Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

MULTISECTION: PARALLELIZED BISECTION

Stephen Pallone
Peter I. Frazier
Shane G. Henderson

Operations Research & Information Engineering
Cornell University
Rhodes Hall
Ithaca, NY 14853, USA

ABSTRACT

We consider a one-dimensional bisection method for finding the zero of a function, where function evaluations can be performed asynchronously in a parallel computing environment. Using dynamic programming, we characterize the Bayes-optimal policy for sequentially choosing points at which to query the function. In choosing these points, we face a trade-off between aggressively reducing the search space in the short term, and maintaining a desirable spread of queries in the long-term. Our results provide insight on how this trade-off is affected by function evaluation times, risk preferences, and computational budget.

1 INTRODUCTION

Parallel computing platforms are now ubiquitous, no longer being restricted to the high-performance computing environments used for massive-scale computation. Platforms that are broadly accessible include cloud computing, local computer clusters, and personal computers containing multicore processors. These platforms have the potential to allow us to more rapidly solve existing formulations of optimization problems, or to solve much larger, in the sense of computational requirements, formulations, provided that we can devise algorithms that appropriately exploit parallelism.

Perhaps the simplest optimization problem is to find the zero (assumed to exist and be unique) of a continuous function f that is positive to the right of the root and negative to the left of the root on the interval $(0,1)$, where f is known only through a computational procedure that returns $f(x)$ when queried at $x \in (0,1)$. This problem can be viewed as an optimization problem since f can be interpreted as the derivative of a smooth unimodal function that attains its minimum in $(0,1)$. We minimize that unimodal function by seeking a zero of its derivative. Even for this problem it is not clear how to exploit parallel computing, and thus this problem is our starting point and the subject of this paper.

A natural sequential approach for solving root-finding problems of such functions is bisection, in which an interval containing the root is successively reduced through evaluations of the function at the midpoint of the interval. When multiple cores are available to compute f, we might generalize bisection in such a way that we evaluate the function at multiple points simultaneously. At which points should we evaluate the function in order to rapidly reduce the interval in which we know the root lies? Throughout this paper we assume an idealized model of communication, whereby all information is available to all processors.

We use dynamic programming, in which the state of the dynamic program encapsulates the interval in which the root is known to lie and the points that are still being evaluated by cores, to attempt to identify optimal policies. The dynamic program uses a reward function that is a function of the width of the smallest interval we are certain contains the root. The dynamic program only uses the *sign* of the function values, rather than the values themselves, which keeps the calculations tractable. While using the function values themselves makes sense from an intuitive standpoint, we do not know how to formulate and solve a tractable dynamic program in that setting.

First suppose that the time required to perform a function evaluation $f(x)$ is deterministic and does not depend on x. This abstraction seems reasonable if the time to complete a function evaluation is not too variable, and leads one to consider algorithms that are, in some sense, synchronous, in that the cores operate iteratively, where the next iteration is not initiated until all cores complete processing from the current iteration. In this setting, with n cores, it is perhaps natural to evaluate the function at n equally spaced x values in the interior of the interval. Since all cores complete processing simultaneously, the interval is reduced at each stage by a factor of $(n+1)^{-1}$. This indeed turns out to be optimal when the root is initially assumed to be equally likely to be anywhere in the interval, as we assume throughout this paper.

In general, function evaluation times may depend on x and may be random. We do not address the former issue in this paper, which admittedly limits the scope of our analysis, but with good reason; identifying the optimal locations at which to evaluate a function would likely be very difficult if the time to compute $f(x)$ varies in some complicated manner with x. However, we do address the latter issue.

When processing times are random, cores do not necessarily complete processing at the same time, and one is naturally drawn to algorithms that are asynchronous, in that they do not proceed in lockstep as with synchronous algorithms. When one core reports a function value, it enables us to reduce the interval in which the root is contained, and this will render function evaluations proceeding at some of the other cores redundant. This makes the question of where to evaluate the function so as to rapidly reduce the interval a highly nontrivial one. We make the additional assumption that processing times are independent and identically exponentially distributed, which makes the analysis tractable.

