利得の最適連想規則を求める線形時間アルゴリズムの導出

Derivation of Linear Algorithm for Mining Optimized Gain Association Rules

篠埜 功 †* Isao SASANO
胡 振江 † Zhenjiang HU
武市 正人 † Masato TAKEICHI
小川 瑞史 †† Mizuhito OGAWA

† 東京大学大学院 工学系研究科 情報工学専攻
Department of Information Engineering, University of Tokyo
{sasano,hu,takeichi}@ipl.t.u-tokyo.ac.jp

†† 科学技術振興事業団/NTT コミュニケーション科学基礎研究所
Japan Science and Technology Corporation/NTT Communication Science Laboratories
mizuhito@acm.org

Data mining, which is a technology for obtaining useful knowledge from large database, has been gradually recognized as an important subject. Algorithms for data mining have to be efficient, since target database is often huge, and various kinds of efficient algorithms for data mining are individually investigated. Among them, this paper presents that an efficient linear time algorithm for mining optimized gain association rules can be systematically derived from a simple specification by reducing it to an instance of a *maximum marking problem*. Our approach not only automatically guarantees the correctness of the derived algorithm, but also is easy to derive new algorithms for modification of the problem.

1 Introduction

Data mining, which is a technology for obtaining useful knowledge from large database, has been gradually recognized as an important subject. Algorithms for data mining have to be efficient, since target database is often huge. There have been developed many efficient algorithms for various kinds of data mining problems, among which the problem of mining optimized association rules has attracted researchers [2, 3, 4].

To show concretely the problem of mining optimized association rules, we consider the following example. Suppose there is a database recording customers' transactions in a shop, and we are interested in the association rules like the following form:

$$(age \in [a..b]) \Rightarrow BuyRibbon$$

whose confidence exceeds a given threshold θ. There are many rules of the above form by changing a and b. Among them, we would like to find the range of age that maximizes the gain: {the number of customers who bought ribbon whose age are between a and b} minus {the threshold number, *i.e.*, θ times the number of customers whose age is between a and b}. Suppose that the shop makes a profit if 100θ% of customers buy ribbon. Then, the optimized gain range $[a..b]$ is the range of customers that maximizes the shop's profit with respect to the section of ribbon.

This is an example of mining optimized gain association rules problem. This problem is transformed to the problem called *maximum segment sum problem* (MSS for short) [3], and we have a linear time algorithm [1]. Input of the MSS problem is a number list xs, and output is a consecutive sublist of xs that has the maximum sum among all the consecutive sublists of xs. For example, in the case of $xs = [5, -10, 20, -15, 30, -5]$, the result is $[20, -15, 30]$, which has the maximum sum 35.

Rule of the above form may not be satisfactory in some cases. For example, we may hope to find up to k ranges of age for the rule

$$(\bigvee_{i=1}^{k} age \in [a_i..b_i]) \Rightarrow BuyRibbon$$

that maximizes the gain. This problem, which is transformed to a k-MSS problem (Section 2), is

* 日本学術振興会特別研究員 JSPS Research Fellow

more general and more difficult to be solved efficiently. A smart $O(kn)$ algorithm has been proposed in [2], but its correctness is not easy to verify. Furthermore, it is difficult to adapt the algorithm even for a simple modification. For instance, we may want to compute up to k ranges such that the length of each range is between 5 and 10.

In this paper, we show that an efficient linear time algorithm for mining optimized gain association rules is systematically derived from a simple specification by reducing it to a so-called *maximum marking problem*. Our approach not only automatically guarantee the correctness of the derived algorithm, but also is easy to derive new algorithms for modification of the problem.

Throughout this paper, we use notation like the functional programming language Haskell [5].

2 The Problem of Mining Optimized Gain Association Rules

In this section, we show the problem of mining optimized gain association rules [2] can be transformed to the k-MSS problem.

