A fast MDA approximation algorithm

Staal A. Vinterbo[a,b,c]

[a]*Decision Systems Group, Brigham and Women's Hospital, Boston, USA*
[b]*Harvard Medical School, Boston, USA*
[c]*Harvard-MIT, Division of Health Sciences and Technology, Boston, USA*

DSG Technical Report DSG TR 2004/4
December 2004

Problem Description

The problem of finding a minimum cardinality set among m attributes that preserves discernibility among n points can be formulated as the following "Minimum Discerning Attributes" optimization problem.

Problem 1 *(**MDA**) Let M be an $m \times n$ matrix of natural numbers and let*

$$s(M) = \{H \subseteq \{1, 2, \ldots, m\} | \exists k \in \{1, 2, \ldots, m\} M[i, k] \neq M[j, k] \Rightarrow \exists l \in H M[i, l] \neq M[j, l]\}$$

Find $H \in s(T)$ that minimizes $|H|$.

By the definition of the problem, we don't care about the relative size of the numbers in the matrix. Hence, assume that the entries in T are bounded such that they allow sorting in $O(n)$ time. As an example k-phase Radix sorting can sort n natural numbers in the range $0 \ldots n^k - 1$ in linear time for a constant k.

The algorithm presented here is based on the following. Let I be the indicator function such that $I(P) = 1$ if proposition P is true and $I(P) = 0$ if it is not. Now, define the function $\otimes : \mathbf{N}^m \times \mathbf{N}^m \to \{0, 1\}^m$ as $u \otimes v = w$ where $w[i] = I(u[i] \neq v[i])$. We note that $\otimes$ is identical to the exclusive-or operator on the restriction to $\{0, 1\}^m \times \{0, 1\}^m$.

Let N be the $(n^2 - n)/2 \times m$ binary matrix composed of the $(n^2 - n)/2$ rows of size m given by

$$M[i,] \otimes M[j,] \text{ for } 1 \leq i < j \leq n$$

where $M[i,]$ and $M[j,]$ denote rows i and j of M, respectively.

Proposition 1 *A necessary and sufficient condition on a set H for memership in $s(M)$ is that for any row i in N there must exist a k in H such that $N[i, k] = 1$.*

Proof: (necessity) If $H \in s(M)$, then there exists $k \in H$ such that $M[i, k] \neq M[j, k]$ for any given pair of rows $i \neq j$. Let $v = M[i,] \otimes M[j,]$ be a row in N. By definition of $\otimes$ we have that $v[k]$ is 1 if and only if $M[i, k] \neq M[j, k]$.

(sufficiency) Let $v = M[i,] \otimes M[j,]$ in N, and let $v[k] = 1$. By definition of $\otimes$ we have that $M[i, k] \neq M[j, k]$. ■

The crucial observation is that $H \in s(M)$ is a solution to the Minimum Set Cover problem instance given by the transposed of N, N^t.

The algorithm GREEDY-MDA contains three steps:

1. Transform M into N,
2. transpose N into N^t, and
3. solve the minimum set cover problem instance N^t.

Transposition can clearly be done in linear time. Below we present algorithms for the remaining steps. The total time expenditure is $O(m(n^2 - n)/2)$. Also, as the MSC algorithm presented below is known to guarantee a solution not worse than $1 + \log(m)$ times the optimum for a universe of size m [1], we have that the GREEDY-MDA algorithm guarantees a solution not worse than $1 + \log((n^2 - n)/2)$ times the optimum.

