Theoretical Computer Science 562 (2015) 227–242

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Dynamic algorithms for monotonic interval scheduling problem [☆]

Alexander Gavruskin [a], Bakhadyr Khoussainov [a], Mikhail Kokho [a,*], Jiamou Liu [b]

[a] Department of Computer Science, University of Auckland, New Zealand
[b] School of Computing and Mathematical Sciences, Auckland University of Technology, New Zealand

A R T I C L E   I N F O

Article history:
Received 25 July 2014
Accepted 23 September 2014
Available online 7 October 2014
Communicated by G. Ausiello

Keywords:
Interval scheduling
Dynamic algorithms
Data structures

A B S T R A C T

We investigate dynamic algorithms for the interval scheduling problem. We focus on the case when the set of intervals is monotonic. This is when no interval properly contains another interval. We provide two data structures for representing the intervals that allow efficient insertion, removal and various query operations. The first dynamic algorithm, based on the data structure called compatibility forest, runs in amortised time $O(\log^2 n)$ for insertion and removal and $O(\log n)$ for query. The second dynamic algorithm, based on the data structure called linearised tree, runs in time $O(\log n)$ for insertion, removal and query. We discuss differences and similarities of these two data structures through theoretical and experimental results.

1. Introduction

Imagine a number of processes that need to use a particular resource for a period of time. Each process i specifies a starting time $s(i)$ and a finishing time $f(i)$ between which it needs to continuously occupy the resource. The resource cannot be shared by two processes at any instance. One is required to design a scheduler which chooses a subset of these processes so that 1) there is no time conflict between processes in using the resource, and 2) there are as many processes as possible that get chosen.

The above is a typical set-up for the interval scheduling problem, one of the basic problems in the study of algorithms. Formally, given a collection of intervals on the real line all specified by starting and finishing times, the problem asks for a subset of maximal size consisting of pairwise non-overlapping intervals. The interval scheduling problem and its variants appear in a wide range of areas in computer science and applications such as in logistics, telecommunication, and manufacturing. They form an important class of scheduling problems and have been studied under various names and with application-specific constraints [11].

The interval scheduling problem, as stated above, can be solved by a *greedy* scheduler as follows [10]. The scheduler sorts intervals based on their finishing time, and then iteratively selects the interval with the least finishing time that is compatible with the intervals that have already been scheduled. The set of intervals chosen in this manner is guaranteed to

☆ The research is partially supported by Marsden Fund of the Royal Society of New Zealand, project number 10-UOA-165. A preliminary version of this paper appeared as [5].

* Corresponding author. Tel.: +64 9 373 7599x88260.
E-mail addresses: a.gavruskin@auckland.ac.nz (A. Gavruskin), bmk@cs.auckland.ac.nz (B. Khoussainov), m.kokho@auckland.ac.nz (M. Kokho), jiamou.liu@aut.ac.nz (J. Liu).

http://dx.doi.org/10.1016/j.tcs.2014.09.046
0304-3975/© 2014 Elsevier B.V. All rights reserved.

have maximal size. This algorithm works in a *static* context in the sense that the set of intervals is given a priori and it is not subject to change.

In a *dynamic* context the instance of the interval scheduling problem is usually changed by a real-time events, and a previously optimal schedule may become not optimal. Examples of such real-time events include job cancellation, arrival of an urgent job, change in job processing time. To avoid the repetitive work of rerunning the static algorithm every time when the problem instance has changed, there is a demand for efficient *dynamic algorithms* for solving the scheduling problem on the changed instances. In this dynamic context, the set of intervals changes through a number of *update operations* such as insertion or removal. Our goal is to design data structures that allow us to solve the interval scheduling problem in a dynamic setting. The results of this paper have been partially exposed in [5].

In our effort to dynamise the interval scheduling problem, we focus on a special class of interval sets which we call *monotonic interval sets*. In monotonic interval sets no interval is properly contained by another interval. Considering monotonic intervals is a natural setting for the problem. For example, if all processes require the same amount of time to be completed, then the set of intervals is monotonic. Moreover, monotonic interval sets are closely related to proper interval graphs. An *interval graph* is an undirected graph whose nodes are intervals and two nodes are adjacent if the two corresponding intervals overlap. A *proper interval graph* is an interval graph for a monotonic set of intervals. There exist linear time algorithms for representing a proper interval graph by a monotonic set of intervals [1,8,3]. Furthermore, solving the interval scheduling problem for monotonic intervals corresponds to finding a maximal independent set in a proper interval graph.