Recall the problem in Section 1. Input database is a set of tuples, each of which holds information about a customer, including an integer value that indicates the age of the customer and a boolean value that indicates whether or not the customer bought ribbon.

From the input database, we would like to mine a rule R_{ribbon} of the following form:

$$(age \in [a..b]) \Rightarrow BuyRibbon.$$

This rule means that a customer of age between a and b often buys ribbon in a confidence no less than θ. There are many rules of the above form by changing a and b. Among them, we would like to find one that maximizes the gain of the rule R_{ribbon}:

$$gain(R_{ribbon}) = sup(age \in [a..b] \wedge BuyRibbon) - \theta \times sup(age \in [a..b]).$$

For a condition C, $sup(C)$ is defined as the number of tuples that satisfy the condition C in the database. Intuitively, this gain shows to what extent the number of tuples establishing the rule R_{ribbon} exceeds what is expected with respect to the least confidence θ.

As an assumption for an efficient algorithm, input tuples are sorted according to customers' age[1]. Then, for each set of tuples that belong to the same age, we compute $g = v - \theta \times u$, where u is the number of customers and v is the number of customers who bought ribbon. As a result, a list gs of values of $g = v - \theta \times u$ is obtained. For example, suppose that record of customers is as in Table 1, which is obtained after the sorting process, with $\theta = 0.2$. For this example, $gs = [5, -10, 20, -15, 30, -5]$. The solution of the MSS problem for the list gs gives the solution of the original problem [3]. In this example, the solution of the MSS problem is $[20, -15, 30]$, so the corresponding range of age [17..19] is the optimized gain range. Generally, the problem of mining optimized gain association rules can be transformed to the MSS problem [3].

In similar way, the optimized gain problem that allows up to k ranges can be transformed to the k-MSS problem. For details, refer to [2, 3].

Here, we formally define the k-MSS problem.

Definition 1 (k-MSS problem) *Input of the k-MSS is a list xs of numbers, and output is up to k consecutive sublists of xs that have the maximum sum among all the up to k consecutive sublists of xs.*

For this problem, $O(kn)$ algorithm was proposed in [2], which will be explained in the next section.

Remark

Since the numeric attribute can have real numbers, input tuples are usually classified into the small number of buckets ordered with respect to the numeric attribute. This process is called bucketing [3, 4]. After the bucketing, find a rule considering only ranges consisting of consecutive buckets. Generally, this means that obtained rules are approximate rules. The bucketing process takes $O(n \log m)$ time where n is the number of tuples in database and m is the number of buckets [4].

[1] The same is assumed also in [2, 3].

Table 1: Record of customers

age of customers	15	16	17	18	19	20
number of customers (u)	200	300	250	400	100	100
number of customers who buy ribbon (v)	45	50	70	65	50	15
$v - \theta \times u$	5	-10	20	-15	30	-5

3 Constructing k-MSS Algorithm Manually

In this section, we explain the algorithm for k-MSS developed in [2]. The algorithm is a k-path algorithm, at the i-th path of which a solution of i-MSS is obtained.

- $i = 1$: At the first path, solve 1-MSS as in [1].
- $i > 1$: Let the solution of the $(i-1)$-MSS be $s_1, s_2, \ldots, s_i$ and the remaining sublists be $t_1, t_2, \ldots, t_j$. Solve 1-MSS for $t_1, t_2, \ldots, t_j$ and let one that has the maximum solution be t_{max}. Solve 1-minimum segment sum problem for $s_1, s_2, \ldots, s_i$ and let one that has the minimum solution be s_{min}. If the segment sum of t_{max} plus the segment sum of s_{min} is less than 0, then split s_{min} into three subintervals with the solution of s_{min} as the middle interval and delete s_{min} from the solution of $(i-1)$-MSS and add the first and third intervals to it, which gives the solution of i-MSS. Otherwise, split t_{max} into three subintervals with the solution of t_{max} as the middle interval and add the solution of t_{max} to the solution of $(i-1)$-MSS, which gives the solution of i-MSS.

