}$, solving `OPT-EXTENSION` is FP^{NP}-hard[2] in the number of candidate relations, n.*

Conversely, we have a FP^{NP} algorithm for finding a solution R such that $\text{score}(R) = \lfloor \text{score}(\mathsf{OPT}) \rfloor$, *where* OPT *is the solution to `OPT-EXTENSION`.*

Proof (sketch). Hardness is proved by a reduction from `SET COVER`. Membership is proved by an algorithm using oracle calls to `DEC-EXTENSION`, first in a binary search to determine the optimal Θ, and then in incrementally selecting the relations to integrate that allow reaching the target score of the integration result. □

While we show a reduction from `SET COVER`, known approximation results for `SET COVER` do not apply for our score function: the marginal contribution of an integrated dataset may depend on the other integrated datasets, and hence our score function is in general not monotonic or even convex. We next develop a greedy heuristic for solving `OPT-EXTENSION`, which in particular accounts for the non-convexity of the score$(\cdot)$ function.

3.2 Solutions to `OPT-EXTENSION`

We start by describing a greedy approach for solving `OPT-EXTENSION`. Our general scheme is given in Algorithm 1, which integrates as many relations as possible with the initial one, one-by-one, then returns the intermediate result with highest score (recall that the score may decrease and then increase in this process).

[2] FP^{NP} is the class of PTIME functions that use an oracle for a problem in NP.

Algorithm 1: `OPT-EXTENSION` Solver General Scheme

Input: R_0: initial relation; $\mathcal{R} = \{R_1, \ldots, R_n\}$: set of candidate relations; $\mathcal{I}$: integration black-box; α, β, γ: weights for score function; f: a function selecting a single next relation to integrate

Output: $R_{\max}$ final integration result

1 $R_{\text{curr}}, R_{\max} \leftarrow R_0$;
2 maxScore $\leftarrow$ score$(R_0, \alpha, \beta, \gamma)$;
3 **while** $\mathcal{R} \neq \emptyset$ **do**
4 $\quad R_i, \mathcal{R} \leftarrow f(R_{\text{curr}}, \mathcal{R}, \mathcal{I}, \alpha, \beta, \gamma)$;
5 $\quad$ **if** $R_i = \emptyset$ **then** break;
6 $\quad \mathcal{R} = \mathcal{R} - \{R_i\}$;
7 $\quad R_{\text{curr}} = \mathcal{I}_{\text{res}}(R_{\text{curr}}, R_i)$;
8 $\quad$ Update $R_{\max}$, maxScore if the score of R_{curr} is greater;
9 **return** $R_{\max}$;

In more detail, Algorithm 1 starts from an initial relation R_0, each time integrating it with a single relation $R_i \in \mathcal{R}$. Choosing R_i is done via a function f, which we will instantiate in the sequel with different implementations. f may discard relations from $\mathcal{R}$ – to make this explicit, the output of f also includes the modified $\mathcal{R}$. The scheme iterates until $\mathcal{R}$ is empty (line 3) or until f does not find another relation that can be integrated and returns an empty relation (line 5). Then, the intermediate integration result with highest score is returned.

Greedy approach Our first algorithm for f, `Edmint-Greedy`, inspects all the relations that *can* be integrated (namely, that have a key attribute that matches one of the attributes of the current relation), and greedily chooses the one which maximizes the integration score, formally,

$$\texttt{Edmint-Greedy}(R, \mathcal{R}, \mathcal{I}) \quad := \quad \arg\max_{R_i \in \mathcal{R}} \text{score}(\mathcal{I}_{\text{res}}(R, R_i)) \quad (2)$$

Optimizations for `Edmint-Greedy` Next, we overview an optimized implementation for the f function; this implementation identifies cases when the marginal contribution of the relation is bounded or fixed, and which allow us to immediately integrate or discard relations. We call the overall algorithm obtained by using this implementation `Edmint-Opt`.

Algorithm 2 details the steps of this optimized method. First, in lines 1-3, we discard relations whose marginal contribution is negative, based on the following observation.

Observation 3.3. *Given an integration candidate R_i, the marginal contribution of R_i when integrated to any relation R is bounded by* $\text{score}(\mathcal{I}_{\text{res}}(R, R_i)) - \text{score}(R) \leq \text{gain}(R_i) - \beta\,\text{Cost}_{\text{err}}(R_i) - \gamma\,\text{Cost}_{\text{fixed}}(R_i)$

Since the bound depends only on R_i, it will prune all the relations that match the conditions already on the first iteration of the general scheme.

