Fair Knapsack Pricing for Data Marketplaces

Florian Stahl[1,2(✉)] and Gottfried Vossen[1,2]

[1] ERCIS, WWU Münster, Münster, Germany
{flst,vossen}@wi.uni-muenster.de
[2] Waikato Management School, The University of Waikato, Hamilton, New Zealand
{fstahl,vossen}@waikato.ac.nz

Abstract. Data has become an important economic good. This has led to the development of data marketplaces which facilitate trading by bringing data vendors and data consumers together on one platform. Despite the existence of such infrastructures, data vendors struggle to determine the value their offerings have to customers. This paper explores a novel pricing scheme that allows for price discrimination of customers by selling custom-tailored variants of a data product at a price suggested by a customer. To this end, data quality is adjusted to meet a customer's willingness to pay. To balance customer preferences and vendor interest, a model is developed, translating fair pricing into a Multiple-Choice Knapsack Problem and making it amenable to an algorithmic solution.

Keywords: Data pricing · Knapsack · Data marketplaces · Data quality

1 Introduction

Over the last decades, information has become an important production factor which has led to a point at which data, the basic unit in which information is exchanged, is increasingly being traded on data marketplaces [2,12,15]. Data marketplaces are platforms allowing providers and consumers of data and data-related services to interact with each other. A central problem is the determination of a price for data that is considered fair from both the customer and the vendor perspective. We cast this problem into a universal-relation setting, demonstrate the impact of data quality, and propose an algorithmic solution based on the Multiple-Choice Knapsack Problem.

[1] argues that relational *views* can be interpreted as versions of the 'information good' data – an assumption that will also be made here – and identifies three open problems: (1) pricing of data updates; (2) pricing of integrated data for complex value chains; and (3) pricing of competing data sources that provide essentially the same data but in different quality.

The first challenge can be addressed by calculating the difference between the full price of the new and the old product, which is similar to an approach suggested in [23] for buying samples of XML data. The second problem may be addressed by introducing an intermediary pricing for all providers refining

J. Pokorný et al. (Eds.): ADBIS 2016, LNCS 9809, pp. 46–59, 2016.
DOI: 10.1007/978-3-319-44039-2_4

the raw data. This means that the raw data vendor operates using established means; all vendors following in the value chain have to deal with the output price of the lower level vendor as cost and build their prices accordingly.

The last question has been addressed in [21] on which this paper builds, by presenting a quality-centric price pricing model. In particular, we will demonstrate how the quality of relational data products can be adapted to match a buyer's willingness to pay by employing a *Name Your Own Price* (NYOP) model. We thus achieve two things: Providers of data can discriminate customers so that they realize the maximum price a customer is willing to pay, and customers receive a product that is tailored to their own data quality needs and budgets. To start with, providers offer their data at a price P and a given quality. If a customer is willing to accept it, the deal is settled. Otherwise, if a customer wants to pay $W < P$, then the quality is adjusted accordingly. This concept of trading data quality for a discount was previously suggested in [23,24] and applied to both relational as well as XML data; here we focus on relational data only. In contrast to this previous work, which only considered one quality dimension each (*completeness* and *accuracy*), we consider a larger number of quality dimensions (that can easily be extended) and take user preferences into account.

Our setting is that of relational databases [7]. Data marketplaces host data for a number of providers selling relational data with given attributes. For our purposes, this data can be described as a relation r with n unique attributes A_i and domains $dom(A_i), 1 \leq i \leq n$. The set of all attributes is denoted as $X = \{A_1, \ldots, A_n\}$. Consequently, data is described as an instance r of relational schema R with attribute set X. Most of the time, data providers will not sell one relation only, but $m > 1$ relation instances $\{r_1, \ldots, r_m\}$. When distinct provider act on a given marketplace, we assume that each comes with its own set of relations (thus, we do not delve into issues of providing a common schema across providers or related questions).

Since customers will often require data from different relations, which then need to be joined, we make the simplifying assumption that data providers' offerings come as a *universal relation* (u) [14]. We assume that, given $\{r_1, \ldots, r_m\}$, u is created by joining all m relations r_j in such a way that no data is lost, using a full outer join. This has the advantage that no further joins are necessary during the formal elaborations in the remainder of this paper. Furthermore, any original relation r_j may be obtained by appropriate selections and projections over u. Formally, the universal relation u can be defined as $u = r_1 \rtimes \ldots \rtimes r_m$. Notice that this requires attributes to be unique within each single database; however, this can also be achieved by renaming. Since (a subset of) relational algebra is good enough for querying a marketplace in our setting, we can guarantee that the time for collocating data is negligible when calculating prices. Also, we calculate the price based on the resulting view rather than the query itself. Given that users shall receive a relational data product that matches their data quality needs, it is supposed that users know their complete quality preferences and can express them as a total order.

