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

篠埜 功　胡 振江　武市 正人　小川 瑞史

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. This paper shows 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 the *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.

Derivation of A Linear Algorithm for Mining Optimized Gain Association Rules

Isao Sasano, 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, 日本学術振興会特別研究員, JSPS Research Fellow.

Zhenjiang Hu, 東京大学大学院情報理工学系研究科数理情報学専攻, Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, 科学技術振興事業団 さきがけ研究 21, PRESTO, Japan Science and Technology Corporation.

Masato Takeichi, 東京大学大学院情報理工学系研究科数理情報学専攻, Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo.

Mizuhito Ogawa, 科学技術振興事業団 さきがけ研究 21, PRESTO, Japan Science and Technology Corporation.

コンピュータソフトウェア, Vol.19, No.x (2002), pp.xx–yy.
[小論文] 2001 年 8 月 10 日受付.

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 we focus on the problem of mining optimized association rules [2] [4].

To explain 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:

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

whose confidence exceeds a given threshold θ. There are many rules with the above form by changing a and b. Among them, we would like to find the range of age that maximizes the gain:

$$v - \theta \times u$$

where v denotes the number of customers who bought ribbon and whose age is between a and b, and u denotes 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 $[a..b]$ denotes the age range of customers that maximizes the shop's profit.

This is an example of the problem of mining optimized gain association rules. The essence of the problem is transformed to the problem called *maximum segment sum problem* (MSS for short) [4], for which a linear time algorithm is known [1].

Sometimes we may hope to extend it, say, 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 is transformed to a k-MSS problem (Section 2), which is general and 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 of the problem, such as finding up to k ranges each of which has length between 5 and 10.

In this paper, we show that an efficient linear time algorithm for k-MSS is systematically derived from a simple specification by reducing it to an instance of the *maximum marking problem* [6]. By our approach an efficient linear time algorithm is derived from specification, so the correctness of the derived algorithm is guaranteed. Moreover, we can obtain new algorithms for modification of the problem by only changing the specification and performing similar derivation.

Throughout this paper, we use the functional programming language Haskell [5] to describe our algorithm.

2 Constructing k-MSS Algorithm Manually

Here we give the definition of the k-MSS problem:

Definition 1 (k-MSS problem) Given a list xs of numbers, the k-MSS problem is to find up to k segments of xs whose elements give the maximum sum among all the up to k segments of xs. □

The algorithm for k-MSS developed in [2] is a k-pass algorithm, at the i-th pass of which a solution of i-MSS is obtained.

- $i = 1$: At the first pass, 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 segments 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 segment 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 pass, solve 1-MSS. As a result, the segment $[20, -15, 30]$ is obtained. At the second pass, 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 of $[20], [30]$.

This algorithm is smart, but its correctness is not so obvious. In fact, verifying this algorithm needs careful consideration and takes about four pages [2]. On the contrary, we can derive another $O(kn)$ algorithm from simple specification, and the correctness follows from correctness of derivation steps.

3 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].

3.1 Specification

We specify k-MSS problem as a maximum marking problem: marking up the elements of a data structure with finite kinds of marks ms such that the marked elements meet certain property p and has the maximum value with respect to certain weight function *wf*. The specification can be written as follows:

$$mmp\ wf\ p\ ms\ = \uparrow_{wf} / \circ\ filter\ p\ \circ\ gen\ ms.$$

We use $gen\ ms$ to generate all the possible (finitely many) markings of input data using a set of marks ms (there are $|ms|^n$ possible markings where n is the number of elements in the input data), and from those which satisfy the property p we use $\uparrow_{wf} /$ to select one that has the maximum value with respect to the weight function *wf*. To specify k-MSS problem as a maximum marking problem, we have only to describe ms, wf, and p.

We will use the marks *True* and *False*:

$$ms = [True, False].$$

We attach the mark *True* to the elements that are selected as part of segments and the mark *False* to the others.