Next, the algorithm iterates over the (remaining) candidate relations. In lines 7-10, the algorithm checks for relations that have no more matches to any

Algorithm 2: `Edmint-Opt` implementation of f (selection of the next relation to integrate in Algorithm 1)

Input: R: initial relation; $\mathcal{R} = \{R_1, \ldots, R_n\}$: set of candidate relations; $\mathcal{I}$: integration black-box; α, β, γ: weights for score function

Output: $R_{\max} \in \mathcal{R}$: single relation to integrate; $\mathcal{R}$ updated set of candidates

1 **for** $R_i \in \mathcal{R}$ **do**
2 maxDelta $\leftarrow \text{gain}(R_i) - \beta\,\text{Cost}_{\text{err}}(R_i) - \gamma\,\text{Cost}_{\text{fixed}}(R_i)$;
3 **if** maxDelta < 0 **then** $\mathcal{R} \leftarrow \mathcal{R} - \{R_i\}$;
4 $R_{\max} \leftarrow \emptyset$;
5 maxScore $\leftarrow -\infty$;
6 **for** $R_i \in \mathcal{R}$ **do**
7 **if** $\forall U_k^i \in U^i, \text{attMatch}^+_{\text{R},\mathcal{R}}(U_k^i) = \emptyset$ **then**
8 delta $\leftarrow \text{score}(\mathcal{I}_{\text{res}}(R, R_i)) - \text{score}(R)$;
9 **if** $\text{attMatch}^{-1}_{\text{R},\text{R}_\text{i}}(\text{key}(R_i)) \neq \perp$ *and* delta > 0 **then** Return $R_i, \mathcal{R}$;
10 **else** $\mathcal{R} \leftarrow \mathcal{R} - \{R_i\}$;
11 **else if** $\text{attMatch}^{-1}_{\text{R},\text{R}_\text{i}}(\text{key}(R_i)) \neq \perp$ **then**
12 $R' \leftarrow \mathcal{I}_{\text{res}}(R, R_i)$;
13 Update $R_{\max}$, maxScore if the score of R' is greater;
14 **return** $R_{\max}, \mathcal{R}$;

attribute in another candidate. It uses the transitive and symmetric closure of the attMatch function, denoted attMatch$^+$. While matches may change, we observe that in practice the fact that there are currently no matches at all to other candidates serves as a strong indication that no new matches for R_i will be discovered in future iterations. Consequently in such cases the marginal contribution of R_i to the final score $\text{score}(\mathcal{I}_{\text{res}}(R, R_i)) - \text{score}(R)$ may be viewed as fixed. If R_i matches the current integration result (its key is matched) with a positive marginal contribution, we can integrate R_i at any point without the need to account for the effect of hypothetical future integrations. Therefore, Algorithm 2 returns it as the selected candidate for integration (line 9). Otherwise, the algorithm immediately discards R_i (line 10).

In lines 11-13, if the marginal contribution of R_i is not fixed, we perform a hypothetical integration between R and R_i, and save in $R_{\max}$ the relation with maximal integration score (even if this score is negative). Unless we reach line 9 in one of the iterations, the overall $R_{\max}$ is returned as the selected candidate.

Complexity We next analyze the time complexity of our algorithms, depending on two main factors: the integrations computed by a black-box algorithm, and score computation.

The general scheme in Algorithm 1 line 3 may perform up to $|R|$ iterations, in each of which `Edmint-Greedy` and `Edmint-Opt` iterate over the remaining relations in $\mathcal{R}$. `Edmint-Opt` further checks each relation against all the other candidates in line 7, therefore the number of integrations performed is $O(|\mathcal{R}|^2)$ by `Edmint-Greedy` and $O(|\mathcal{R}|^3)$ by `Edmint-Opt`. In Section 4.1, we describe an architecture that can significantly reduce the number of performed integrations.

For score computation, we can observe that the contribution of each added/resolved/ cell introduced through integration to each of the gain and cost factors depends only on a fixed number of other values. E.g., an added non-null value $t(U)$ depends on $\mathrm{W}^{\mathrm{correct}}(U,t)$ which depends on $\mathrm{P}^{\mathrm{attMatch}}$ and $\mathrm{W}^{\mathrm{link}}$. We can use this property to prove that when integrating relation R_i, score computation is at most linear in the size of R_i,

4 Implementation and Experiments

We next briefly describe `Edmint`, a prototype implementation for our solutions, and then use it in an experimental study.