The remainder of this paper is organized as follows: Relevant quality criteria will be described and the notation of utility introduced in Sect. 2. Section 3

will describe how a custom-tailored data product can be created based on a customer's willingness to pay, detailing the calculation of appropriate quality levels as well as the creation of the final data product. The paper is concluded in Sect. 4.

2 Quality-Based Pricing

In [21,22] a total of 21 quality criteria, originally identified as relevant in the context of the Web in [17], have been reviewed regarding their applicability to data marketplaces and specifically to the idea of versioning, i. e., the creation of lower quality versions of a relational data product. This has resulted in seven quality criteria that allow for *continuous* versioning (tailoring) which means that for these criteria an arbitrarily large number of versions can be created automatically. They will be referred to as $\mathsf{V} = \{$*Accuracy, Amount of Data, Availability, Completeness, Latency, Response Time, Timeliness*$\}$. For simplicity, only two measures in V, which are all scaled in the interval $[0, 1]$, will be demonstrated here in detail, namely *Completeness* and *Timeliness*.

Completeness will be interpreted as a *null-freeness* score. To this end, we follow the closed world assumption (CWA) [3], as it is not particularly relevant why a value is missing; fact is it cannot be delivered to the customer. Furthermore, we will suppose that all information necessary to calculate a quality score is available within the data. Thus, quality criteria such as *consistency* that cannot be calculated without knowing the ground cannot be considered in this framework. In contrast, *completeness* or *null-freeness* can be evaluated by measuring the number of cells of a relation to be sold not containing a null-value ($\perp$) compared to the maximum amount of data possible:

$$c(u) = \frac{|\{\mu[A], \mu \in u, A \in X_u | \mu[A] \neq \perp\}|}{|u| \times |X_u|} \tag{1}$$

According to [3], *Timeliness*, i. e., the freshness of data, depends on a number of characteristics, including (a) delivery time, i. e., the time at which the datum is being delivered; (b) input time, i. e., the time at which the datum was entered into the system; (c) age, i. e., the age of the datum when entered into the system; and (d) volatility, i. e., the typical time period a datum keeps its validity. We abstract from age, as it is assumed that time-sensitive data is entered into the system immediately. Furthermore, in most cases it is only relevant when a datum was last updated and how long it remains valid. Adopting the definition of [3], the *Timeliness* of a record or tuple t_μ is a function of delivery time (DT), input time (IT), and volatility (v) defined as:

$$t_\mu(IT, v) = max\left\{0, 1 - \frac{DT - IT}{v}\right\} \tag{2}$$

In order to make *Timeliness* measurable, we assume that a *LastUpdated* attribute and a volatility constant v exist for each view u. Then, the overall timeliness score can be calculated as average timeliness for all tuples in u:

$$tim(u) = \frac{\sum_{\mu \in u} t_\mu(\mu[LastUpdated], v)}{|u|} \tag{3}$$

In addition to the seven criteria that allow for *continuous* versioning, five criteria have been established for which a limited number of versions can be created, i. e., that allow for discrete versioning, collocated in $\mathsf{G} = \{$*Customer Support, Documentation, Security, Representational Conciseness, Representational Consistency*$\}$. From this category, *Customer Support* will serve as an example. For illustration purposes, suppose the following service levels:

1. E-mail support with a 48 h response guarantee;
2. Telephone support (9 to 5) and 24 h response time e-mail support;
3. 24/7 telephone and e-mail support.

To address all quality criteria we introduce $Q = \mathsf{V} \cup \mathsf{G}$. Furthermore, the order of quality criteria will be of importance, hence, from now on, a list of quality criteria q will be used: $q = (q_1, \ldots, q_{n_q})$ with $n_q = |Q|$ elements.

In micro-economics, it is a fundamental assumption that goods provide utility and commonly micro economists investigate utility functions for a number of goods [18]. In contrast, we will here focus on one relational data good and its quality properties. Therefore, the *utility* or *benefit function* will be formalized as $b = f(q_1, \ldots, q_{n_q})$; here q_i denotes the quality scores for the i-th quality criterion. Moreover, it will be supposed that quality criteria are independent, i. e., that the *consumption* of one quality criterion does not effect the utility of other quality criteria. While this is not the case for extremes, e. g., an incomplete data set is less likely to be accurate than a complete one, this is a necessary simplification to handle all dimensions in the following. Two well-known function types commonly serve as utility functions: logarithm functions (first and foremost the natural logarithm) as well as any root function $\sqrt[a]{x}, a \in \mathbb{N}_{\geq 2}$.

