Improved Algorithm for Maximizing Service of Carousel Storage

Chung-Lun Li
Department of Logistics
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, China
Phone: +852-2766-7410; Fax: +852-2334-1765
Email: *lgtclli@polyu.edu.hk*

Guohua Wan
Faculty of Business Administration
University of Macao
Taipa, Macao, China
Phone: +853-397-4571; Fax: +853-838320
Email: *ghwan@umac.mo*

Abstract

We consider a problem of maximizing service of a carousel storage system from which items are removed in groups, where each group consists of a certain given number of items of each type. Kim [4] has developed an algorithm for solving this problem with a running time of $O(j^2)$. In this article, we present an algorithm with an improved complexity of $O(j \log j)$.

Keywords: Carousel storage; Computational complexity

November 2003
Revised January 2004

1. Introduction

We consider a problem encountered in carousel storage systems that hold cases of items, with an objective of maximizing the number of groups of items that can be retrieved from the system before running out of stock. This problem was originally proposed by Jacob *et al.* [2] and is defined as follows: A storage system can hold N cases of items. The cases are of fixed size, regardless of their contents. There are j different types of items. Within each case, the items are identical. The storage system is to be stocked with full cases, each containing c_i items of type i $(i = 1, 2, \ldots, j)$. Items will be removed from the system in groups, each consisting of n_i items of type i. The objective is to determine the appropriate values for m_i, the number of cases containing items of type i, for $i = 1, 2, \ldots, j$, so as to maximize the number of groups that can be removed from the system. The number of groups of items, g, removable from the system is limited by the availability of each item i. In other words, g must not exceed $c_i m_i / n_i$, where $c_i m_i$ is the total number of items of type i. Thus, the problem is to maximize g subject to the constraints

$$g \leq \frac{c_1 m_1}{n_1},\ g \leq \frac{c_2 m_2}{n_2}, \ldots,\ g \leq \frac{c_j m_j}{n_j},\ \sum_{i=1}^{j} m_i = N,$$

where the decision variables $g, m_1, m_2, \ldots, m_j$ are all integers.

Jacob *et al.* [2] developed a heuristic algorithm for solving this problem. Yeh [6] proposed another heuristic that has a higher accuracy than Jacob *et al.*'s. Jacob *et al.* [3] provided an optimal algorithm for solving the problem in polynomial time. Their algorithm uses binary search to obtain the optimal value of g and has a running time of $O(j \log(\alpha_{\min} N))$, where $\alpha_{\min} = \min_{i=1,\ldots,j}\{c_i m_i / n_i\}$. Kim [4] modified Yeh's heuristic to form an optimal algorithm which runs in $O(j^2)$ time. This strongly polynomial algorithm is significantly more efficient than Jacob *et al.*'s [3] algorithm for problems with a large number of item types. In what follows, we develop a modified version of Kim's algorithm and show that our algorithm has a running time of $O(j \log j)$.

2. The Algorithm

We first present Kim's [4] algorithm. In his algorithm, the solution to the LP relaxation problem is first obtained by setting $m_i = \left[(n_i/c_i)/\sum_{\ell=1}^{j}(n_\ell/c_\ell)\right]N$. It is followed by an iterative procedure to determine the optimal solution of the original problem. A detailed description of the algorithm is given below.

Kim's Algorithm:

Step 0: Set

$$x_i = \left(\frac{n_i/c_i}{\sum_{\ell=1}^{j} n_\ell/c_\ell}\right)N \quad \text{and} \quad y_i = \lceil x_i \rceil \quad \text{for } i = 1, 2, \ldots, j,$$

and set

$$t = \sum_{i=1}^{j} y_i - N.$$

Step 1: If $t = 0$, then stop. Otherwise, compute $h_i = c_i(y_i - 1)/n_i$ $(i = 1, 2, \ldots, j)$ and choose k such that $h_k = \max_{i=1,2,\ldots,j}\{h_i\}$. If more than one such k, break ties arbitrarily.

Step 2: Set $y_k \leftarrow y_k - 1$ and $t \leftarrow t - 1$. Go to Step 1.

The final values of g and m_i $(i = 1, 2, \ldots, j)$ are obtained as

$$g = \left\lfloor \min\left\{\frac{c_1 y_1}{n_1}, \frac{c_2 y_2}{n_2}, \ldots, \frac{c_j y_j}{n_j}\right\}\right\rfloor \quad \text{and} \quad m_i = y_i \quad (i = 1, 2, \ldots, j).$$

Kim [4] proved that the solution generated by this algorithm is always optimal. The running time of this algorithm is $O(j^2)$. Note that in this algorithm, a linear search for h_k (among $h_1, h_2, \ldots, h_j$) is required in every iteration, even when only one of the h_i values is changed. One way to improve the running time of this algorithm is to maintain a sorted list of $h_1, h_2, \ldots, h_j$ so that the maximum value of this list can be obtained quickly. However, once the value of an h_k is changed, we need a method to update the sorted list efficiently. In what follows, we present a modified algorithm with an improved complexity

of $O(j \log j)$. In this modified algorithm, we use a balanced binary tree data structure (also called AVL tree, named after its discoverers G.M. Adel'son-Vel'skii and E.M. Landis [1]) to store the values of $h_1, h_2, \ldots, h_j$ so that h_k can be searched and updated efficiently. A description of the algorithm is given below.