Transforming M to N

The following algorithm takes as input the $n \times m$ instance T of the MDA problem such that all rows in T are unique. The algorithm computes the $(n^2 - n)/2$ length array S of sets $d(i, j) = \{k | t_{ik} \neq t_{jk} \wedge 1 \leq i < j \leq n\}$.

```
LTMDA2MHS($T$)
(1)   for $i = 1$ to $(n^2 - n)/2$
(2)       $S[i] \leftarrow$ NULL
(3)   $q \leftarrow 0$
(4)   for $i = 1$ to $m$
(5)       $(sorted, orgpos) \leftarrow$ sort(column($T, i$))
(6)       $j \leftarrow 1$
(7)       $k \leftarrow 1$
(8)       $incd \leftarrow$ false
(9)       while $j < n$
(10)          while $sorted[j] = sorted[k]$
(11)              $k \leftarrow k + 1$
(12)          for $l = k$ to $n$
(13)              if $incd =$ false
(14)                  $q \leftarrow q + 1$
(15)                  $incd \leftarrow$ true
(16)              $idx \leftarrow$ index($orgpos[j], orgpos[l]$)
(17)              insert($q, S[idx]$)
(18)          $j \leftarrow k$
```

Data structures S is an array of length $(n^2 - n)/2$ and $S[i]$ contains the elements of C_i in a data structure that can be iterated through in $O(|C_i|)$ time (e.g., a linked list).

Functions column(T, i) returns column i of T as an array.

sort(v) sorts array v and returns the sorted array together with an array representing a map from sorted rank to original position in v. As v is an array of natural numbers sort(v) is implemented using radix sorting and has running time $O(|v|)$.

index(i,j) $= (i-1)n+j-i(i-1)/2$ and is essentially a translator from indices of an upper triangular (not including the diagonal) n^2 matrix to indices of an array storing this upper triangular matrix.

Operation Given the set $S = \{1, 2, \ldots, n\}$ of row indices. A column k of T induces a partition $P_k(S)$ of S into equivalence classes consisting of those row indices in S for which the value in column k is the same. The set $d(i, j)$ contains k if and only if i and j are in different equivalence classes in $P_k(S)$. Because $d(i, j)$ is symmetric and $d(i, i) = \emptyset$ we need only compute $d(i, j)$ for $i < j$. The algorithm computes $P_i(S)$ for each column k by sorting, and due to this ordering we can readily identify for which combinations $i < j$ we have that k is in $d(i, j)$.

Running time analysis As sort() runs in $O(n)$ time, the running time of line 5 is $O(nm)$. For column i the loop 9 to 18 is executed as many times as there are different values in column i. Loop 10 to 11 ends up going through the entire column, so all in all mn times. When arriving at line 12, $k > i$ has the index of the next column induced equivalent class (if it exists) "down stream" from the one j is in, meaning that i is a member of all $d(j, k)$ where $j < k \leq n$. Hence lines 13, 16, and 17 are executed $\sum_{i=1}^{m} \sum_{j=1}^{|S|} |S[j] \cap \{i\}|$ times all in all. The test on line 13 is only true once for each time loop 4-18 is executed, hence 14 and 15 are executed $O(m)$ times (actually $|\bigcup_{i=1}^{|S|} S[i]|$ times). It follows that the running time is $O(\sum_{i=1}^{m} \sum_{j=1}^{|S|} |S[j] \cap \{i\}|) = O(\sum_{i=1}^{|S|} |S[i]|)$.

A Fast MSC algorithm

Let $\mathcal{C} = \{C_i\}_{i=1}^{n}$ such that $\{1, 2, \ldots, m\} = \cup_{C_i \in \mathcal{C}} C_i$, and $C_i \neq \emptyset$ for all $1 \leq i \leq n$. The following algorithm implements Johnson's [1] greedy set covering algorithm in $O(\sum_{C_i \in \mathcal{C}} |C_i|)$ time.

Data structures S is an array of length n and $S[i]$ contains the elements of C_i in a data structure that can be iterated through in $O(|C_i|)$ time (e.g., a linked list).

SS is an array of length m. Further let $SS[i]$ contain a doubly linked list such that insertion and removal are $O(1)$ operations. $SS[i]$ will contain the surviving parts of C_i that are of i length.

E is an array of length m, and $E[i]$ contains a list of pointers to elements in SS representing the lists C_i that contain the element i.

r returned by alloc(4) is a pointer to a record of size 4, such that for each C_i, $r \rightarrow [1]$ contains i, $r \rightarrow [2]$ contains the number of elements of C_i still not covered at the current time, $r \rightarrow [3]$ contains the pointer to the previous record in $SS[r \rightarrow [2]]$, and $r \rightarrow [4]$ contains the pointer to the next record in $SS[r \rightarrow [2]]$.