Related work. On a somewhat related work, S. Fung, C. Poon and F. Zheng [4] investigated an online version of interval scheduling problem for weighted intervals with equal length (hence, the intervals are monotonic), and designed randomised algorithms. We also mention that R. Lipton and A. Tompkins [7] initiated the study of online version of the interval scheduling problem. In this version a set of intervals are presented to a scheduler in order of start time. Upon seeing each interval the algorithm must decide whether to include the interval into the schedule.

A related problem on a set of intervals I asks to find a minimal set of points S such that every interval from I intersects with at least one point from S. Such a set S is called a *piercing set* of I. A dynamic algorithm for maintaining a minimal piercing set S is studied in [6]. The dynamic algorithm runs in time $O(|S|\log|I|)$. We remark here that if one has a maximal set J of disjoint intervals in I, one can use J to find a minimal piercing set of I, where each point in the piercing set corresponds to the finishing time of an interval in J in time $O(|J|)$. Therefore our dynamic algorithm can be adapted to one that maintains a minimal piercing set. Our algorithm improves the results in [6] when the interval set I is monotonic.

Kaplan et al. in [9] studied a problem of maintaining a set of nested intervals with priorities. The problem asks for an algorithm that given a point p finds the interval with maximal priority containing p. Similarly to our dynamic algorithm, the solution in [9] also uses dynamic trees to represent a set of intervals.

Our results. We provide two dynamic algorithms for solving the interval scheduling problem on monotonic set of intervals. Both algorithms allow efficient insertion, removal and query operation. Formal explanation is in the next sections.

The first algorithm maintains the *compatibility forest* data structure and is denoted by CF. We call the *right compatible interval* of a given interval i the interval j such that $f(i) < s(j)$ and there does not exist an interval ℓ such that $f(i) < s(\ell)$ and $f(\ell) < f(j)$. The CF data structure maintains the right compatible interval relation. The implementation of the data structure utilises, nontrivially, the dynamic tree data structure of Sleator and Tarjan [12]. As a result, in Theorem 6 of Section 3 we prove that the insert and remove operations take amortised time $O(\log^2 n)$ and the query operation takes amortised time $O(\log n)$.

The second dynamic algorithm maintains the *linearised tree* data structure and is denoted by LT. We say that intervals are *equivalent* if their right compatible intervals coincide. The LT data structure maintains both the right compatibility relation and the equivalence relation. Then, in Theorem 15 of Section 4 we prove that the insertion, removal and query operations take time amortised $O(\log n)$. However, this comes with a cost. As opposed to the CF data structure that keeps a representation of an optimal set after each update operation, the linearised tree data structure does not explicitly represent the optimal solution.

To test the performance of our algorithms, we carried out experiments on random sequences of update and query operations. The experiments show that the two data structures CF and LT perform similarly. The reason for this is that the first dynamic algorithm based on CF reaches the bound of $\log^2 n$ only on specific sequences of operations, while on uniformly random sequences the algorithm may run much faster. Both algorithms outperform the modified naive algorithm (described in Section 2).

Organisation of the paper. Section 2 introduces the problem, monotonic interval sets and the modified naive dynamic algorithm. Sections 3 and 4 describe the CF and LT data structures and present our dynamic algorithms. Section 5 extends the data structures by adding the report operation that outputs the full greedy solution. Section 6 discusses the experiments.

2. Preliminaries

An *interval* is a pair $(s(i), f(i)) \in \mathbb{R}^2$ with $s(i) < f(i)$, where $s(i)$ is the *starting time* and $f(i)$ is the *finishing time* of the interval. We abuse notation and write i for the interval $(s(i), f(i))$. We say that an interval x *contains* an interval y if $s(x) < s(y)$ and $f(x) > f(y)$. Sometimes we look at an interval i as a set of real numbers between $s(i)$ and $f(i)$. Then we talk about the left and the right endpoints of an interval and denote x contains y as $x \supset y$.