Modified Algorithm:

Step 0: Set

$$x_i = \left(\frac{n_i/c_i}{\sum_{\ell=1}^{j} n_\ell/c_\ell} \right) N \quad \text{and} \quad y_i = \lceil x_i \rceil \quad \text{for } i = 1, 2, \ldots, j,$$

and set

$$t = \sum_{i=1}^{j} y_i - N.$$

If $t = 0$, then stop. Otherwise, go to step 1.

Step 1: Compute $h_i = c_i(y_i - 1)/n_i$ for $i = 1, 2, \ldots, j$, and sort all the h_i values in ascending order. Use a balanced binary tree to store these j sorted values.

Step 2: Choose the largest value in the balanced binary tree, i.e., the rightmost leaf node in the tree. Suppose this largest value is h_k. Then, set $y_k \leftarrow y_k - 1$, $h_k \leftarrow c_k(y_k - 1)/n_k$, and $t \leftarrow t - 1$. If $t = 0$, then stop. Otherwise, insert the new h_k value into the balanced binary tree, and repeat Step 2.

The final values of g and m_i $(i = 1, 2, \ldots, j)$ are obtained in the same way as Kim's algorithm. Note that the only difference between this modified algorithm and Kim's algorithm is that in this algorithm a balanced binary tree data structure is employed to store the h_i values so as to improve the efficiency in the searching and updating the value of h_k in Step 2. Hence, the solution generated by the modified algorithm is the same as that generated by Kim's algorithm, which has been proven to be optimal.

We now provide a brief description of balanced binary trees (see Knuth [5] for details). A binary search tree is a binary tree (with no more than two subtrees at each node) having

a value associated with each node, such that the value at each node is greater than or equal to any value in the left subtree and is less than or equal to any value in the right subtree. On average, searching a value in a binary search tree with n nodes takes $O(\log n)$ time, but in the worst case, it takes $O(n)$ time when the tree degenerates into a linear list. In order to prevent the tree from degenerating into a linear list, we need to balance the tree. For any nonempty binary tree T, we define

$$LeftHeight(T) = \begin{cases} Height(LS(T)) + 1, & \text{if } LS(T) \neq \emptyset; \\ 0, & \text{if } LS(T) = \emptyset; \end{cases}$$

where $Height()$ denotes the height of a tree and $LS(T)$ denotes the left subtree of tree T. $RightHeight(T)$ is defined similarly. Also, for any node v of T, we define $LeftHeight(v)$ as the $LeftHeight$ of its left subtree rooted at v, and we define $RightHeight(v)$ similarly. Thus, a leaf has $LeftHeight$ and $RightHeight$ both equal to 0, and the height of any node is the maximum of its $LeftHeight$ and $RightHeight$. The "balance" of node v is defined as $RightHeight(v)$ minus $LeftHeight(v)$. A binary tree T is balanced if every node has a balance of -1, 0, or 1. A balanced binary tree with n nodes has the following desirable properties [5]:

(i) Its height is $O(\log n)$.

(ii) A node can be added to or deleted from the tree in $O(\log n)$ time, while preserving all properties of a balanced binary tree.

Therefore, if we use a balanced binary tree to store the values of h_i $(i = 1, 2, \ldots, j)$ in Step 2 of the Modified Algorithm, then choosing the largest values of h_i $(i = 1, 2, \ldots, j)$ from and inserting the new h_k value into the balanced binary tree can be done in $O(\log j)$ time.

We now consider the running time of the modified algorithm. Clearly, Steps 0 and 1 take $O(j \log j)$ time. Step 2 is iterated at most $j - 1$ times. In each iteration, searching for the element with the largest value of h_i in the balanced binary tree requires $O(\log j)$

time, removing it from the tree takes $O(1)$ time, and inserting a new value of h_k into the tree requires $O(\log j)$ time. Hence, the overall complexity of the modified algorithm is $O(j \log j)$.

3. Conclusion

We have presented an optimal algorithm for the problem of maximizing service of carousel storage. Our algorithm has a running time of $O(j \log j)$. This low complexity enables us to solve large-sized problems more efficiently than Jacob's [3] and Kim's [4] algorithms.

Acknowledgments

The first author was supported in part by the Areas of Strategic Development Fund of The Hong Kong Polytechnic University. The second author was supported in part by the University of Macao (RG078/02-03S/WGH/FBA).

References

[1] Adel'son-Vel'skii GM and Landis EM. An algorithm for the organization of information. *Soviet Mathematics* 1962;3:1259–63.

[2] Jacobs DP, Peck JC and Davis JS. A simple heuristic for maximizing service of carousel storage. *Computers and Operations Research* 2000;27:1351–6.

[3] Jacobs DP, Peck JC and Davis JS. A fast algorithm for shelf optimization. *Research Journal of Textile and Apparel* 2000;4(2):47–51.

[4] Kim B. Maximizing service of carousel storage. *Computers and Operations Research,* forthcoming.

[5] Knuth DE. *The Art of Computer Programming. Volume 3: Sorting and Searching,* 2nd edition. Addison–Wesley, 1998.

[6] Yeh D-H. A note on "a simple heuristic for maximizing service of carousel storage". *Computers and Operations Research* 2002;29:1605–8.