Property p checks whether the number of marked segments does not exceed the given k. The property p, which searches the elements in turn counting the number of segments, is defined using accumulating parameter (m, e) as follows:

```
p xs = p' xs (False, k)
p' [] (m, e) = True
p' (x : xs) (m, e) =
  case m of
    True → case markKind x of
             True → p' xs (True, e)
             False → p' xs (False, e)
    False → case markKind x of
             True → if e > 0 then
                       p' xs (True, e − 1)
                    else False
             False → p' xs (False, e).
```

The first accumulating parameter m holds the kind of mark of the previous element, starting with the value of *False*. The second accumulating parameter e holds the number of segments remaining, starting from k. The function *markKind* takes a marked element x as its argument and returns the kind of the mark attached to the element x.

The weight function *wf* can be written as follows:

```
wf xs = sum (map f xs)
  where
    f x = case markKind x of
            True → w x
            False → 0.
```

The function w returns the weight of the element x.

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

$$kmss = mmp\ wf\ p\ [True, False].$$

3.2 Derivation

Our derivation of a linear time algorithm is based on the following theorem [6].

Theorem 1 If p is a finite accumulative property and *wf* is a homomorphic weight function, then $(mmp\ wf\ p\ ms)$ can be solved in linear time. □

Noticing that the property p and weight function *wf* in the k-MSS problem written in the previous section have met the required condition in Theorem 1, we can apply the theorem to obtain the following result:

$$
\begin{aligned}
&kmss = opt\ (f, +, 0)\ accept\ \phi_1\ \phi_2\ \delta\ [True, False] \\
&\quad \textbf{where} \\
&\qquad accept\ (c, e) = c \wedge e == (False, k) \\
&\qquad f\ x = \textbf{case}\ markKind\ x\ \textbf{of} \\
&\qquad\qquad True \rightarrow w\ x \\
&\qquad\qquad False \rightarrow 0 \qquad \phi_1\ (m, e) = True \\
&\qquad \phi_2\ x\ (m, e)\ r = \\
&\qquad\quad \textbf{case}\ m\ \textbf{of} \\
&\qquad\qquad True \rightarrow r \\
&\qquad\qquad False \rightarrow \textbf{case}\ markKind\ x\ \textbf{of} \\
&\qquad\qquad\qquad True \rightarrow \textbf{if}\ e > 0\ \textbf{then}\ r \\
&\qquad\qquad\qquad\qquad \textbf{else}\ False \\
&\qquad\qquad\qquad False \rightarrow r \\
&\qquad \delta\ x\ (m, e) = \\
&\qquad\quad \textbf{case}\ m\ \textbf{of} \\
&\qquad\qquad True \rightarrow \textbf{case}\ markKind\ x\ \textbf{of} \\
&\qquad\qquad\qquad True \rightarrow (True, e) \\
&\qquad\qquad\qquad False \rightarrow (False, e) \\
&\qquad\qquad False \rightarrow \textbf{case}\ markKind\ x\ \textbf{of} \\
&\qquad\qquad\qquad True \rightarrow (True, e - 1) \\
&\qquad\qquad\qquad False \rightarrow (False, e).
\end{aligned}
$$

The definition of the function *opt* is given in Figure 1. Though we will not explain the detail of the algorithm, there are three points worth mentioning here.

