On the list update problem

Author(s):
Ambühl, Christoph

Publication Date:
2002

Permanent Link:

https://doi.org/10.3929/ethz-a-004394113 →

DISS. ETH No. 14529, 2002

On the List Update Problem

A dissertation submitted to the
Swiss Federal Institute of Technology, ETH Zürich
for the degree of Doctor of Technical Sciences

presented by
Christoph Ambühl
Dipl. Informatik-Ing. ETH
born 15.9.1973, citizen of Willisau-Stadt, Switzerland

accepted on the recommendation of
Prof. Dr. Emo Welzl, ETH Zürich, examiner
Prof. Dr. Susanne Albers, Universität Freiburg i.B., co–examiner
Dr. habil. Bernhard von Stengel, London School of Economics, co–examiner

Abstract

An unsorted linear list is one of the simplest data structures on which one can perform insertions, deletions and lookups. To perform a lookup, the list has to be traversed linearly until the requested item is found. The performance of this data structure can be enhanced by making it self-organizing. In general, the most recently requested item will be moved closer to the front of the list. This is motivated by the empirical observation that, in many cases, requests to items are clustered over time.

An algorithm that updates the list based on the current and past requests is called a list update algorithm. These algorithms are called *online* since they do not know what the forthcoming requests will be.

A very simple algorithm is called `MOVE TO FRONT (MTF)`. Here, the most recently requested item is moved to the front just after the lookup. In 1985, Sleator and Tarjan proved 2-competitiveness of `MTF` which is defined as follows: For any sequence of requests, the running time of `MTF` is at most twice the running time of the optimal *offline* algorithm `OPT`. Compared to online algorithms, offline algorithms are therefore more powerful since they know the whole request sequence in advance.

Using randomized techniques, one can find algorithms which are even more competitive. The best algorithm known to date is the 1.6-competitive `COMB` algorithm due to Albers, von Stengel, and Werchner. It is known that no algorithm can be better than 1.5-competitive.

The ultimate goal is, of course, to find the optimally competitive list update algorithm. All results in this thesis are aimed to give more insight into the structure of the list update problem.

The first result shows that, in the partial cost model, no algorithm can be better than 1.50115-competitive. This is the first non-trivial lower bound in this model. The partial cost model is much easier to analyze. Furthermore, any c-competitive algorithm in the partial cost model is also c-competitive in the standard model.

The second result gives a characterization of all projective algorithms.

They are basically the only kind of algorithms which can be analyzed so far. To prove that a projective algorithm is c-competitive, one only has to prove this on lists with two items which is, of course, much easier. Using this characterization, we give a matching lower bound for projective algorithms in the partial cost model.

The third result shows that it is $\mathcal{NP}$-hard to compute OPT. Hence, there is probably no efficient implementation of OPT. Furthermore, there is only little hope that a better understanding of OPT might give new insights into the list update problem.

Zusammenfassung

Unsortierte lineare Listen gehören zu den einfachsten Datenstrukturen, auf denen sich die Operationen Einfügen, Löschen und Suchen ausführen lassen. Um ein Element in der Liste zu finden muss diese linear durchsucht werden, bis man auf das gesuchte Element stösst. Die Effizienz kann gesteigert werden, indem man die Elemente in der Liste immer wieder umordnet. Man spricht von *selbstorganisierenden Datenstrukturen*. Im Allgemeinen wird das gerade angefragte Element in der Liste weiter nach vorne verschoben, da in vielen praktischen Anwendungen die Anfragen auf die Elemente zeitlich gehäuft sind.

Wir bezeichnen einen Algorithmus, welcher die Liste umordnet, als *list update Algorithmus*. Es handelt sich hier um *online* Algorithmen, da der Algorithmus die zukünftigen Anfragen an die Datenstruktur nicht kennt.

`MOVE TO FRONT (MTF)` gehört zu den einfachsten Algorithmen. Bei jeder Anfrage wird das angefragte Element an den Anfang der Liste verschoben. Sleator und Tarjan zeigten 1985, dass `MTF` 2-kompetitiv ist. Dies bedeutet, dass für jede Sequenz von Operationen die Laufzeit von `MTF` höchstens doppelt so lang ist wie die Laufzeit des optimalen *offline* Algorithmus `OPT`. Im Gegensatz zu online Algorithmen kennen offline Algorithmen die gesamte Sequenz von Beginn weg und sind damit sogar mächtiger als online Algorithmen.

Der beste bekannte Algorithmus ist der 1.6-kompetitive `COMB` (Albers, von Stengel, Werchner). Weiter ist bekannt, dass kein Algorithmus besser als 1.5-kompetitiv sein kann. Das Hauptziel des list update problems besteht darin, den besten Algorithmus zu finden.

Das erste Resultat der Arbeit zeigt, dass kein Algorithmus im $i-1$ Modell besser als 1.50115-kompetitiv sein kann. Dies ist die erste nicht triviale untere Schranke in diesem Modell. Im $i-1$ Modell kostet ein Zugriff auf das $i$te Elemement der Liste nur $i-1$ Zeiteinheiten, während im standard Modell dafür i Einheiten bezahlt werden müssen. Im $i-1$ Modell lassen sich Algorithmen einfacher analysieren und die Analysen lassen sich dann direkt auf das standard Modell übertragen.

Als zweites Resultat werden alle projektiven Algorithmen charakterisiert. Bis auf eine Ausnahme gehören alle bis jetzt analysierten Algorithmen zu dieser Klasse. Um zu zeigen, dass ein projektiver Algorithmus c-kompetitiv ist, reicht es zu zeigen, dass er dies auf Listen mit nur zwei Elementen ist. Dies ist natürlich viel einfacher als im allgemeinen Fall. Mit Hilfe dieser Charakterisierung wird sodann eine untere Schranke von 1.6 für die Kompetitivität von projektiven Algorithmen gezeigt. `COMB` ist also ein optimaler projektiver Algorithmus.

Im letzten Kapitel wird gezeigt, es die Berechnung der optimalen offline Kosten ein $\mathcal{NP}$-hartes Problem darstellt. Dies bedeutet, dass wahrscheinlich kein effizienter Algorithmus dafür exisitert. Es besteht damit wenig Hoffnung, dass ein besseres Verständnis von `OPT` zu effizienteren online Algorithmen führen würde.