This algorithm iterates k times the process of finding the most effective sublist and splitting it, and its complexity is $O(kn)$ [2].

For example, consider 2-MSS problem for input list $[5, -10, 20, -15, 30, -5]$. At the first path, solve 1-MSS. As a result, the sublist $[20, -15, 30]$ is obtained. At the second path, let $s_1 = [20, -15, 30]$, $t_1 = [5, -10]$, and $t_2 = [-5]$. In this case, we split s_1 to $[20], [-15], [30]$ and get the result $[20], [30]$.

This algorithm is smart, but its correctness is not so obvious. In fact, verifying this algorithm needs careful consideration [2]. On the contrary, we will derive $O(kn)$ algorithm from simple specification, and the correctness is automatically guaranteed.

4 Deriving k-MSS Algorithm Automatically

In this section, we derive an $O(kn)$ algorithm for the k-MSS problem by specifying it as a maximum marking problem and applying the theorem proposed in [6].

4.1 Specification

We specify this problem as a maximum marking problem: marking up the elements of a data structure with finite kinds of marks such that the marked elements meet certain property p and has the maximum value with respect to certain weight function *wf*. First, we determine what kind of marks we use. We use the marks 1 and 2, and we attach the mark 1 to the elements that are selected as part of sublists and the mark 2 to the others. Second, we describe property p. Property p checks whether the number of sublists does not exceed the given k. This can be written as follows:

$$
\begin{array}{l}
p\ xs = p'\ xs\ (2,k) \\
p'\ []\ (m,e) = \mathit{True} \\
p'\ (x:xs)\ (m,e) = \\
\quad \textbf{case}\ m\ \textbf{of} \\
\quad\quad 1 \rightarrow \textbf{case}\ \mathit{markKind}\ x\ \textbf{of} \\
\quad\quad\quad\quad 1 \rightarrow p'\ xs\ (1,e) \\
\quad\quad\quad\quad 2 \rightarrow p'\ xs\ (2,e) \\
\quad\quad 2 \rightarrow \textbf{case}\ \mathit{markKind}\ x\ \textbf{of} \\
\quad\quad\quad\quad 1 \rightarrow \textbf{if}\ e > 0\ \textbf{then} \\
\quad\quad\quad\quad\quad\quad p'\ xs\ (1,e-1) \\
\quad\quad\quad\quad\quad\quad \textbf{else}\ \mathit{False} \\
\quad\quad\quad\quad 2 \rightarrow p'\ xs\ (2,e).
\end{array}
$$

The function *markKind* is defined as follows:

$$markKind\ (x, m) = m.$$

Finally, we write the weight function *wf*. The function *wf* can be written as follows.

$$\begin{array}{l} wf\ xs = sum\ (map\ f\ xs) \\ \quad \textbf{where} \\ \quad\quad f\ x = \textbf{case}\ markKind\ x\ \textbf{of} \\ \quad\quad\quad\quad 1 \to w\ x \\ \quad\quad\quad\quad 2 \to 0 \end{array}$$

The function w is defined by $w\ (y, m) = y$.

Now we can describe the problem as a maximum marking problem as follows.

$$mmm\ p\ wf\ 2$$

The function *mmm* generates all the possible 2^n marked lists, and from those which satisfy the property p selects one that has the maximum value with respect to *wf*. For detail, refer to [6].

4.2 Derivation

By the theorem in [6], the derivation of a linear time algorithm is straight forward.