We propose to create relational data product versions based on the expected utility. Thus, the utility function is used to create m_l utility-based versions or levels so that $b_j - b_{j-1} = const., 1 < j \leq m_l$. To this end, the quality scores which by definition lie in the interval $[0, 1]$ will be scaled to fit a sector of the utility function's domain $[x_{min}, x_{max}]$, e. g., $[0, 100]$ for the square root. It is worth noting that data with some quality scores beneath a certain threshold t_q are useless. To address this, it is also possible to transform only the interval $[t_q, 1], 0 \leq t_q \leq 1$ from the original score to the representative sector of the utility function, i. e., at a quality score of t_q the utility level of that quality score is 0. To arrive at the necessary minimum quality score for each utility level, the inverted utility function is used, e. g., x^2 for $\sqrt{x}$.

In the following, we will use the square root function as it produces more reasonable utility level intervals. A positive side-effect of using the square root with, for instance, a domain of $[1, 100]$ and $m_l = 10$ utility levels, as done in this paper, is that examples are more illustrative.

Now, the utility-based quality level vector l contains the concrete values of the utility level l_j in order. In the example manifestation presented here, we suppose that $l_j = j, 1 \leq j \leq m_l$.

While this applies for those quality criteria that allow for continuous versioning (i. e., $q \in \mathsf{V}$), for criteria that only allow for discrete versioning (i. e., $q \in \mathsf{G}$) a smaller number has to be chosen. Here, we suggest using three utility levels $l_1 = 3, l_2 = 6, l_3 = 9$ for $q \in \mathsf{G}$ – following Goldilocks principle, discussed in [20], according to which 3 is a good number of versions in the absence of further indicators. To differentiate between the utility level vectors of both sets, they have an according superscript, resulting in the two vectors l^{V} and l^{G}. Since the latter quality levels do not correspond to concrete quality scores, determining a value for them is meaningless. Therefore, the amount of service for each level has to be manually determined, as has been shown for *Customer Support*. Sample figures for both variants are presented in Table 1, where levels for the second type have been marked with an X.

Table 1. Used utility level mapped to versions; showing the required quality score (QS).

Utility level (l_j)	0	1	2	3	4	5	6	7	8	9	10
QS for $q \in \mathsf{V}$	0	1	4	9	16	25	36	49	64	81	100
QS for $q \in \mathsf{G}$	0	$\perp$	$\perp$	X	$\perp$	$\perp$	X	$\perp$	$\perp$	X	$\perp$

While in reality the utility provided by a certain quality level is likely to differ between customers, the general trend is the same and will here be approximated by the same function. Furthermore, we acknowledge that not all quality criteria have the same importance for customers. For example, *Completeness* may be more important for a customer than *Timeliness* because they want to do some time-independent analysis, while for another customer *Timeliness* might be more important because they base time-critical decisions on the data. To represent this in the model, the utility gained from each q_i's quality score is weighted with a user provided ω_i that represents the importance of all quality criteria relative to each other. To this end, users are asked to express their preferences as mentioned earlier. This results in a weight vector $\omega = (\omega_1, \ldots, \omega_{n_q})$ for which $\sum_{i=1}^{n_q} \omega_i = 1$ holds.

A weight matrix B can now be calculated for each user. This matrix shows for which quality criterion q_i with an according weight ω_i what actual utility b_{ij} can be reached for the different utility levels l_j^{V} and l_j^{G}. It is calculated as follows, where the quality levels in l^{G} are normalized:

$$b_{ij} = \begin{cases} \omega_i \times l_j^{\mathsf{V}} \text{ f. a. } q_i \in \mathsf{V} \\ \omega_i \times \frac{l_j^{\mathsf{G}} \times l_{m_l}^{\mathsf{V}}}{l_{m_l}^{\mathsf{G}}} \text{ f. a. } q_i \in \mathsf{G} \end{cases}$$

Inspired by [23,24], this work builds on the idea that providers offer data for an *ask price* P and customers may suggest an alternative (lower) *bid price* W. If $W < P$ the quality of the data is adjusted to meet the price W suggested

by the customer. In contrast to [23,24] we consider an arbitrary number of quality criteria. To this end, besides P providers have to specify the importance of different quality criteria from their point of view. This may either be done based on the cost the different quality criteria cause when being created or based on the perceived utility of the different criteria. As argued in [21], the utility-based approach is preferable; however, the cost-based approach can serve as point of reference if no further information is available. Thus, similar to the user weighting vector ω, providers define a weight vector $\kappa = (\kappa_1, \ldots, \kappa_{n_q})$ for which $\sum_{i=1}^{n_q} \kappa_i = 1$ holds.