The "memoryless" property of the exponential distribution ensures that the residual processing-time distribution is unchanged if we re-assign cores to different x values before they complete processing, so that nothing is lost in doing so. Furthermore, the minimum of independent exponential random variables is also exponential with a reduced mean, so one can obtain solutions more rapidly by "stacking" cores on a single x value. These properties are very special, and so while we do briefly consider them as being approximations of reality when processing times are extremely variable, as might arise in a cloud-computing environment where cores are shared by other users that are not visible to us, the main part of our analysis does not exploit them.

Our main findings in the random-processing-time context are two-fold. If stacking is not allowed, and one has only two cores available, then a policy that is similar to golden-section search is optimal under a specific risk-neutral reward function, and is close to optimal for others. When stacking *is* allowed, then for all risk preferences and computational budgets, it is always optimal to stack queries at the midpoint of the interval.

When we parallelize bisection, as in the present paper, multiple points are simultaneously evaluated. Accordingly, the term "multisection" seems appropriate as a descriptor of algorithms like those considered here, hence the title of the paper. We henceforth refer to parallelized bisection algorithms as "multisection algorithms" and the associated problem in deciding where to query the function as the "multisection problem."

The remainder of this paper is organized as follows. In Section 2 we define a stochastic process that tracks the progress of multisection. In Section 3 we introduce a dynamic-programming recursion that allows us to characterize some important properties of an optimal policy. Section 4 computes the optimal policy when processing times are deterministic and equal on all cores. Section 5 develops our primary results for random processing times in both the "no-stacking" case and the case where stacking is permitted.

2 PROBLEM SPECIFICATION

We model the allocation problem using dynamic programming and Bayesian statistics. A priori, we take the root $X^* \sim \text{Unif}(0,1)$. Observing the value of the function f at points within $(0,1)$ allows us to localize

X^* more precisely. We keep track of the smallest interval in which X^* is known to reside, and those $m \le n$ points within this interval currently being evaluated, through an ordered tuple of real numbers $S = (S^{(0)}, S^{(1)}, \dots, S^{(m)}, S^{(m+1)})$. (See Section 5.2 for an example.) In this tuple, the first and last values, $S^{(0)}$ and $S^{(m+1)}$, describe the smallest interval $[S^{(0)}, S^{(m+1)}]$ in which X^* is currently known to reside, and $S^{(1)}, \dots, S^{(m)}$ are points at which we are currently evaluating f. Without loss of generality, we assume $S^{(k)} \le S^{(k+1)}$. We construct a dynamic program, and a tuple of this form will be our state variable. When m, the number of points currently being evaluated, is strictly less than n, the number of available cores, this tuple or state indicates that we may assign idle cores to evaluate f at new points. We call such states "incomplete states." The set of incomplete states is contained in the set

$$\mathbb{S} = \{(s_0, s_1, \dots, s_m, s_{m+1}) : 0 \le m \le n,\, s_k \le s_{k+1} \text{ for } 0 \le k \le m\}.$$

We track the state variable over time. While our problem is a continuous-time problem, the state variable only changes at moments in time when a core finishes a function evaluation. These are also the moments in time when we make decisions about how to allocate newly idle cores to new points. We index this discrete set of decision epochs by j, and indicate the (pre-decision) state variable at the beginning of the jth epoch, before any idle cores are re-allocated, by $S_j = (S_j^{(0)}, \dots, S_j^{(m_j+1)})$, where m_j is the number of active cores at the beginning of the jth decision epoch. We respectively denote the left and right endpoints of the search interval S_j as $A_j = \min(S_j)$ and $B_j = \max(S_j)$.

Let the time at which the jth decision epoch occurs be $\sigma(j) \ge 0$. The first decision epoch is indexed by $j = 0$, and starts at time $\sigma(0) = 0$. The state variable at this moment in time is S_0. Typically we consider the case $S_0 = (0, 1)$, but we also consider other values for S_0 in a dynamic program constructed below. At each decision epoch j, we make a decision about how to allocate our idle cores according to some decision rule or policy.