4.1 Implementation

The main bottleneck in implementing a solution to `OPT-EXTENSION` is the number of costly (hypothetical) integrations performed by the solution algorithms. We therefore employ a hashing-based technique, following previous work on scalable dataset integration [36]: we compute a *sketch* for each attribute – a set of hash values – such that attributes likely to be matched will coincide on hash values with high probability. A suitable index can then be used to find, in constant time, the candidate matches to each attribute. We then use the index in our implementation as follows.

- In Algorithm 1 line 4, every implementation of f uses the index first to check if the key of the candidate dataset matches any attribute of the current integration result R_{curr}, thereby avoiding candidates that do not match our current dataset. E.g., line 11 in Algorithm 2.
- In line 7 of Algorithm 2 we can check in constant time whether a dataset R_i has no potential matches, and then if its key attribute is not matched – immediately discard it, even without computing its integration score.
- When we update R_{curr} in line 7 of Algorithm 2, we only have to remove the attributes of R_i from the index, and add/recompute the hash values for the attributes that were added or included resolved values in the new R_{curr}. This can be done in time linear in the size of R_i.

The choice of hash function depends on the method for matching attributes, employed by the black-box integration tool, e.g., Jaccard set containment [12, 29, 36], Jaccard Similarity [12], Cosine Similarity and others [20].

We have implemented our solutions in a prototype system using Python 7.3 and datasketch (`https://pypi.org/project/datasketch/`) for hashing. The system accepts as input relations in CSV files, and pre-processes them with a few simple steps of normalizing the data. This included (1) deleting special characters; (2) normalizing cell format, by e.g., identifying values such as N/A and empty cells as `NULL` values, using a uniform format for numbers, phone numbers, etc.; (3) identifying and cleaning key attributes, by finding attributes that are near-keys (having $X\%$ unique values), and removing tuples with non-unique or `NULL` key. According to Section 2, our architecture includes a few black-boxes. In our experimental study, for attribute matching, we used the

Jaccard Set Containment [36]. We normalized this metric to derive matching probabilities, and used exact matching for values (with probability 1 for an exact match after preprocessing, and 0 otherwise).

4.2 Experimental Setup

We now describe the experimental study we have conducted using our system. Our experiments were conducted on a Linux machine with Intel Xeon Gold 6240 processor and 32GB of DDR4 memory.

Data collections We have used collections of real data of varying scale and heterogeneity, as follows.

- *Kaggle Collection.* This collection includes 40 relational datasets obtained from Kaggle (`https://www.kaggle.com/`), focusing on movies, books and related topics, with 120-1M tuples and 3-49 attributes per table. Overall percentage of matching relation pairs is 0.41. This collection illustrates a scenario where the user already collected datasets relevant to the topic of interest.