For the actual attribution of individual prices to the different quality levels and quality criteria, two fundamentally different approaches can be implemented. In any case the overall price would be distributed to the different quality criteria using κ, with the highest quality level being sold at κP. Then, prices can be attributed to the different quality levels using the utility levels or using the relative satisfaction of each quality criterion. The first will lead to linear prices corresponding to the benefit, which is arguably a fair way of pricing a data product. In this case, the price w_{ij} for each quality criterion q_i at each quality level l_j is calculated using a formula of the form $w_{ij}(P, \kappa, b)$, in detail:

$$w_{ij} = P \times \kappa_i \times \frac{b_{ij}}{b_{i,n_q}}$$

The alternative is to model prices linear to the actual quality scores required to reach this level. This will result in increasing prices for the utility levels. However, looking at it from the discount perspective, this means that the biggest discount is granted for the sacrifice of the first utility level and then decreases. The calculation of w_{ij} is in this case conducted based on the inverted utility function $w_{ij}(P, \kappa, l) = P \times \kappa_i \times b^{-1}(l_j)$ and the overall utility levels in l:

$$w_{ij} = \begin{cases} P \times \kappa_i \times \frac{b^{-1}(l_j^{\mathsf{V}})}{b^{-1}(l_{m_l}^{\mathsf{V}})} \text{ f. a. } q_i \in \mathsf{V}, 1 \leq j \leq l_{m_l}^{\mathsf{V}} \\ \\ P \times \kappa_i \times \frac{b^{-1}(l_j^{\mathsf{G}})}{b^{-1}(l_{m_l}^{\mathsf{G}})} \text{ f. a. } q_i \in \mathsf{G}, 1 \leq j \leq l_{m_l}^{\mathsf{G}} \end{cases}$$

Which of the two is the better alternative cannot be stated per set. There are some quality scores, such as the *amount of data*, for which it is sensible to grant a good discount if less data is to be delivered. In other cases such as *accuracy* it might make more sense to scale prices according to the utility levels. That being said, what model to choose is a business decision that has to be made for each individual criterion depending on the attributes of the criterion as well as on the intended fairness of the pricing model. Given the stronger decrease when using the inverted utility function, the average price across all levels is smaller than in the linear case, which speaks in favor of the latter model from a customer's perspective. After all, it is not important what product is actually delivered as the cost of creating it is marginal. What is more important is that customers get a fair discount for their scarifies of quality. This is achieved by either of them.

3 Fair Knapsack Pricing

The knapsack problem was already studied in 1897 and has been modified in several ways since. One of them is Multiple-Choice Knapsack Problem (*MCKP*) [11] used here. Instead of choosing items from one set of available items, they are chosen from n_q sets, an additional restriction being that from each set exactly one item has to be chosen. Using the variables from the previous sections, pricing can be formalized using the *MCKP*:

$$\text{maximize} \sum_{i=1}^{n_q} \sum_{j=1}^{m_l} b_{ij} a_{ij} \tag{4}$$

$$\text{subject to} \sum_{i=1}^{n_q} \sum_{j=1}^{m_l} w_{ij} a_{ij} \leq W \tag{5}$$

$$\text{and} \sum_{j=1}^{m_l} a_{ij} = 1,\ i = 1, \ldots, n_q \tag{6}$$

$$\text{and } a_{ij} \in \{0; 1\},\ i = 1, \ldots, n_q, j = 1, \ldots, m_l \tag{7}$$

Equations 4 and 5 extend the original knapsack problem to multiple sets to choose from. Equation 6 restricts the choice to one item per set, and Eq. 7 determines that items are indivisible.

In order to create a custom-tailored relational data product, we need to solve a Multiple-Choice Knapsack Pricing Problem (MCKPP). This is non-trivial since already the basic knapsack is $\mathcal{NP}$-complete [8]. *MCKP* is also $\mathcal{NP}$-complete [10], as it can be reduced from the ordinary knapsack problem [11]. Consequently, for a very large input, an exact solution cannot be expected within reasonable time, so that approximations are necessary. Fortunately, *MCKP* can be solved in pseudo-polynomial time using, for instance, dynamic programming or several other algorithms [19]. Most algorithms start by solving the linear *MCKP* to obtain an upper bound. For the linear *MCKP* the restriction $a_{ij} \in \{0; 1\}$ has been relaxed to $a_{ij} \in [0, 1]$, which means it allows choosing a fraction of an item [19].