Mathematically, we define a policy π to be a function $\pi : \mathbb{S} \to \mathbb{X}(\mathbb{S})$ that maps states to actions, where the action is an ordered tuple $X = (x_0, x_1, \cdots, x_n, x_{n+1}) \in \mathbb{S}$ of the same form as the state variable, which assigns all n cores to evaluate $f(x_1), \dots, f(x_n)$. We require that this action assign all n cores, leave unchanged those cores that are still actively performing function evaluations, and not alter the interval in which X^* is known to reside. To achieve this, we define the space of allowed actions for a given state variable (also called the "action space") as

$$\mathbb{X}(S) = \{(x_0, x_1, \dots, x_n, x_{n+1}) : X \supseteq S,\, x_0 = \min(S),\, x_{n+1} = \max(S),\, x_k \le x_{k+1} \text{ for } 0 \le k \le n\}. \tag{1}$$

The set of allowed actions at the first decision epoch is $\mathbb{X}(S_0)$. The constraint $X \supseteq S$ ensures that we do not disturb active function evaluations. The constraints $x_0 = \min(S)$ and $x_{n+1} = \max(S)$ ensure that we leave unchanged the smallest known interval containing X^*. Changing this interval requires observing the results from function evaluations.

At this first decision epoch, we also draw, for each newly assigned core $i = 1, \dots, n$, an independent random variable $\delta_0^{(i)} \sim F$, where F is some known probability distribution with support on $(0, \infty)$. The random variable $\delta_0^{(i)}$ is the wall-clock time at which the job just started on core i will complete. We let $\delta_0 = (\delta_0^{(1)}, \dots, \delta_0^{(n)})$. Our model allows early termination when the point being evaluated is eliminated from the interval containing the root by another newly completed function evaluation, and so function evaluations do not always run for the full time period $\delta_0^{(i)}$. In this paper, we consider two possibilities: either F takes a single value with probability one, or F is an exponential distribution.

Given these times, the first time at which a function evaluation will complete is $\sigma(1) = \min_i \delta_0^{(i)}$. Over the time period $(0, \sigma(1))$, the state of the cores is described by $\pi(S_0)$. Let $Q_1 = \arg\min_{i=1,\dots,n} \delta_1^{(i)}$ be the jobs that complete at time $\sigma(1)$. More than one job may finish at a time. We observe function values for all points $(S_0^{(i)} : i \in Q_1)$. These cores become momentarily idle, and the new observations of f reduce the interval in which X^* is known to reside, and may also eliminate some points not in Q_1 that are actively being evaluated. This then produces a new incomplete state variable, which we call S_1. Below we describe the conditional distribution of S_1 given $\pi(S_0)$ and Q_1. We then choose the next action using our policy, evaluating the points implied by $\pi(S_1)$.

We proceed iteratively in this fashion, constructing a sequence of random variables $(S_j, \delta_j, \sigma(j) : j = 0,1,\ldots)$. At each decision epoch $j = 1,2,\ldots$, let $Q_j = \arg\min_{i=1,\ldots,n} \delta_{j-1}^{(i)}$ be the jobs that complete at the beginning of this decision epoch, and let $\sigma(j) = \min_{i=1,\ldots,n} \delta_{j-1}^{(i)}$ be the wall-clock time at which this decision epoch occurs. At the beginning of this decision epoch, we draw S_j from its conditional distribution given S_{j-1} and Q_j, as described below. We then choose an action $\pi(S_j)$ that allocates all newly idle cores. For those cores that are still performing a previous function evaluation, we set $\delta_j^{(i)}$ to the previous value, which is $\delta_{j-1}^{(i')}$ for some i'. However, if a core is newly allocated, then we generate a new iid processing time for the new function evaluation, drawing $\delta_j^{(i)}$ such that $\delta_j^{(i)} - \sigma(j) \sim F$.

In this way, each policy implies a probability distribution P^π over the sequence of random variables $(S_j, \delta_j, \sigma(j) : j = 0,1,2,\ldots)$. Let $\mathbb{E}^\pi$ be the expectation operator corresponding to this probability distribution. Define the policy space Π to contain all policies $\pi : \mathbb{S} \mapsto \mathbb{X}(\mathbb{S})$ satisfying $\pi(S) \in \mathbb{X}(S)$ for all $S \in \mathbb{S}$. We show later that it suffices to define a policy on a much smaller domain.

