How to Treat Delete Requests in Semi-Online Problems

Yang Dai\(^1\), Hiroshi Imai\(^2\), Kazuo Iwano\(^3\), and Naoki Katoh\(^1\)

\(^1\) Dept. of Management Science, Kobe University of Commerce, Kobe 651-21, Japan
\(^2\) Dept. of Information Science, University of Tokyo, Tokyo 113, Japan
\(^3\) Tokyo Research Laboratory, IBM Japan, Kanagawa 242, Japan

1 Introduction

We propose a new approach to obtain an semi-online fully dynamic algorithm, given a partially dynamic algorithm, by introducing a new way of handling delete requests. Briefly speaking, we are interested in how online algorithms become more efficient with some partial knowledge of future delete requests. Here, a dynamic algorithm solves the problem for the current instance every time when an add/delete request is made to change the instance. If a dynamic algorithm allows only add requests, we call it partially dynamic, otherwise we call it fully dynamic. An offline algorithm gives a set of solutions of a dynamic algorithm when the entire request sequence is known beforehand. A semi-online problem is a special case of online problems defined as follows: (1) For each update request \( \sigma \) we are given a superset \( SS_\sigma \) of size \( O(k) \) which contains all delete requests in the succeeding \( k \) update requests for any positive integer \( k \) (Notice that we don't have to know the exact sequence of delete requests.); (2) We are also given the total number of requests, \( l \); (3) No information on future add requests is required; (4) After each update request, a solution of the problem so updated is computed. We call this request for a solution a query for an update request. Thus, a semi-online problem has properties both of online and offline problems. As typical semi-online problems, we have offline problems and minimum range problems [13].

Since these online/semi-online/offline dynamic algorithms have practical importance, we can find an extensive list of previous research activities. For example, Frederickson studied online updating of minimum spanning trees [7]. Eppstein et al. considered the maintenance of minimum spanning forest in a dynamic planar graph [5], and Buchsbaum et al. studied the path finding problem when only arc insertions are allowed [2]. Eppstein also devised an offline algorithm for dynamic maintenance of the minimum spanning tree problem [4]. In the design of a fully dynamic algorithm, we often face with the following difficulty: that is, since a delete request may drastically change the basic structure of a problem, we have to rebuild necessary data structures from scratch at each delete request. This is a reason why the design of efficient fully dynamic algorithms is hard.

However, if the partial information on the future deletion requests is available, we can overcome the above difficulty for a certain class of problems including the Subset Sum problem, the connectivity problem, the Integer Knapsack problem,
and the optimization 0-1 Knapsack problem. We introduce a new mechanism which minimizes the total reconstruction work associated with delete requests by dividing a sequence of requests into several phases. At each phase, we create the base data structure which remains the same during the phase and maintain a data structure for items to be deleted or to be newly added in this phase. In this way, we can limit the amount of work for maintaining data structures at each update request. As an instance, we develop an $O(\sqrt{n}d + 1 \cdot IK)$ time algorithm for the semi-online dynamic Subset Sum problem with a target value $K$, when a series of $l$ requests including $#d$ deletions is made to the initially empty set. We also devise semi-online dynamic algorithms for the connectivity problem, Integer Knapsack problem, and optimization 0-1 Knapsack problem which runs in $O(l\sqrt{n}d + K + l \cdot \alpha(lK^{1.5}, K))$ time, and $O((#d+1)^{1/2}lK)$ time, respectively, where $l$ and $#d$ are the numbers of requests and delete requests, $n$ is the number of vertices in the connectivity problem, and $K$ is the target value in the Integer Knapsack problem or the Knapsack capacity in the optimization 0-1 Knapsack problem. Notice that $\alpha(\cdot)$ is the functional inverse of the Ackermann function. To the authors’ knowledge, these bounds are new and nontrivial, and these algorithms are faster than trivial ones.

We have also devised an $O(mnK)$ time algorithm for the $m$ best solutions for the optimization 0-1 Knapsack problem by applying the mechanism of maintaining two data structures at each phase. Notice that the currently best known solution, the naive method, requires $O(mn^2K)$ time [12].

Finally, as an application of the semi-online Subset Sum problem, we consider the problem of finding a cut with the minimum range among all balanced cuts. Here the range of a cut is the maximum difference of edge weights in the cut. As discussed in [3], a minimum range balanced cut algorithm can be used for finding an approximate solution for the minimum balanced cut problem. Since the minimum balanced cut problem, a NP-complete problem [9], has important applications such as the circuit partitioning problem in the VLSI design and has been studied well [6, 11], an approximate solution by using an efficient minimum range balanced cut algorithm would be of broad interest. We then develop an $O(m + n^{2.5})$ time minimum range balanced cut algorithm, which improves an $O(m + n^3)$ time algorithm based on Martello et al.’s general approach to minimum range problems [13].

This paper is organized as follows: In Section 2, we solve the semi-online dynamic Subset Sum problem in $O(\sqrt{n}d + 1 \cdot IK)$ time. In Section 3, we generalize a technique developed in Section 2, and apply for developing semi-online dynamic algorithms for the connectivity problem, the Integer Knapsack problem, and the optimization 0-1 Knapsack problem. We, finally, introduce an $O(m + n^{2.5})$ time minimum range balanced cut algorithm.

2 The semi-online dynamic Subset Sum problem

In this section, we consider the semi-online dynamic Subset Sum problem and introduce a new approach for handling delete requests. Our algorithm takes