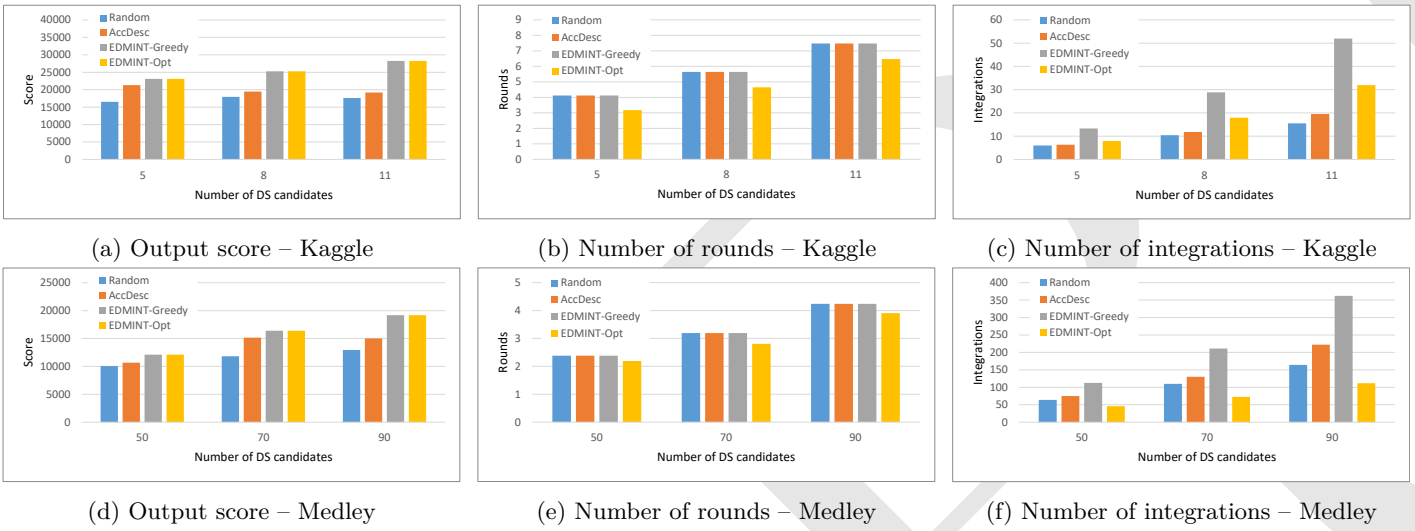
- *Medley Collection.* This collection includes 100 relational datasets on various topics (including art, music, literature, COVID-19 and others) and is a union of several smaller collections from Canadian Government Open Data (`https://open.canada.ca/en/open-data`), data.world (`https://data.world/`), data.gov (`https://www.data.gov/`), Institute of Museum and Library Services(`https://www.imls.gov/`) and the aforementioned Kaggle dataset. Each relation consists of 120-1M tuples and 2-67 attributes. Overall percentage of matching relation pairs is 0.12. This collection illustrates scenario where the users start with a data lake consisting of datasets that are not necessarily related.

Each of our experiments on these datasets is averaged over 10 executions, where we randomly draw the percentage of accurate data between 0.45-0.99 and the fixed cost of each relation, between 0 and the relation size (the former can be estimated, e.g., as in [7] and the latter can be determined by the user). We also draw the initial relation uniformly at random. The default value used for α, β, γ was 0.1.

Compared algorithms The algorithms that we compare operate according to the general scheme of Algorithm 1, namely they choose and integrate one relation at-a-time, considering only relations that can be integrated, using hashing for efficient identification of matching candidates and eventually returning the intermediate result with highest score. This renders even our baseline algorithms more effective than, e.g., algorithms that perform integrations until reaching a certain size or score of the integration result.

- *Random.* This baseline chooses at each step, uniformly at random, a relation among the remaining datasets that can be integrated.

- *AccDesc.* Integrated relations that lower the accuracy of the integration result also affect the subsequent integrations accuracy. Thus, this baseline



(a) Output score – Kaggle (b) Number of rounds – Kaggle (c) Number of integrations – Kaggle

(d) Output score – Medley (e) Number of rounds – Medley (f) Number of integrations – Medley

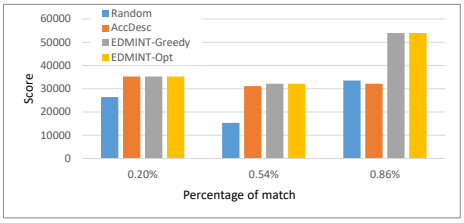
Figure 1: Experiments with varying number of input candidate relations

orders candidates by their initial correctness probability and each time integrates the most accurate dataset that can be integrated at this point.

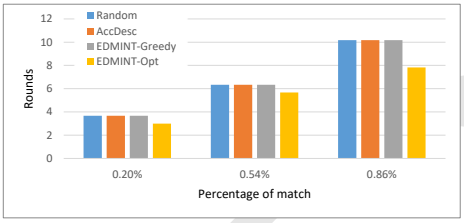
- *`Edmint-Greedy`.* As described in Section 3.2.
- *`Edmint-Opt`.* As described in Section 3.2.
- *Brute-Force.* This algorithm tests every possible order of integration of the candidate relations, and returns the one with maximal score among all the intermediate results observed. Thus, the score achieved by this algorithm is optimal. Due to its high cost, we use this baseline only in small-scale experiments.