The state variable does not change between decision epochs, taking the value $X_j = \pi(S_j)$ in the time interval $(\sigma(j), \sigma(j+1))$.

We now describe the conditional distribution of S_j under $\mathbb{P}^\pi$ given S_{j-1} and Q_j. For each core $i \in Q_j$ that finishes at the beginning of the jth epoch, we observe $f\left(X_{j-1}^{(i)}\right)$. From the sign of this function value, we infer whether $X^* \le X_{j-1}^{(i)}$ or $X^* > X_{j-1}^{(i)}$. We use this information to update the search interval and make it as small as possible. Accordingly, let

$$L_j = \left\{ i \in Q_j : X^* \ge X_{j-1}^{(i)} \right\} \quad \text{and} \quad U_j = \left\{ i \in Q_j : X^* \le X_{j-1}^{(i)} \right\}. \tag{2}$$

For each core $i \in L_j$, the results from the most recent function evaluations tell us that $X^* \ge X_{j-1}^{(i)}$. Let ℓ_j be largest such index, i.e., let $\ell_j = \max L_j$ if L_j is nonempty, and otherwise set $\ell_j = 0$. Likewise, set $u_j = \min U_j$ if U_j is nonempty, and otherwise set $u_j = n+1$. Thus, these most recent function evaluations allow us to localize X^* to the interval $[X_{j-1}^{(\ell_j)}, X_{j-1}^{(u_j)}]$.

We terminate a function evaluation at a point $X_{j-1}^{(i)}$ that has not finished before decision epoch j if the point is cut off by some point in Q_j, i.e., if $i < \ell_j$ or $i > u_j$. Any other jobs that have not yet completed will be allowed to run until at least the next epoch. The state S_j is then a tuple describing the new interval in which X^* is known to reside, and the jobs that are still running at time $\sigma(j)$. Formally, $S_j = (X_{j-1}^{(i)} : i = \ell_j, \ldots, u_j)$. The updated search interval is $[A_j, B_j] = [\min S_j, \max S_j]$.

Conditioning on which jobs complete at decision epoch j, we can find the probability of certain outcomes for (A_j, B_j) using the uniform prior on X^*. Given Q_j and X_{j-1}, (A_j, B_j) is conditionally independent of S_k and δ_k for all $k < j$, and

$$\mathbb{P}\left((A_j, B_j) = \left(Q_j^{(i-1)}, Q_j^{(i)} \right) \,\middle|\, Q_j, X_{j-1} \right) = \frac{Q_j^{(i)} - Q_j^{(i-1)}}{B_{j-1} - A_{j-1}}.$$

We now define a reward function $R : \mathbb{S} \to \mathbb{R}$ that measures progress in reducing the interval. For any state S_j, let $\|S_j\| = B_j - A_j$ be the length of the search interval. We choose the reward function $R(S) = \|S\|^{-r}$, where $r \in (0,1]$. The parameter r controls our risk preference. As $r \to 0$ we become more risk averse, because there is less gain for making the interval smaller. Choosing $r = 1$ minimizes risk aversion.

We choose a time horizon T, and seek to make decisions so that the reward function applied to the state at time T is large. Conceptually, one could use a fixed value for T, or allow T to be a random variable, modeling the situation in which one is uncertain when starting a computation about how long we will run it before asking it to produce an answer. For tractability, we assume that T is exponentially distributed with rate parameter α, independent of all else.

Let $\tau = \sup\{j : T \geq \sigma(j)\}$ so that the random time T falls in the interval $[\sigma(\tau), \sigma(\tau+1))$. Then the interval to which X^* has been localized at time T is $[A_\tau, B_\tau]$, which has associated reward $R(S_\tau)$.

The value, i.e., expected reward, attained by policy $\pi \in \Pi$ is $V^\pi = \mathbb{E}^\pi R(S_\tau)$. We want to find a policy that maximizes the expected reward. Let

$$V = \sup_{\pi \in \Pi} V^\pi. \tag{3}$$

An *optimal policy* is any policy π that attains the supremum in (3). An ε*-optimal policy* is any policy π such that $V^\pi \geq V - \varepsilon$ for some given $\varepsilon > 0$.