Functions The functions insert(element,list), remove(element,list), first(list) and append(element, list) insert element into the list, remove element from the list, return the first element of the list, and append element to the list respectively in $O(1)$ time.

Operation The algorithm iteratively adds set indices to the solution until the sets indexed by the solution form a cover. Let C_i^j be the surviving part of set C_i before the jth set is added, i.e., if P_j is the solution before adding the jth set's index, then $C_i^j = C_i - \cup_{l \in P_j} C_l$. Let us call this point in the algorithm for iteration j. At iteration j the data structure SS is such that $SS[i]$ contains a list of tuples (l, i) consisting of l's such that $|C_l^j| = i$ paired with i. Let i be the largest such that $SS[i]$ is non-NULL. The first element k of $SS[i]$ is selected, i.e., k is such that $|C_k^j|$ is maximal. Then k is added to the solution, for each set each element in C_i^j is in, the structure SS is updated

```
LTGreedy-MSC($S$)
(1)   /* initialize data structures */
(2)   $C \leftarrow \emptyset$
(3)   for $i = 1$ to $m$
(4)       $Covered[i] \leftarrow$ false
(5)       $SS[i] \leftarrow$ NULL
(6)   for $i = 1$ to $n$
(7)       $r \leftarrow$ alloc(4)
(8)       $r \rightarrow [1] \leftarrow i$
(9)       $size \leftarrow 0$
(10)      foreach $j \in S[i]$
(11)          insert($r$, $E[i]$)
(12)          $size \leftarrow size + 1$
(13)      $r \rightarrow [2] \leftarrow size$
(14)      insert($r$, $SS[size]$)
(15)  /* Compute the greedy cover */
(16)  $l \leftarrow m$
(17)  while true
(18)      while $l \geq 1$ and $SS[l]$ = NULL
(19)          $l \leftarrow l - 1$
(20)      if $l \leq 0$
(21)          return $C$
(22)      $r \leftarrow$ first($SS[l]$)
(23)      $C \leftarrow$ append($r \rightarrow [1]$,$C$)
(24)      foreach $i \in S[r \rightarrow [1]]$
(25)          if not $Covered[i]$
(26)              $Covered[i] \leftarrow$
(27)              foreach $p \in E[i]$
(28)                  if $p \rightarrow [2] > 0$
(29)                      remove($p$, $SS[p \rightarrow [2]]$)
(30)                      $p \rightarrow [2] \leftarrow p \rightarrow [2] - 1$
(31)                      insert($p$, $SS[p \rightarrow [2]]$)
```

to reflect the situation after adding k to the solution. This is done iteratively until all sets have their elements covered.

Running time analysis All tests, function calls and assignments are of $O(1)$ time. The loop line 6-14 is executed once for each set, and the inner loop 10-12 for each element in the set, making the number of times $\sum_{i=1}^{n} |C_i|$.

The loop 17-31 is executed $|C|$ times. Line 19 where the size of the largest survivor is determined is executed maximally m times. The loop starting at line 24 is executed once for each element in the set that has a largest survivor. The test on line 25 is executed once for each element in the set added to the solution, hence $\sum_{i \in C} |C_i|$ times. The loop 27-31 is executed once for each set the element currently being covered (loop starting at 24) is in. As each element is covered only once (test on line 25), this makes the access on line 27 and the statements on lines 28-31 being executed a total of $\sum_{i=1}^{m} \sum_{j=1}^{n} |C_j \cap \{i\}| = \sum_{i=1}^{n} |C_i|$ times.

We can conclude that the algorithm runs in $O(\sum_{i=1}^{n} |C_i|)$ time.

Acknowledgements

This work was supported by grant R01-LM07273 from the National Library of Medicine.

References

[1] Johnson, D. S. (1974) *Journal of Computer and System Sciences* **9**, 256–278.