Metrics We consider two general types of metrics for the integration result. The first is the score of the integration result, reflecting the quality of the integration, and aggregating metrics for completeness (number of NULLs), accuracy (number of errors), cost (fixed cost) and size (gain). The second type of metrics relate to the efficiency of the algorithms, which we measure by

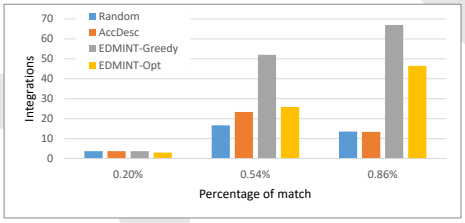
- *Number of rounds.* This metric considers the number of actual integrations performed by the algorithm, excluding hypothetical integrations. Equivalently, this is the number of iterations in the external loop of the general scheme (Algorithm 1).
- *Number of integrations.* This metric is our main indication for the algorithm cost in terms of time (and space), assuming integrations may be relatively costly, but their exact cost varies depending on the black-box implementation.
- *Execution time.* To verify that our approach is practical, we also consider execution times for the process of solving `OPT-EXTENSION`, per integration or overall.



(a) Output score



(b) Number of rounds



(c) Number of integrations

Figure 2: Varying the percentage of matching attributes for subsets of Kaggle

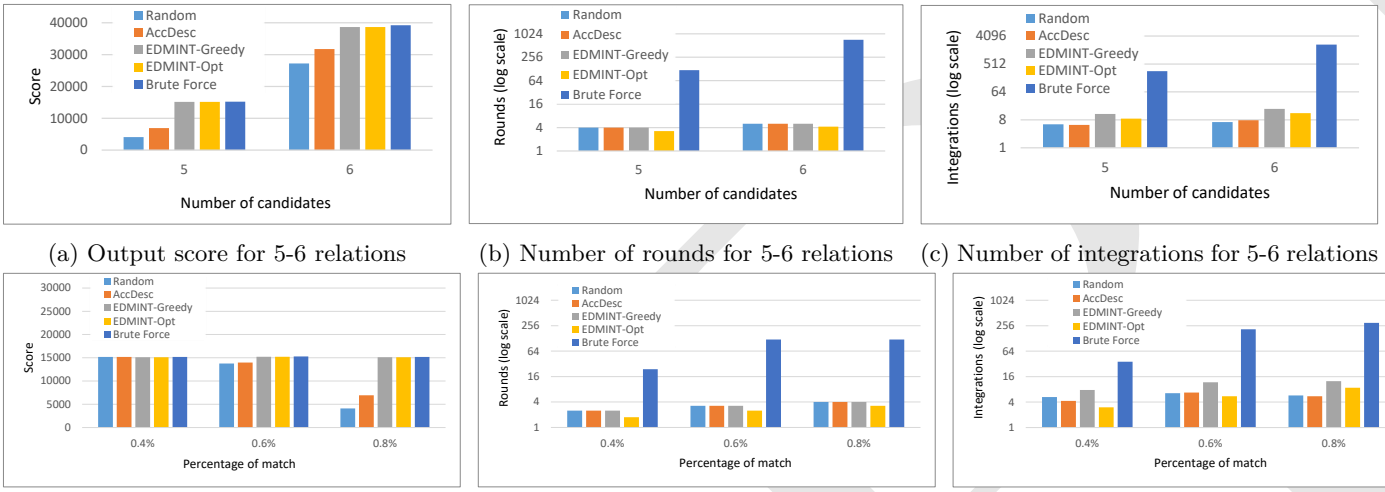
4.3 Experimental Results

Varying candidate collection sizes Our first experiment has tested the effect of the number of candidates in the initial set on the integration cost and results, by randomly choosing candidate sets of varying sizes. Figures 1a-1c show, respectively, the final score, number of rounds and number of integrations for subsets of the Kaggle Collection. Figures 1d-1f show the same for the Medley Collection. The results may be summarized as follows.

- `Edmint-Greedy` and `Edmint-Opt` tie for the best score reached, which indicates the optimizations of `Edmint-Opt` indeed do not affect the achieved score. These algorithms achieve the best score by a significant gap.
- The three metrics increase as the number of datasets increases for all algorithms, since the integration result can include more relations. The gap in score between `Edmint` and the baselines is more significant, since there is more room for optimization.
- The optimizations of `Edmint-Opt` are highly successful in reducing the number of rounds and integrations compared to `Edmint-Greedy`. The gap is more significant for the number of integrations, whose upper bound is quadratic in the number of candidate relations.
- The Medley Collection requires on average less rounds than Kaggle, since its matching probability is lower. For the former, `Edmint-Opt` is able to make even less integrations than the baselines, by effective pruning of irrelevant candidates.