2.1 Deterministic and Equal Waiting Times for Queries

We first assume that it takes a deterministic and constant time to evaluate the function at any point in $[0,1]$. That is, F represents a point mass at some $C > 0$. This assumption is reasonable when the variability of computation times is small. The wait time is independent of where the function is being evaluated, which may not reflect reality, but makes the model more tractable. Because every task takes the same amount of time to complete, $Q_j = \{1, \dots, n\}$ for all j, and hence all n cores are reallocated at each decision point and $\sigma(j) - \sigma(j-1) = C$ for all j. This predictability makes the state space smaller and the optimal allocation easier to identify. The deterministic case will serve as a benchmark for the exponential case.

2.2 Exponential Waiting Times for Queries

What is the effect of variability of function evalution times on the optimal allocation of points? We model this variability by assuming the evalution-time distribution F is exponential with rate λ. The choice of exponential evaluation times is a computational convenience that admits evaluation uncertainty while retaining tractability. Since there is probability zero of any two independent exponential random variables taking the same value, Q_j is a singleton with probability one. The memoryless property of the exponential distribution ensures that for every j and for every $i = 1, \dots, n$, the residual processing time $\delta_j^{(i)} - \sigma(j) \sim \text{exponential}(\lambda)$, independent of all past history. Therefore, it is equally likely for any job to finish first, so that $\mathbb{P}(Q_j = \{i\}) = n^{-1}$ for every i. In addition, $\sigma(j+1) - \sigma(j)$ is exponentially distributed with rate $n\lambda$.

3 DESIRABLE PROPERTIES OF THE VALUE FUNCTION

3.1 Value Function Recursion

The problem (3) is a stochastic control problem, and we study it using dynamic programming. We first show that under the assumptions of Section 2.1 or 2.2, the problem is an infinite-horizon discrete-time Markov decision process. Under either assumption the stopping index τ has a geometric distribution with $P(\tau = j) = (1-\gamma)\gamma^j$ for $j \geq 0$, where γ can be easily computed. Indeed, when processing times are deterministically equal to $C > 0$, $\gamma = e^{-\alpha C}$, and when they are exponentially distributed, $\gamma = n\lambda/(\alpha + n\lambda)$. Using the fact that τ is independent of S_j, S_0, it can be shown that for any policy π,

$$V^\pi(S) = (1-\gamma)\mathbb{E}^\pi \left[\sum_{j=0}^{\infty} \gamma^j R(S_j) \mid S_0 = S \right].$$

Thus, except for a constant factor of $1-\gamma$, solving (3) is equivalent to solving the infinite-horizon discrete-time discounted-cost Markov decision process $\sup_\pi \mathbb{E}^\pi \left[\sum_{j=0}^{\infty} \gamma^j R(S_j) \mid S_0 = S\right]$. This Markov decision process is over the time-homogeneous controlled Markov process $(S_j : j = 0,1,2,\dots)$, which evolves independently of $(\delta_j : j = 0,1,2,\dots)$ under the assumptions of either Section 2.1 or Section 2.2.

Define the value function $V(S) = \sup_\pi V^\pi(S)$. Under the conditions of Theorem 1 below, standard results in dynamic programming, e.g., p. 46 of Hernández-Lerma and Lasserre (1996), show that

$$V(S) = (1-\gamma)R(S) + \gamma\left(\max_{X\in\mathbb{X}(S)} \mathbb{E}\left[V(S_1)\mid X_0 = X, S_0 = S\right]\right). \tag{4}$$

3.2 Scaling and Shifting Made Easy

Intuition tells us that there should be a relationship between states that are either scaled or shifted. This would greatly reduce the size of the state space. For notational convenience, for any $h,c\in\mathbb{R}$, let $(hS+c)^{(i)} = hS^{(i)}+c$ for every i. Recall that $R(S) = \|S\|^{-r}$ for $r\in(0,1]$. Then one can verify that $R(aS+b) = a^{-r}R(S)$. This property extends to the value function, as follows.

