Operations Research Letters 44 (2016) 165–169

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

Coupled bisection for root ordering

Stephen N. Pallone *, Peter I. Frazier, Shane G. Henderson

School of Operations Research and Information Engineering, Cornell University, Ithaca, NY 14853, United States

ARTICLE INFO

Article history:
Received 19 February 2015
Received in revised form
27 October 2015
Accepted 23 December 2015
Available online 31 December 2015

Keywords:
Bisection
Dynamic programming
Lagrangian relaxation
Index policies

ABSTRACT

We consider the problem of solving multiple "coupled" root-finding problems at once, in that we can evaluate every function at the same point simultaneously. Using a dynamic programming formulation, we show that a sequential bisection algorithm is a close-to-optimal method for finding a ranking with respect to the zeros of these functions. We show the ranking can be found in linear time, prove an asymptotic approximation guarantee of 1.44, and conjecture that this policy is near-optimal.

1. Introduction

Consider a function $f : \mathcal{S} \times [0, 1) \to \mathbb{R}$, where $\mathcal{S}$ is finite but large, and for every s, $f(s, \cdot)$ is monotonic with unique root. If we are interested in finding the zero $x^*(s)$ of $f(s, \cdot)$ for all elements $s \in \mathcal{S}$, then for each $f(s, \cdot)$ we could employ the classical bisection method. However, if one evaluation of f for some x yields values of $f(s, x)$ for all $s \in \mathcal{S}$, then we could potentially solve multiple bisection problems at once. Furthermore, if we are only interested in the *ordering* of elements s with respect to their zeros $x^*(s)$, calculating the zeros to precision is computationally unnecessary.

The coupled root-ordering setting has applications in computing Gittins [3] and Whittle [12] index policies, respectively used in multi-arm bandit and restless bandit problems. We are then interested in ordering states in the state space according to their Gittins or Whittle indices, which correspond to the zero of a particular function. The ordering of the states is all that is required to implement the index policy. Methods for evaluating these indices to precision are prevalent in the literature (see [6] for a discussion on computational methods). In practical applications where Gittins indices are computed, the problems typically have additional structure, such as sparse transition kernels or the ability to compute indices in an online fashion. The most competitive algorithms for computing Gittins index policies exploit these kinds of structure. Coupled bisection does not take advantage of any additional structure, and therefore is not competitive with these algorithms. However, its generality allows it to compute index policies for a wide range of problems that be formulated as instances of coupled root-ordering [5,13,2,4].

Coupled bisection can also be used in solving coupled root-finding problems, because it can be more computationally efficient to sort the roots before further refining their respective locations. One such example is the estimation of phase diagrams during the evaluation of piezoelectric materials [10]. Given a pair of chemical compounds A and B, one must determine for each temperature s in some set a critical threshold $x^*(s)$ such that mixtures of A and B with a fraction $x > x^*(s)$ of A form one crystal phase, while mixtures with $x < x^*(s)$ form another phase. Here, $f(s, x)$ is the crystal phase at temperature s and mixing coefficient x, and can be observed through a physical experiment. Synthesizing a particular linear combination x is time consuming, but allows easy observation of $f(s, x)$ for all temperatures s. This is a coupled root-finding problem.

Coupled root-finding also arises in remote sensing, when finding the boundary of a forest or other geographical feature from an airborne laser scanner [1]. Here, an aircraft chooses a latitude x at which to fly and observes the presence or absence of the feature, $f(s, x) \in \{0, 1\}$, for all longitudes s in the flight path. The boundary at longitude s is given by the root $x^*(s)$.

Naively, we could discretize the interval $[0, 1)$ and calculate f with respect to all discretized values of x. Although this is easy to program and understand, the computational investment can be massive and unnecessary. We develop a coupled bisection method that can sort these elements in a more efficient fashion.

We solve this problem by sequentially evaluating f at different values of x. When we evaluate f at some value x, we find $f(s, x)$ for every element s, and we can deduce for every element whether $x^*(s) \geq x$ or $x^*(s) < x$. At every iteration, we know of a subinterval

* Corresponding author.
E-mail addresses: snp32@cornell.edu (S.N. Pallone), pf98@cornell.edu (P.I. Frazier), sgh9@cornell.edu (S.G. Henderson).

http://dx.doi.org/10.1016/j.orl.2015.12.015
0167-6377/

that contains $x^*(s)$. These subintervals form a disjoint partition of $[0, 1)$. By evaluating f for a different value of x at each iteration, we refine the previous partition, choosing one subinterval to split into two. This continues until for every element each subinterval contains at most one root.