Algorithm 1 presents a greedy algorithm to solve *MCKPP*. It has been adapted from the one outlined in [11]. The main difference is that the original algorithm contained a preparation step to derive the LP-extremes of each set, which is not necessary for *MCKPP* because of the way in which the matrices are constructed. The algorithm eventually results in a matrix A indicating which items to choose, a value $W - \bar{c}$, which represents the total cost of these items, and a score z, indicating the total utility achieved. Moreover, it calculates the so-called split item a_{st}, i. e., the item that fits only partially into the knapsack, where s indicates the criterion an t the level.

Algorithm 1. Greedy Algorithm to Solve *MCKPP* adapted From [11].

1: # Let i be the index for quality scores and n denote the number of quality scores; j is the utility level index and m denotes the total number of levels.
2: #Initialize:
3: **for** $i = 1 \dots n$ **do**
4: $\quad \bar{c} = W - w_{i1}$ ▷ Residual weight
5: $\quad z = u_{i1}$ ▷ Achieved utility
6: $\quad$ **for** $j = 2; j < m$ **do**
7: $\quad\quad \tilde{b_{ij}} = b_{ij} - b_{i,j-1}$ ▷ Incremental benefit matrix
8: $\quad\quad \tilde{w_{ij}} = w_{ij} - w_{i,j-1}$ ▷ Incremental weight matrix
9: $\quad\quad \tilde{e}_{ij} = \frac{\tilde{u}_{ij}}{\tilde{w}_{ij}}$ ▷ Incremental efficiency matrix
10: $\quad$ **end for**
11: **end for**
12: #Sort:
13: L := sort($\tilde{e}_{ij}$) ▷ List of $\tilde{e}_{ij}$; maintaining original indices
14: #Solve:
15: **for all** $\tilde{e}_{ij}$ in L **do**
16: $\quad$ **if** $\bar{c} - \tilde{w_{ij}} > 0$ **then** ▷ If space left add to knapsack
17: $\quad\quad z \mathrel{+}= \tilde{p_{ij}}$
18: $\quad\quad \bar{c} \mathrel{-}= \tilde{w_{ij}}$
19: $\quad\quad a_{ij} = 1$
20: $\quad\quad a_{i,j-1} = 0$
21: $\quad$ **else** ▷ Split item a_{st} has been found
22: $\quad\quad a_{ts} = \frac{\bar{c}}{\tilde{w_{ts}}}$
23: $\quad\quad a_{t,s-1} = 1 - a_{ts}$
24: $\quad\quad z \mathrel{+}= \tilde{p_{st}}$
25: $\quad\quad$ break loop
26: $\quad$ **end if**
27: **end for**

At this point, we suppose that the number n of quality scores is strictly larger than the number m of quality levels, with $m \leq 10$. As a consequence, only n is relevant while initialising the knapsack. Thus, the overall runtime of Algorithm 1, has a running time of $\mathcal{O}(n \log n)$ owing to the sorting in Line 13. This form of a greedy-type algorithm is often used as a starting point for further procedures such as branch-and-bound [11]. Furthermore, the split solution is generally a good heuristic solution. However, It should be mentioned that the greedy algorithm can perform arbitrarily bad. This means while it operates quickly, there is no guarantee the solution produced is (close to) an optimal solution [11]. Yet, ϵ-approximation algorithms exist provide certain performance guarantees. [9] presents a binary search approximation algorithm running in time $\mathcal{O}(n_t \log n_q)$, where n_q is the number of quality criteria and n_t is the total number of items over all quality criteria $n_t = \sum_{i=1}^{n_q} m_{l_i}$[1]. However, the guarantee is $\epsilon = 0.8$, which is still a considerably bad result even though the authors argue that the actual

[1] m_{l_i} is used to indicate that depending on whether $q_i \in \mathsf{V}$ or $q_i \in \mathsf{G}$, ${m_l}^{\mathsf{G}}$ or ${m_l}^{\mathsf{V}}$ has to be substituted.

performance may be much better than that. Using dynamic programming, a fully polynomial time approximation scheme can be developed [11]. [13] presents an ϵ-approximation that runs in $\mathcal{O}(n_t \log n_t + \frac{n_t n_q}{\epsilon})$, the first term being due to sorting which might be omitted here. [11] presents a similar approach.