The effect of matching attributes We now examine, as another factor of our system's performance, the percentage of relations, in pairs, that are apriori matched (i.e., the key of one matches the attribute of the other). For that, we select subsets of our Collections and compute the matching percentage, compute the matching percentage and order the results accordingly. Figure 2 shows the results for the Kaggle Collection.

- `Edmint-Opt` and `Edmint-Greedy` consistently tie for the best score (Figure 2a). The score does not necessarily increase with the matching percentage, since the subset of relations considered in each percentage is different. This proves that even when there are more matches, they may not be beneficial matches that increase the score.



(a) Output score for 5-6 relations

(b) Number of rounds for 5-6 relations

(c) Number of integrations for 5-6 relations

(d) Output score for 5 tables and varying matching percentage

(e) Number of rounds for 5 tables and varying matching percentage

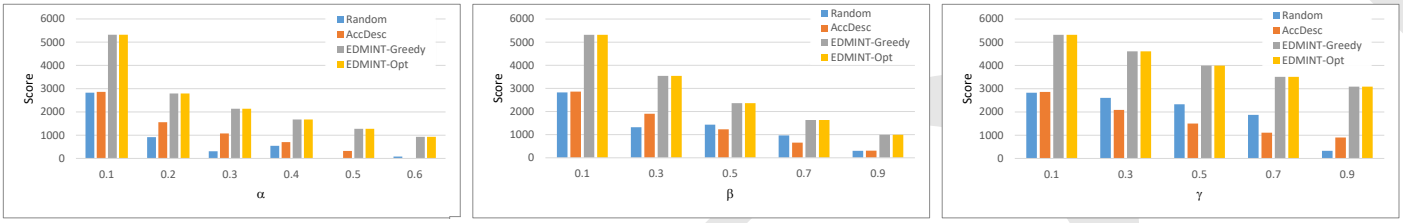
(f) Number of integrations for 5 tables and varying matching percentage

Figure 3: Comparison with Brute-Force over subsets of Kaggle

- The number of rounds (Figure 2b) and integrations (Figure 2c) increase with matching percentages since more tables are matched (even if the match is eventually not used). With low percentages `Edmint-Opt` is able to save integrations, but as the percentages increase it becomes closer to `Edmint-Greedy`, since fewer relations can be pruned.

Comparison to the optimal algorithm Next, we compare the performance of our heuristic algorithms to the optimal, Brute-Force approach. Due to the high cost of the latter, we perform these experiments on 5-6 relations from Kaggle, varying the number of candidates and the percentage of matching relation pairs – the latter is done by selecting subset with higher or lower match percentage. Our main findings are shown in Figure 3.

- `Edmint-Greedy` and `Edmint-Opt` achieve optimal or near-optimal score (Figures 3a, 3d). In the few executions where a difference was observed, it was due to different ordering of the same selected datasets, which lead to slightly different value resolution and weights.
- Again `Edmint-Opt` achieves the best balance between high scores and efficient execution.
- As expected, the number of rounds and integrations of Brute-Force are exponentially larger than other algorithms, and grows exponentially with the number of datasets (Figures 3b, 3c) whereas the others grow roughly linearly.
- The optimal score is not affected here by the percentage of matching columns, since the eventually selected integration did not increase in size (Figure 3d). However, while `Edmint-Greedy` and `Edmint-Opt` remain close



(a) Varying α (weight of $\text{Cost}_{\text{NULL}}(\cdot)$) (b) Varying β (weight of $\text{Cost}_{\text{err}}(\cdot)$) (c) Varying γ (weight of $\text{Cost}_{\text{fixed}}(\cdot)$)

Figure 4: Output scores for varying weight parameters, for the Kaggle Movie Collection

$\lvert\mathcal{R}\rvert$	$\sum\lvert U^i\rvert$	% match	Index time	Avg. Integration time			
				Random	AccDesc	`Edmint-Greedy`	`Edmint-Opt`
90	3628	0.11	07:32	<00:01	<00:01	<00:01	<00:01
70	2221	0.15	05:09	<00:01	<00:01	<00:01	<00:01
50	2358	0.13	03:33	<00:01	<00:01	<00:01	<00:01
19	300	0.63	03:07	00:02	00:01	00:02	00:02
12	183	0.85	00:35	00:01	00:01	<00:01	<00:01
5	76	0.4	00:22	<00:01	<00:01	<00:01	00:01

Table 4: Execution times for various inputs (minutes)

to optimal, the performance of the baselines deteriorates, since the number of optional matches increases, and they cannot distinguish the good matches from the bad ones.