Theorem 1 ([6]) *If p is a finite mutumorphic function and wf is a homomorphic weight function, then (mmm p wf k') can be solved in linear time.*

We omit explanation of the theorem and the process of transforming p and *wf* to the required form, but show only the final result.

$$\begin{array}{l} opt\ 2\ accept\ (f, +, 0)\ \phi_1\ \phi_2\ \delta \\ \quad \textbf{where} \\ \quad\quad accept\ (c, e) = c \wedge e == (2, k) \\ \quad\quad f\ x = \textbf{case}\ markKind\ x\ \textbf{of} \\ \quad\quad\quad\quad 1 \to w\ x \\ \quad\quad\quad\quad 2 \to 0 \\ \quad\quad \phi_1\ (m, e) = True \\ \quad\quad \phi_2\ x\ (m, e)\ r = \\ \quad\quad\quad \textbf{case}\ m\ \textbf{of} \\ \quad\quad\quad\quad 1 \to r \\ \quad\quad\quad\quad 2 \to \textbf{case}\ markKind\ x\ \textbf{of} \\ \quad\quad\quad\quad\quad\quad 1 \to \textbf{if}\ e > 0\ \textbf{then}\ r \\ \quad\quad\quad\quad\quad\quad\quad\quad \textbf{else}\ False \\ \quad\quad\quad\quad\quad\quad 2 \to r \end{array}$$

$$\begin{array}{l} \delta\ x\ (m, e) = \\ \quad \textbf{case}\ m\ \textbf{of} \\ \quad\quad 1 \to \textbf{case}\ markKind\ x\ \textbf{of} \\ \quad\quad\quad\quad 1 \to (1, e) \\ \quad\quad\quad\quad 2 \to (2, e) \\ \quad\quad 2 \to \textbf{case}\ markKind\ x\ \textbf{of} \\ \quad\quad\quad\quad 1 \to (1, e - 1) \\ \quad\quad\quad\quad 2 \to (2, e) \end{array}$$

The definition of the function *opt* is given in Figure 1. The complexity is $O(kn)$ where n is the length of the list. Algorithm obtained by derivation does not need verification, which is an important benefit of deriving efficient algorithm from specification.

4.3 Comparison

Here we compare the algorithm above with the algorithm developed manually in [2] (See Section 3). The derived algorithm is a dynamic programming algorithm and recursive on the input list. In each step it generates $2k$ candidate solutions, where 2 corresponds to whether the current head element is selected or not, and k corresponds to the number of sublists in the current list. So, the derived algorithm performs $O(k)$ operations n times. On the contrary, the algorithm in Section 3 is a greedy algorithm, which generates only one candidate in each step. But in each step, $O(n)$ operations are performed. So, the algorithm in Section 3 performs $O(n)$ operations k times. So, though order of complexity is same, these two algorithms are essentially different.

4.4 Dealing with Change of Specification

Here, consider the modified k-MSS problem with the condition that the length of each sublist must be between 5 and 10. It is not so easy to adapt the algorithm in Section 3 to this modified problem. However, by our method, the only thing we have to do is to change the property p as follows.

$$p\ xs = p'\ xs\ (2, k) \wedge q\ xs$$

The property q checks whether all the selected sublists have length between 5 and 10. Similarly to Section 4.2, we obtain an $O(kn)$ algorithm for the

