Theoretical Computer Science 607 (2015) 113–123

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Approximation of function evaluation over sequence arguments via specialized data structures

Tamal T. Biswas [*], Kenneth W. Regan [*]

Department of CSE, University at Buffalo, Amherst, NY 14260, USA

ARTICLE INFO

Article history:
Received 1 October 2014
Received in revised form 6 March 2015
Accepted 28 March 2015
Available online 8 April 2015

Keywords:
Data structures
Memoization
Sequences
Metrics
Topology
Machine learning
Cloud computing

ABSTRACT

This paper proposes strategies for maintaining a database of computational results of functions f on sequence arguments $\vec{x}$, where $\vec{x}$ is sorted in non-decreasing order and $f(\vec{x})$ has greatest dependence on the first few terms of $\vec{x}$. This scenario applies also to symmetric functions f, where the partial derivatives approach zero as the corresponding component value increases. The goal is to pre-compute exact values $f(\vec{u})$ on a tight enough net of sequence arguments, so that given any other sequence $\vec{x}$, a neighboring sequence $\vec{u}$ in the net giving a close approximation can be efficiently found. Our scheme avoids pre-computing the more-numerous partial-derivative values. It employs a new data structure that combines ideas of a trie and an array implementation of a heap, representing grid values compactly in the array, yet still allowing access by a single index lookup rather than pointer jumping. We demonstrate good size/approximation performance in a natural application.

1. Introduction

In many computational tasks, we need to evaluate a function on many different arguments, in applications such as aggregation where we can tolerate approximation. Evaluations $f(x)$ may be expensive enough to demand *memoization* of pre-computed values, creating a fine enough grid of argument-value pairs to enable approximating $f(x)$ via one or more neighboring pairs $(u, f(u))$. In this paper we limit ourselves to external memoization, here meaning building the complete grid in advance. Such applications of grids in high-dimensional real spaces $\mathbb{R}^\ell$ are well known (see history in [2]). What distinguishes this paper is a different kind of space in which the arguments are homogeneous *sequences* not just arbitrary vectors, where f and the space obey certain large-scale structural properties.

To explain our setting and ideas, consider first the natural grid strategy in $\mathbb{R}^\ell$ of employing the Taylor expansion to approximate $f(x)$ via a nearby gridpoint u:

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

If we make the grid fine enough, and assume that f is reasonably smooth, we can ignore the terms with second and higher partials, since $(x_i - u_i)(x_j - u_j)$ is quadratically small in the grid size. Doing this still requires knowledge of the gradient of f

* Corresponding authors.
E-mail addresses: tamaltan@buffalo.edu (T.T. Biswas), regan@buffalo.edu (K.W. Regan).

http://dx.doi.org/10.1016/j.tcs.2015.03.049
0304-3975/© 2015 Elsevier B.V. All rights reserved.

on the grid-points, however. Memoizing—that is, precomputing and storing—all the partials on the gridpoints might be ℓ times as expensive as memoizing the values $f(u)$. Hence, depending on the application, one may employ an approximation to the gradient or a recursive estimation of the partials.

In our setting, we are given a different kind of structure with a little more knowledge. Here we need to compute functions f on sequence arguments $\vec{x} = (x_1, x_2, x_3, \ldots)$ under the following circumstances.

(a) The sequence entries and function values belong to $[0, 1]$.
(b) For all $i < j$ and $\vec{x}$, $\partial f/\partial x_i > \partial f/\partial x_j$ at $\vec{x}$.
(c) The sequences are non-decreasing, and for all i, $\partial f/\partial x_i$ becomes small as x_i approaches 1.
(d) While exact computation of $f(\vec{x})$ is expensive, moderate precision suffices, especially when there is no bias in the approximations.

Part (b) says that the initial terms have the highest influence on the result, while (c) together with (b) implies that as the sequence approaches its ceiling, terms lose their influence. Part (c) also allows us to assume all sequences have the same length ℓ, using 1.0 values as trailing padding if needed. Applications obeying (d) include calculation of means and percentiles and other aggregate statistics, as are typical for streaming algorithms [7], and various tasks in curve fitting, machine learning [6], complex function evaluation, and Monte Carlo simulations (see [5]).

For further intuition, note that this setting applies to any function f that is symmetric, that is whose value is independent of the order of the arguments. Such a function really depends on the values of the arguments in a ranking structure. We may without loss of generality restrict the arguments to be sequenced by rank. Then (b) says that the first-ranked arguments matter most, while (c) says that elements with not only poor rank but also poor underlying scores have negligible marginal influence.

The goal is to build a data structure U of arguments and values $f(\vec{u})$ with these properties:

1. Coding: For all argument sequences $\vec{x}$ there are $\vec{u} \in U$ such that $|f(\vec{u}) - f(\vec{x})| < \epsilon$.
2. Size: $|U|$ is not too large, as a function of ϵ.
3. Efficiency: A good neighbor or small set of neighbors $\vec{u}$ can be found in time proportional to the length of the sequence, with only $O(1)$ further computation needed to retrieve the value $f(\vec{u})$.
4. Only a "black box" memo table of values $f(\vec{u})$ is needed, with the remainder of the approximation algorithm staying (essentially) independent of f.

Our main contribution is the construction of a family $\mathcal{G}$ of grid structures U and a simple memoizing algorithm A that is parameterized by a weighting function $wt(\cdots)$ used in lieu of the gradient. Among functions we consider are $wt(\cdots) = 0$, which means ignoring the gradient, and

$$wt(i, \vec{x}, -) = \frac{1}{i}(1 - x_i). \tag{1}$$

The intuition for this is that the first i elements have size no larger than x_i, which is to say equal or higher rank and influence on $f(\vec{x})$ to x_i. If they were all equal, then each would have a share $1/i$ of their total influence. The multiplier $(1 - x_i)$ aims to moderate the influence as the argument component x_i itself increases. It is also the simplest multiplier that goes to zero as x_i goes to 1. Our algorithm A uses weights in a balancing strategy that reflects the other sequence elements x_j for $j < i$.

We report success on a fairly general range of functions f that arise from a natural application in which probabilities are estimated. Some of the f represent salient basic mathematical problems in their own right. Key features of our data structure, algorithm, and overall strategy are:

- The grid is not regular but "warped": it starts fine but becomes coarse as i increases, eventually padding with nonce 1 values.
- The grid has an efficient compact mapping to an array, like a "warped" array implementation of a heap, so that values $f(\vec{u})$ can be looked up by one index rather than pointer jumping.
- The actual gradient of f is replaced by the universal substitute (1) and various other weighting method described later, which reflects properties (b) and (c) above.
- The algorithm to find a good grid neighbor $\vec{u}$ to the given argument $\vec{x}$ uses weights $w_i = wt(i, \cdots)$ to balance rounding.

This seems like a "cookbook" approach, but our point is that we have more structure to work with, before the steps of the recipe that have f as a particular ingredient need to be acted on. In our application there is no connection between the weights w_i and the functions f except for the axiomatic properties (b) and (c) and lack of unusual pathology in f. We demonstrate performance that is almost ten times faster than without memoization, and with four-place accuracy from a grid that starts with only two-place fineness. First we describe the application.