Approaches to solve *MCKP* optimally can be found [4–6,19]. Moreover, *MCKP* can commonly be solved quickly in practice [6]. Given that in the *MCKPP* the weights correlate with the benefits per definition, this results in strongly correlated data instances, which are particularly hard for knapsack algorithms, as no dominated items exist [11,19].

Once the appropriate quality levels have been calculated, the data needs to be modified before being delivered. Largely, modifications to the quality can be grouped into three categories:

1. The modification of accompanying services applying to $q \in \mathsf{G}$, e. g., delivery conditions and comprehensiveness of *support*;
2. The modification of the data itself, e. g., decreasing the *completeness*;
3. The modification of the view on the data, e. g., a limited *timeliness*.

We argue that for any of the quality measures used in our framework, an algorithm can be found that creates a quality decreased relational data product according to a proposed discount. For accuracy, this has extensively been described in [24], here, we consider algorithms to modify the *Completeness* as representation of a quality measure that needs modification of the data itself as well as *Timeliness* as representation of a quality criterion that needs modification of a view. For *Customer Support* as representation for quality measures in G, simply the calculated level of service has to be agreed on in a contract.

Obviously, the order in which the quality is decreased is important; for instance, if null values are inserted first and then the accuracy is reduced, the accuracy reduction might build on a wrong distribution. Therefore, we suggest to apply criteria first that reduce the size, then lower the quality of further quality metrics and reduce completeness last.

The first quality measure to be looked at in more detail is *Completeness*, which we have defined as the number of non-null value cells divided by the overall number of cells in Eq. 1. Alternatively and supposing that $n_v = |\{\mu[A], \mu \in u, A \in X_u | \mu[A] = \perp\}|$, this may be written as:

$$c(u) = 1 - \frac{n_v}{|u| \times |X_u|} \tag{8}$$

Now, in order to reduce the completeness further, null values have to be inserted at random. In the following u is the universal relation to be sold before any modification and u^* afterwards. The same applies to other relevant variables, n_v is the number of null values before and $n_{v\text{target}}$ after the quality modification, the suffix indicating a target value. Furthermore, x_{max} denotes the maximum of the domain of the utility function and x the utility score at the chosen level. To lower the completeness the actual value for completeness has to be determined and the target value for completeness has to be calculated based on the selected quality level; consequently the target number of null values $n_{v\text{target}}$ can be calculated:

$$c_t = \frac{x}{x_{max}} \times c(u); \quad \frac{x}{x_{max}} \times c(u) \stackrel{!}{=} 1 - \frac{n_{v\text{target}}}{|u| \times |X_u|} \tag{9}$$

which results in:

$$n_{v\text{target}} = \left\lfloor |u| \times |X_u| \times \left(1 - \frac{x}{x_{max}} c(u)\right) \right\rfloor \tag{10}$$

Note that the floor function has to be used in Eq. 10 to ensure $n_{v\text{target}}$ is an integer, as no half null values exist. Alternatively, the ceiling function could be used, this is at the providers discretion but would result in a slightly worse quality. Based on this target value for null values $n_{v\text{target}}$, a sample method to achieve the modified data set u is described in [21]; it is omitted here for space reasons. We now suppose that u' is a modification of u with null values added.

Timeliness, as defined in Eq. 3, does not require an algorithm as it is concerned with delayed delivery. However, it requires some calculus. In order to further analyse it regarding the quality score, Eq. 2 hast to be plugged in to result in:

$$tim(u) = \frac{\sum_{\mu \in u} max\left\{0, 1 - \frac{DT - \mu[LastUpdated^*]}{v^*}\right\}}{|u|} \tag{11}$$

For better readability $\mu[LastUpdated^*]$ will be denoted as LU. Furthermore, the max function can be omitted supposing that the target score $t_{\text{target}} = \frac{x}{x_{max}}$ is positive. Additionally $|u|$ will be represented by n. Thus:

$$tim(u) = \frac{\sum_{\mu \in u} 1 - \frac{DT - LU}{v^*}}{n} \tag{12}$$

Plugging in a target value t_{target} yields

$$t_{\text{target}} \stackrel{!}{\geq} \frac{\sum_{\mu \in u} 1 - \frac{DT - LU}{v^*}}{n} \quad \Leftrightarrow \quad t_{\text{target}} \times n \times v^* \geq n \times v^* - \sum_{\mu \in u} DT - LU \tag{13}$$