In this process, we must find a way to sequentially select the next value of x at which we evaluate f. One might conjecture by analogy that the optimal decision is to choose some subinterval and select the midpoint of that interval to be the next value of x. This policy is *not* optimal, but we show it can sort the elements by their associated zeros in $O(|\mathcal{S}|)$ iterations in expectation, and calculate the asymptotic constant to be bounded above by 1.44. We also provide a lower bound of $|\mathcal{S}| - 1$ for the minimum number of iterations for any policy, implying an approximation guarantee of 1.44. Moreover, we give computational evidence suggesting our proposed policy is even closer to optimal than the 1.44 guarantee suggests.

2. Problem specification

We first model the underlying decision process. Suppose $X = \left\{[x^{(i)}, x^{(i+1)}) : i = 0, \ldots, m\right\}$ denotes a partition of the interval $[x^{(0)}, x^{(m+1)})$. We assume $x^{(i)} < x^{(i+1)}$ for all i. Let $N = (n^{(0)}, n^{(1)}, \ldots, n^{(m)})$ represent the numbers of roots that lie in the corresponding subintervals in X. Together, the pair (X, N) determine the *computational state*. Suppose at decision epoch j, our current computational state is (X_j, N_j). We choose a refined partition

$$X_{j+1} = \left\{\left[x_j^{(0)}, x_j^{(1)}\right), \ldots, \left[x_j^{(\ell)}, \bar{x}_{j+1}\right), \left[\bar{x}_{j+1}, x_j^{(\ell+1)}\right), \ldots, \left[x_j^{(j)}, x_j^{(j+1)}\right)\right\},$$

where $\ell \in \{0, \ldots, j\}$ and $\bar{x}_{j+1} \in (x_j^{(\ell)}, x_j^{(\ell+1)})$. Accordingly, let $\mathbb{X}(X_j)$ be the set containing all refined partitions of X_j containing $j + 1$ subintervals. We then evaluate f at $\bar{x}_{j+1}$. For all elements $s \in \mathcal{S}$ we observe $f(s, \bar{x}_{j+1})$, and therefore we can determine whether $x^*(s)$ is less than or greater than $\bar{x}_{j+1}$. At this point, the $n_j^{(\ell)}$ roots in the original subinterval $\left[x_j^{(\ell)}, x_j^{(\ell+1)}\right)$ are split among the two newly-created subintervals. Hence, we have

$$N_{j+1} = \left(n_j^{(0)}, \ldots, n_j^{(\ell-1)}, \bar{N}_{j+1}, n_j^{(\ell)} - \bar{N}_{j+1}, n_j^{(\ell+1)}, \ldots, n_j^{(j)}\right),$$

where $n_{j+1}^{(\ell)} = \bar{N}_{j+1}$ and $n_{j+1}^{(\ell+1)} = n_j^{(\ell)} - \bar{N}_{j+1}$. All other components in N_j remain the same because we learn nothing new about roots in the other subintervals.

A priori, we assign a prior distribution to the location of $x^*(s)$ for every element $s \in \mathcal{S}$. For simplicity, we assume that for every s, independent of all else, $x^*(s) \sim \text{Unif}[0, 1)$. Otherwise, as long as under the prior distribution the root locations are i.i.d. and absolutely continuous with respect to Lebesgue measure, we can use an inverse mapping to appropriately stretch the real line to yield the above case. Therefore, *a priori*, $\bar{N}_{j+1} \sim \text{Binomial}\left(n_j^{(\ell)}, (\bar{x}_{j+1} - x_j^{(\ell)})/(x_j^{(\ell+1)} - x_j^{(\ell)})\right)$. Since we would like to find an ordering for $x^*(\cdot)$, we stop evaluating f when every subinterval in the partition X contains at most one root, i.e., we stop when $n^{(i)} \le 1$ for all $i \in \{0, \ldots, |N| - 1\}$. Define the stopping time $\tau = \inf\{j \in \mathbb{N} : N_j^{(i)} \le 1 \ \forall i = 0, 1, \ldots, j\}$.