1. The complexity of derived algorithm is $O(kn)$ where n is the length of the list. Here we compare the derived algorithm above with the algorithm developed manually in [2] (See Section 2). The algorithm in Section 2 is a greedy algorithm, which generates only one candidate in each step. But in each step, $O(n)$ operations are performed. So, the algorithm performs $O(n)$ operations k times. On the contrary, 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 segments in the current list. So, the derived algorithm performs $O(k)$ operations n times. Although the order of complexity is same, these two algorithms are essentially different.
2. We can derive systematically a linear time algorithm. Note that correctness of the derived algorithm is automatically guaranteed. We have implemented Theorem 1 using an automatic program transformation system MAG [3]. The input to the MAG system is described in Figure 2. By giving the input to the MAG system, the result written using the function `opt` is obtained in a fully automatic way. Currently, MAG system allows only `if` expressions for conditional branches, thus `case` expressions in the specification are converted to `if` expressions. In Figure 2, `mmpRule` corresponds to Theorem 1 and the fusion rule `fusion` for functions with an accumulating parameter is used for representing the property `p` in the required form of `foldrh`.
3. We can derive a linear time algorithm for modification of the problem. Consider the modified k-MSS problem with the condition that the length of each segment must be between 5 and 10. It is not so easy to adapt the algorithm in Section 2 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 segments have length between 5 and 10. Similarly to Section 3.2, we obtain an $O(kn)$ algorithm for the modified problem.

```
opt (f, ⊕, ι⊕) accept φ1 φ2 δ ms xs =
  let opts = foldr ψ2 ψ1 xs
  in snd (↑fst / [(w, r*) | Just (w, r*) ← [opts!i | i ← range bnds,
                                                     opts!i ≠ Nothing, accept i]])
  where ψ1 = array bnds [(i, g i) | i ← range bnds]
        ψ2 x cand = accumArray h Nothing bnds
                      [((φ2 x* e c, e), (f x* ⊕ w, x* : r*))
                      | x* ← mark ms x,
                        e ← acclist,
                        ((c, _), Just (w, r*)) ←
                          [(i, cand!i) | i ← [(c', δ x* e) | c' ← classlist],
                                         inRange bnds i,
                                         cand!i ≠ Nothing]]
        g (c, e) = if (c == φ1 e) then Just (ι⊕, []) else Nothing
        h (Just (w1, x1)) (w2, x2) = if w1 > w2 then Just (w1, x1)
                                     else Just (w2, x2)
        h Nothing (w, x) = Just (w, x)
        bnds = ((head classlist, head acclist), (last classlist, last acclist))
acclist = list of all the values in Acc
classlist = list of all the values in Class
```

Fig.1 Optimization function *opt*.

4 Concluding Remarks

In this paper, we show that a linear time algorithm for solving k-MSS problem is derived from simple specification by reducing it to a *maximum marking problem*. k-MSS problem is the essence of mining optimized gain association rules. A smart $O(kn)$ algorithm is presented in [2], but its correctness is not easy to verify. Actually it takes about four pages to verify the correctness of the algorithm. Moreover, the algorithm is fragile to modifications of problems. On the contrary, by our method, we can systematically derive an $O(kn)$ algorithm not only for the original problem but also for the modified problem, where the correctness of derived algorithms automatically follows from correctness of derivation steps.

Acknowledgments

We 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, *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA USA, ACM Press, August 1999, pp. 135–144.

[3] de Moor, O. and Sittampalam, G.: Generic Program Transformation, *Proceedings of the 3rd International Summer School on Advanced Functional Programming (AFP'98)*, LNCS 1608, Braga, Portugal, Springer-Verlag, September 1998, pp. 116–149.

[4] Fukuda, T., Morimoto, Y., Morishita, S., and Tokuyama, T.: Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, Montreal, Canada, ACM Press, June 1996, pp. 13–23.

[5] Peyton Jones, S. and Hughes, J.(eds.): *The Haskell 98 Report*, February 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, *Semantics, Applications, and Implementation of Program Generation (SAIG'01)*(Taha, W.(ed.)), Lecture Notes in Computer Science, Vol. 2196, Firenze, Italy, Springer-Verlag, September 2001, pp. 72–91.

```
kmss: kmss = mmp wf p [True,False];

wf: wf xs = (foldr (+) 0 (map w xs));
w: w x = if markKind x == True then weight x
         else 0;

p: p xs = p' (foldr (:) [] xs) (False,k);

p'1: p' [] acc = True;
p'2: p' (x:xs) acc = phi x acc (p' xs (delta x acc));

phi: phi x (m,e) r = if m == True then r
                     else if markKind x == True then
                        if e > 0 then r else False
                     else r;

delta: delta x (m,e) = if m == True then
                          if markKind x == True then (True,e)
                          else (False,e)
                       else
                          if markKind x == True then (True,e-1)
                          else (False,e);

mmpRule: mmp wfun p ms
           = opt (fun,oplus,e) (\(c,e) -> c && e==e0) phi1 phi2 delta ms,
         if {wfun = \xs -> foldr oplus e (map fun xs);
             p = \xs -> foldrh (phi1, phi2) delta xs e0};

fusion: f (foldr step e xs) = foldrh (phi1', phi2') delta xs,
        if {
             f e = phi1';
             \ y ys acc -> f (step y ys) acc =
                    \ y ys acc -> phi2' y acc (f ys (delta y acc))
           }
```

Fig.2 Input for the MAG system