Given that only LU is variable:

$$t_{\text{target}} \times n \times v^* \geq n \times v^* - \left(n \times DT - \sum_{\mu \in u} LU\right) \tag{14}$$

$$\frac{1}{n} \times \sum_{\mu \in u} LU \leq v^* \times (t_{\text{target}} - 1) + DT \tag{15}$$

Equation 15 shows what the average timeliness depending on the target value t_{target} should be and could also be written as:

$$AvgLU(t) \leq v^* \times (t_{\text{target}} - 1) + DT \quad \text{or} \quad LU_{\text{target}} \leq v^* \times (t_{\text{target}} - 1) + DT$$

The delivery time will always be the current time. Thus, it will be represented by the variable now, which will be replaced by the current timestamp upon query time. This allows for further modification to result in

$$LU_{\text{target}} \leq now - v^* \times (1 - t_{\text{target}}).$$

Introducing a delay function:

$$d(v^*, t_{\text{target}}) := v^* \times (1 - t_{\text{target}}) \quad \text{results in} \quad LU_{\text{target}} \leq now - d(v^*, t_{\text{target}})$$

At first sight one might require each data set to have an average timeliness not greater than LU_{target}. However, using the overall average of a data set is slightly problematic, as this allows the selection of data that is very old together with very fresh data and then only use the fresh data. To avoid this, the timeliness of any record is required to be not greater than LU_{target}. In this way it is ensured that records with a timeliness worse than or equal to what has been paid for is delivered. In practical terms customers do query a view u^* on u such that:

$$u^* = \sigma_{\mu[LastUpdated^*] \leq now - d(v, t_{\text{target}})}(u)$$

In this model it is important that when records are updated, the original record is kept so that customers can still access the older record rather than receiving an empty result set. This might seem to complicate matters for providers; from a practical point of view, they will only need to store a number of versions as no customer will complain about getting fresher data than expected.

Finally, addressing the question of pricing competing data sources based on quality, *MCKPP* can be applied to multiple vendors as well. In this case not the scores of one provider have to be mapped to the quality levels but the best scores of all providers have to be used to determine the quality levels. This may results in a scenario where some providers might not be able to deliver all quality criteria at the highest level. Subsequently a *MCKPP* has to be solved for all providers given a customer's query and preference individually. In doing so, it can be determined which provider offers the best product for a customer at the given bid price W.

4 Conclusions and Future Work

In this paper, we have demonstrated a pricing model that allows providers of relational data products to apply a *Name Your Own Price* scheme. This enables them to tap into the willingness-to-pay of customers who would otherwise not buy their (relational) data product. By adjusting the quality it can be ensured

that a customer gets exactly what they pay for, so that a form of fair pricing results. In fact, using this model providers do not have to specify a price publicly at all. They also could use an internal price P and still apply the same pricing model. While this would require users to bid exactly the price they are willing to pay it lacks transparency. An alternative would be advertising a price $P^p > P$ publicly. This would result in additional profits from customers paying a price W for which $P \leq W \leq P^p$ holds.

With developing a quality-based pricing model, it has been shown that pricing on a data marketplace can be expressed as a *MCKP*. The components that influence *MCKPP* are *Quality Criteria*, *Customer Info* comprising the preference vector ω and a bid price W, *Provider Info* comprising a weighting vector κ and an ask price P, a *versioning function* b, a *weighting function* w, and a *Quality Adaptation Algorithm* for each *Quality Criterion*. It is a distinct feature of this model that all components can be adjusted to match the needs of data marketplace providers as well as the needs of data providers. An implementation is an important future work in order to evaluate the algorithm presented in Sect. 3 in the context of pricing. In this regard, the question of whether the linear *MCKPP* might be an alternative for quality criteria in V, as they allow for unlimited versions to be created, is interesting.

Conducting experiments, some work has to be invested into the question of how to actually create the required relational data products on the spot as this might also take a considerable amount of time. For the methods presented in this paper, it can be said that run time is negligible as every record has to be processed at most once, yielding $\mathcal{O}(n)$, where n denotes the number of requested tuples. However, other adaptations might be more difficult.

We have excluded the issue of potential cannibalization from our discussion, i. e., that customers who would have bought expensive products switch to a cheaper version when it becomes available, which is an organizational aspect subject to future research. Furthermore, it should be evaluated whether this pricing model is indeed perceived as fair. To this end, an alternative pricing scheme could be experimented with, in which not all prices are calculated automatically but users are provided with feedback regarding the actual quality levels while entering their prices and preferences. In this case they would know what quality level they receive and can experiment with input variables. This might also increase the perceived fairness. Moreover, truth revelation might be an issue [16]. The question remains if customers can actually cheat the system by not mentioning their true preference. At this point, the argument is that if the algorithm used indeed delivers optimal results, then customers cannot cheat the system as it delivers a custom-tailored product for exactly the suggested price.