We would like to model this multiple root-finding problem as a dynamic program that finds the Bayes-optimal policy minimizing the expected number of evaluations of $f(\cdot, x)$ needed to find an ordering of all elements $s \in \mathcal{S}$ with respect to x^*. We define the value function for *computational effort* under policy π

$$W^\pi(X, N) = \mathbb{E}^\pi\left[\tau \mid X_0 = X, N_0 = N\right], \tag{1}$$

where π is a policy that maps computational states (X, N) to partitions $\bar{X} \in \mathbb{X}(X)$. The value function W^π counts the expected number of iterations needed to sort the elements $s \in \mathcal{S}$ with respect to x^* under policy π. We define the value function $W(X, N) = \inf_{\pi \in \Pi} W^\pi(X, N)$, where Π denotes the set of all policies π.

3. Recursion

Using the original definition of the value function in (1), we can derive a recursion for the computational dynamic program. For a computational state pair (X, N) that do not satisfy the stopping conditions, we have that

$$W^\pi(X, N) = 1 + \mathrm{E}^\pi\left[W^\pi(X_1, N_1) \mid (X_0, N_0) = (X, N)\right],$$

where $X_1 = \pi(X_0, N_0)$ indicates the next refinement of the partition, and N_1 is the subsequent spread of elements among subintervals.

By the principle of optimality, we can iteratively take the best partition X_1 over each step, which gives us a recursion for W, the value function under the optimal policy. Because the process $((X_j, N_j) : j \ge 0)$ is time-homogeneous, we can drop the subscripts and denote $\bar{X}$ as the refinement of partition X, and $\bar{N}$ as the resulting distribution of elements among subintervals. Thus, we have

$$W(X, N) = 1 + \min_{\bar{X} \in \mathbb{X}(X)} \mathrm{E}\left[W(\bar{X}, \bar{N}) \,\middle|\, X_0 = X, N_0 = N\right]. \tag{2}$$

3.1. Decomposition and interval invariance

We can greatly simplify this recursion by decomposing it by the different subintervals.

Theorem 1. *Under the optimal policy, the value function W has the following two properties.*

- *Decomposition:* $W(X, N) = \sum_{i=0}^{|N|-1} W\left(\left\{[x^{(i)}, x^{(i+1)})\right\}, n^{(i)}\right)$
- *Interval invariance:* $W\left(\left\{[x^{(i)}, x^{(i+1)})\right\}, n^{(i)}\right) = W\left(\{[0, 1)\}, n^{(i)}\right)$.

Proof. We will first prove the decomposition result. For any initial computational state (X, N), consider the following policy. For each subinterval $[x^{(i)}, x^{(i+1)})$, take an optimal policy for the computational state $\left(\left\{[x^{(i)}, x^{(i+1)})\right\}, n^{(i)}\right)$, and once we find a partition of the subinterval satisfying the stopping conditions, we do the same for another subinterval. Therefore, it must be $W(X, N) \le \sum_{i=0}^{|N|-1} W\left(\left\{[x^{(i)}, x^{(i+1)})\right\}, n^{(i)}\right)$.

Now we will prove the opposite inequality. First, note that the order we choose to further partition the subintervals is irrelevant, since we only seek to minimize the number of evaluations required, and each evaluation provides refinement only within its subinterval. Without loss of generality, consider only policies that evaluate the function with value within the leftmost subinterval that still does not satisfy the stopping conditions. Suppose this interval is $[x^{(i)}, x^{(i+1)})$ and contains the zeros of $n^{(i)}$ elements. Before we are allowed to move to the next subinterval, we must find a partition of $[x^{(i)}, x^{(i+1)})$ that satisfies the stopping conditions. By definition, this takes a minimum of $W\left(\left\{[x^{(i)}, x^{(i+1)})\right\}, n^{(i)}\right)$ steps. Since we only evaluate the function at one value at a time, we perform one evaluation on exactly one subinterval at each step. Therefore, repeating the same logic for every subinterval tells us $W(X, N) \ge \sum_{i=0}^{|N|-1} W\left(\left\{[x^{(i)}, x^{(i+1)})\right\}, n^{(i)}\right)$.

We will now prove the second claim of the theorem using a pathwise argument. Suppose we have initial computational state $(X_0, N_0) = \left([a, b), n^{(i)}\right)$. Define the operator $T((X, N)) = ((b - a)X + a, N)$. If we define $\left(\tilde{X}_j, \tilde{N}_j\right) = T^{-1}\left(X_j, N_j\right)$ for all time