$$
\begin{array}{l}
opt\ k'\ accept\ (f, \oplus, \iota_\oplus)\ \phi_1\ \phi_2\ \delta\ xs = \\
\quad \textbf{let}\ opts = foldr\ \psi_2\ \psi_1\ xs \\
\quad \textbf{in}\ snd\ (\uparrow_{fst} /\ [\,(w, r^*)\,|\,Just\ (w, r^*) \leftarrow [\,opts!i\,|\,i \leftarrow range\ bnds, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad opts!i \neq Nothing, accept\ i\,]\,]) \\
\quad \textbf{where}\ \psi_1 = array\ bnds\ [\,(i, g\ i)\,|\,i \leftarrow range\ bnds\,] \\
\qquad\qquad \psi_2\ x\ cand = accumArray\ h\ Nothing\ bnds \\
\qquad\qquad\qquad\qquad [\,((\phi_2\ x^*\ e\ c, e),\ (f\ x^* \oplus w,\ x^* : r^*)) \\
\qquad\qquad\qquad\qquad |\ x^* \leftarrow [\,(x, 1), (x, 2), \ldots, (x, k')\,], \\
\qquad\qquad\qquad\qquad\ e \leftarrow acclist, \\
\qquad\qquad\qquad\qquad\ ((c, \_), Just\ (w, r^*)) \leftarrow \\
\qquad\qquad\qquad\qquad\quad [\,(i, cand!i)\,|\,i \leftarrow [\,(c', \delta\ x^*\ e)\,|\,c' \leftarrow classlist], \\
\qquad\qquad\qquad\qquad\qquad\qquad inRange\ bnds\ i, \\
\qquad\qquad\qquad\qquad\qquad\qquad cand!i \neq Nothing\,]\,] \\
\qquad\qquad g\ (c, e) = \textbf{if}\ (c == \phi_1\ e)\ \textbf{then}\ Just\ (\iota_\oplus, []\,)\ \textbf{else}\ Nothing \\
\qquad\qquad h\ (Just\ (w_1, x_1))\ (w_2, x_2) = \textbf{if}\ w_1 > w_2\ \textbf{then}\ Just\ (w_1, x_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else}\ Just\ (w_2, x_2) \\
\qquad\qquad h\ Nothing\ (w, x) = Just\ (w, x) \\
\qquad\qquad bnds = ((head\ classlist, head\ acclist), (last\ classlist, last\ acclist)) \\
acclist = \text{list of all the values in } Acc \\
classlist = \text{list of all the values in } Class
\end{array}
$$

Fig. 1: Optimization function *opt*.

modified problem.

5 Concluding Remarks

In this paper, we show that a linear time algorithm for mining optimized gain association rules is derived from simple specification by reducing it to a *maximum marking problem*. Although a smart $O(kn)$ algorithm is presented in [2], its correctness is not easy to verify. Moreover, the algorithm is fragile to modifications of problems. On the contrary, by our method, we can systematically derive an $O(kn)$ algorithm, and its correctness is automatically guaranteed.

By our previous method in [7], we can also derive a linear time algorithm for the same problems. However, since its specification does not allow accumulating parameters, the constant of the algorithm grows exponentially and the resulting complexity becomes $O(2^k n)$.

Acknowledgments

We would like to thank Shinichi Morishita for introducing us the problems of data mining and CACA seminar members for fruitful discussion.

References

[1] Bentley, J. L.: Programming Pearls: Algorithm Design Techniques, *Communications of the ACM*, Vol. 27,No. 9(1984), pp. 865–871.

[2] Brin, S., Rastogi, R., and Shim, K.: Mining Optimized Gain Rules for Numeric Attributes, *Proc. ACM KDD'99*, 1999, pp. 135–144.

[3] Fukuda, T., Morimoto, Y., Morishita, S., and Tokuyama, T.: Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization, *Proc. ACM SIGMOD'96*, 1996, pp. 13–23.

[4] Fukuda, T., Morimoto, Y., Morishita, S., and Tokuyama, T.: Mining Optimized Association Rules for Numeric Attributes, *Proc. ACM PODS'96*, 1996, pp. 182–191.

[5] Jones, S. P. and Hughes, J.(eds.): *The Haskell 98 Report*, 1999. Available from http://www.haskell.org/definition/.

[6] Sasano, I., Hu, Z., and Takeichi, M.: Generation of Efficient Programs for Solving Maximum Multi-Marking Problems, *SAIG'01*, LNCS 2196, 2001. to appear.

[7] Sasano, I., Hu, Z., Takeichi, M., and Ogawa, M.: Make it Practical: A Generic Linear-Time Algorithm for Solving Maximum-Weightsum Problems, *Proc. ACM ICFP'00*, 2000, pp. 137–149.