- The numbers of rounds and integrations (Figures 3e, 3f) resemble the trends of Figure 2. The performance of Brute-Force approaches trying all the possible orders, since the high matching percentages enable more matching orders.

Varying the weight parameters Next, we consider the effect of weight parameters on the comparative performance of the different algorithms. In each experiment, we have varied the value of one parameter, with the default value 0.1 for the others. The results for the Kaggle Collection are depicted in Figure 4.

- All scores consistently decrease with the parameters, but comparatively, the decrease of `Edmint-Opt` and `Edmint-Greedy` is slower, such that relative gap between them and the baselines increases.
- $\text{Cost}_{\text{NULL}}(\cdot)$ leads to a faster decrease in score, due to the number of NULL values, and at $\alpha \geq 0.6$ the baselines are unable to match any relation. This is not surprising, since these algorithms do not consider NULLs.

Execution time Table 4 shows some examples of execution times of `Edmint` and of the baselines. For each example execution, we specify the number of

candidate relations $|\mathcal{R}|$, the overall number of attributes $\sum \left|U^i\right|$, and the percentage of matching relation pairs. We then show the time needed to compute the hash values and construct the index (see Section 4.1) and the average time per integration in the executions of the different algorithms. We can observe that the index time (which is done offline, and can be reused) generally increases with the number of relations and match percentages, and that integration time takes less than a second or at most 2 seconds. This shows the efficiency of our framework for a computationally nontrivial problem. Clearly, using a different black-box for integration, e.g., that includes machine learning or complex reasoning, may increase the time per integration and thus the overall time; the optimizations of `Edmint-Opt` would be even more critical in such a case.

5 Related Work

Data integration and related tasks have been extensively studied, and we now overview the work on sub-tasks related to our context.

Source selection Source selection for data integration considers the retrieval and/or selection of most adequate datasets for a given target. One broad line of work that falls under this category, and is perhaps the most related to ours is the search for *joinable/unionable datasets* or links between datasets, typically within an enterprise or a datalake. This includes finding links between tables based on similar attribute content (also termed *domain search*) [22, 34, 36, 35], relation profiles [6, 11], semantic information [15, 14, 33], structural properties [3, 27] and provenance [15, 34]. Most of these works solve a problem different from ours, of finding top-k or even all pair-wise links. The work of [5] considers the join of multiple tables, but their evaluation of the join is based on a concrete usage scenario in feature selection, rather than inherent properties of the integration result like our gain and cost metrics. Some metrics that resemble our gain are discussed in the recent [34] (rate of new rows and columns) but are defined for a single join.

Another related problem is that of source selection for data fusion, which considers the integration of multiple sources for the same data, trying to resolve conflicts [7, 21]. The work of [7] relies on gain and cost notions, but for different metrics suitable for their setting. Data fusion is complementary to our solutions: given means for choosing the best value in case of a conflict and measuring its accuracy, the integration result in our case would have a higher accuracy, which may improve the overall score. Using such methods effectively in our framework may require adapting our different optimizations, and will probably lead to integrations that are more expensive, computationally.

Schema matching A component of our problem, which we consider as a black-box, is Relational schema matching takes two relational database schemata (attributes) as input and returns a mapping between their attributes as output. Previous work has studied means of efficiently and accurately finding the best matching, see e.g., the surveys of [8, 25]. Our work is more related to instance-level matchers, which consider the content of relations for the attribute matching task. This includes, e.g., the work of [1, 4, 23, 24, 28], which use different techniques, mainly based on machine learning but also in inference [1], to learn

mappings between schemata. The uncertainty of matching can be mitigated by different methods, including human feedback obtained via crowdsourcing methods [10], and different similarity metrics accounting for the contents of matched attributes [12, 20, 29, 36].

Record linking and entity resolution Another component of our problem is that of matching tuples, which is related to the areas of *record* and *entity linkage/matching/mapping/resolution/alignment*. Extensive previous work in different fields proposed solutions for linking databases [19, 13, 18, 32], Web tables [2, 16, 26], knowledge bases [30, 31], entities from textual sources [17] and many others. Such solutions may be used as the record linking black-box in our framework, since many of them rely on an underlying notion of similarity or even probability of linking [9, 30], and in turn, this may be used as link weights or value matching probabilities.