References

1. Balazinska, M., Howe, B., Koutris, P., Suciu, D., Upadhyaya, P.: A discussion on pricing relational data. In: Tannen, V., Wong, L., Libkin, L., Fan, W., Tan, W.-C., Fourman, M. (eds.) Buneman festschrift 2013. LNCS, vol. 8000, pp. 167–173. Springer, Heidelberg (2013)
2. Balazinska, M., et al.: Data markets in the cloud: an opportunity for the database community. PVLDB **4**(12), 1482–1485 (2011)
3. Batini, C., et al.: Data Quality: Concepts, Methodologies and Techniques. Data-Centric Systems and Applications. Springer, Heidelberg (2006)
4. Dudziński, K., et al.: Exact methods for the knapsack problem and its generalizations. Eur. J. Oper. Res. **28**(1), 3–21 (1987)
5. Dyer, M., et al.: A branch and bound algorithm for solving the multiple-choice knapsack problem. J. Comput. Appl. Math. **11**(2), 231–249 (1984)
6. Dyer, M., et al.: A hybrid dynamic programming/branch-and-bound algorithm for the multiple-choice knapsack problem. J. Comput. Appl. Math. **58**(1), 43–54 (1995)
7. Garcia-Molina, H., et al.: Database Systems: The Complete Book. Pearson Education Limited, Upper Saddle River (2013)
8. Garey, M.R., et al.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
9. Gens, G., et al.: An approximate binary search algorithm for the multiple-choice knapsack problem. Inf. Process. Lett. **67**(5), 261–265 (1998)
10. Ibaraki, T., et al.: The multiple choice knapsack problem. J. Oper. Res. Soc. Jpn. **21**, 59–94 (1978)
11. Kellerer, H., et al.: Knapsack Problems. Springer, Berlin (2004)
12. Koutris, P., et al.: Toward practical query pricing with QueryMarket. In: SIGMOD Conference, pp. 613–624 (2013)
13. Lawler, E.L.: Fast approximation algorithmsfor knapsack problems. In: 18th Annual Symposium on Foundations of Computer Science, pp. 206–213 (1977)
14. Maier, D., et al.: On the foundations of the universal relation model. ACM TODS **9**(2), 283–308 (1984)
15. Muschalle, A., Stahl, F., Löser, A., Vossen, G.: Pricing approaches for data markets. In: Castellanos, M., Dayal, U., Rundensteiner, E.A. (eds.) BIRTE 2012. LNBIP, vol. 154, pp. 129–144. Springer, Heidelberg (2013)
16. Narahari, Y., et al.: Dynamic pricing models forelectronic business. Sadhana (Acad. Proc. Eng. Sci.) **30**(2 & 3), 231–256 (2005). Indian Academy of Sciences
17. Naumann, F.: Quality-Driven Query Answering for Integrated Information Systems. LNCS, vol. 2261. Springer, Heidelberg (2002)
18. Pindyck, R.S., et al.: Mikroökonomie. 8. überarbeitete Auflage. Pearson Deutschland GmbH, München (2013)
19. Pisinger, D.: A minimal algorithm for the multiple-choice knapsack problem. Eur. J. Oper. Res. **83**(2), 394–410 (1995)
20. Shapiro, C., et al.: Information Rules: A Strategic Guide to the Network Economy. Strategy/Technology/Harvard Business School Press, Boston (1999)
21. Stahl, F.: High-Quality Web Information Provisioning and Quality-Based Data Pricing. Ph.D. thesis. University of Münster (2015)
22. Stahl, F., Vossen, G.: Data quality scores for pricing on data marketplaces. In: Nguyen, N.T., Trawińki, B., Fujita, H., Hong, T.-P. (eds.) ACIIDS 2016. LNCS, vol. 9621, pp. 215–224. Springer, Heidelberg (2016)

23. Tang, R., Amarilli, A., Senellart, P., Bressan, S.: Get a sample for a discount. In: Decker, H., Lhotská, L., Link, S., Spies, M., Wagner, R.R. (eds.) DEXA 2014, Part I. LNCS, vol. 8644, pp. 20–34. Springer, Heidelberg (2014)
24. Tang, R., Shao, D., Bressan, S., Valduriez, P.: What you pay for is what you get. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013, Part II. LNCS, vol. 8056, pp. 395–409. Springer, Heidelberg (2013)