6 Conclusion

In this work we have defined the problem of finding, for a given relation, a set of relations such that when integrated together the result optimizes the balance between information gain and cost related to decrease in accuracy and completeness, and the cost of integration itself. We have shown a concrete model and a solution framework that performs well both in terms of the score it achieves, as well as efficiency of computation. Future work includes many opportunities for enhancing this framework with e.g. support of non-relational data; accounting for other aspects of integration such as content relevance, data cleaning and data fusion; and allowing operations (querying) of datasets to change their structure and improve the integration quality.

Acknowledgements

This research was supported by the Israel Science Foundation (grants No. 1157/16 and 2015/21) and by a grant from the Israel Ministry of Science and Technology.

References

[1] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. Designing and refining schema mappings via data examples. In T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors, *SIGMOD*, 2011.

[2] C. S. Bhagavatula, T. Noraset, and D. Downey. Tabel: Entity linking in web tables. In *ISWC*, 2015.

[3] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1), 2009.

[4] C. Chen, B. Golshan, A. Y. Halevy, W. Tan, and A. Doan. BigGorilla: An open-source ecosystem for data preparation and integration. *IEEE Data Eng. Bull.*, 41(2), 2018.

[5] N. Chepurko, R. Marcus, E. Zgraggen, R. C. Fernandez, T. Kraska, and D. Karger. ARDA: automatic relational data augmentation for machine learning. *PVLDB*, 13(9), 2020.

[6] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The Data Civilizer system. In *CIDR*, 2017.

[7] X. L. Dong, B. Saha, and D. Srivastava. Less is more: Selecting sources wisely for integration. *PVLDB*, 6(2), 2012.

[8] X. L. Dong and D. Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2015.

[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1), 2007.

[10] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*, 2014.

[11] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE*, 2018.

[12] R. C. Fernandez, J. Min, D. Nava, and S. Madden. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *ICDE*, 2019.

[13] A. Gruenheid, X. L. Dong, and D. Srivastava. Incremental record linkage. *PVLDB*, 7(9), 2014.

[14] R. Hai, S. Geisler, and C. Quix. Constance: An intelligent data lake system. In *SIGMOD*, 2016.

[15] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing Google's datasets. In *SIGMOD*, 2016.

[16] O. Hassanzadeh, K. Q. Pu, S. H. Yeganeh, R. J. Miller, L. Popa, M. A. Hernández, and H. Ho. Discovering linkage points over web data. *VLDB*, 6(6), 2013.

[17] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa. Low-resource deep entity resolution with transfer and active learning. In *ACL*, 2019.

[18] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12), 2016.

[19] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2), 2010.

[20] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[21] Y. Lin, H. Wang, J. Li, and H. Gao. Data source selection for information integration in big data era. *Inf. Sci.*, 479, 2019.

[22] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *PVLDB*, 11(7), 2018.

[23] H. Nottelmann and U. Straccia. Information retrieval and machine learning for probabilistic schema matching. *Inf. Process. Manage.*, 43(3), 2007.

[24] E. Peukert, J. Eberius, and E. Rahm. A self-configuring schema matching system. In A. Kementsietsidis and M. A. V. Salles, editors, *ICDE*, 2012.

[25] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4), 2001.

[26] E. K. Rezig, E. C. Dragut, M. Ouzzani, and A. K. Elmagarmid. Query-time record linkage and fusion over web databases. In *ICDE*, 2015.

[27] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, 2012.

[28] R. Shraga, A. Gal, and H. Roitman. Adnev: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. *PVLDB*, 13(9), 2020.

[29] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW*, 2015.

[30] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3), 2011.

[31] Z. Sun, W. Hu, Q. Zhang, and Y. Qu. Bootstrapping entity alignment with knowledge graph embedding. In *IJCAI*, 2018.

[32] V. S. Verykios, A. K. Elmagarmid, and E. N. Houstis. Automating the approximate record-matching process. *Inf. Sci.*, 126(1-4), 2000.

[33] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, 2012.

[34] Y. Zhang and Z. G. Ives. Finding related tables in data lakes for interactive data science. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, editors, *SIGMOD*, 2020.

[35] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. JOSIE: overlap set similarity search for finding joinable tables in data lakes. In *SIGMOD*, 2019.

[36] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH ensemble: Internet-scale domain search. *VLDB*, 9(12), 2016.