Theorem 1 Suppose that $R(S) = \|S\|^{-r}$ for $r\in(0,1]$. For $c,h\in\mathbb{R}$ with $h>0$, the value function V has the following properties:

1. **Shift Invariance:** $V(S+c) = V(S)$
2. **Inverse Scaling:** $V(hS) = h^{-r}V(S)$

Proof. We show the result by induction. For a function $f:\mathbb{S}\to\mathbb{R}$, the Bellman operator T is given by

$$Tf(\cdot) = (1-\gamma)R(\cdot) + \gamma\left(\max_{X\in\mathbb{X}(\cdot)} \mathbb{E}\left[f(S_{j+1})\mid S_j = \cdot, X_j = X\right]\right).$$

We claim that the result holds for $T^{(k)}R$ for all $k\in\mathbb{N}$. As a base-case, consider $k=0$. We let $T^{(0)}R(\cdot) = R$, so the result holds trivially. Now assume $T^{(k)}R$ has the above properties, and let $\bar{S} = hS+c$, $\bar{X} = hX+c$.

$$T(T^{(k)}R(\bar{S})) = (1-\gamma)R(\bar{S}) + \gamma\left(\max_{\bar{X}\in\mathbb{X}(\bar{S})} \mathbb{E}\left[T^{(k)}R(S_{j+1})\mid S_j = \bar{S}, X_j = \bar{X}\right]\right).$$

The transition kernel itself is shift and scale invariant. That is, $\mathbb{P}(S_{j+1} = S' \mid S_j = S, X) = \mathbb{P}(S_{j+1} = hS'+c \mid S_j = hS+c, hX+c)$. Therefore, for any measurable function g, we have $\mathbb{E}[g(S_{j+1})\mid S_j = S, X_j = X] = \mathbb{E}[g(hS_{j+1}+c)\mid S_j = \bar{S}, X_j = \bar{X}]$. We can rewrite the expectation using the process $(S_j : j\in\mathbb{N})$. Using the induction hypothesis, we see

$$T(T^{(k)}R(\bar{S})) = (1-\gamma)R(hS+c) + \gamma\left(\max_{X\in\mathbb{X}(S)} \mathbb{E}\left[T^{(k)}R(hS_{j+1}+c)\mid S_j = S, X_j = X\right]\right)$$
$$T^{(k+1)}R(\bar{S}) = h^{-r}(1-\gamma)R(S) + h^{-r}\gamma\left(\max_{X\in\mathbb{X}(S)} \mathbb{E}\left[T^{(k)}R(S_{j+1})\mid S_j = S, X_j = X\right]\right),$$

i.e., that $T^{(k+1)}R(\bar{S}) = h^{-r}T^{(k+1)}R(S)$. Therefore, we have proved that the result holds for $T^{(k)}R$, for all k. We now wish to assert $\lim_{k\to\infty}T^{(k)}R(S) = V(S)$ for all $S\in\mathbb{S}$. According to the result on p.46 of Hernández-Lerma and Lasserre (1996) this holds under the following three conditions listed on p.28.

- The action space is compact: As we defined it, the action space $\mathbb{X}(S)$ for every S is a bounded subset of $\mathbb{R}^{n+2}$ that is the intersection of a finite number of closed halfspaces; see (1).
- The reward R is lower semi-continuous and non-negative: We have $R(S_j) = [B_j - A_j]^{-r}$, and if we define $R(S_j) = \infty$ for states where $A_j = B_j$, then the inverse image of every open set of rewards is an open set of states, which is exactly the definition of lower semi-continuity. Hence, R is lower semi-continuous on a compact set.

- The transition kernel is continuous: The construction in Section 2 ensures this, although a full verification is cumbersome and thus omitted.

Therefore, $T^{(k)}R \to V$ pointwise as $k \to \infty$. Moreover, for any state S and parameters h, c, there exists K_1 and K_2 such that for all $k \geq K_1$, we have $|V(S) - T^k R(S)| \leq \varepsilon$, and for all $k \geq K_2$, we have $|V(hS+c) - T^k R(hS+c)| \leq \varepsilon$. So for all $k \geq \max\{K_1, K_2\}$,

$$\begin{aligned}\varepsilon &\geq |V(hS+c) - T^{(k)}R(hS+c)| \\ &= |V(hS+c) - h^{-r}V(S) + h^{-r}V(S) - h^{-r}T^{(k)}R(S)| \geq |V(hS+c) - h^{-r}V(S)| - h^{-r}\varepsilon,\end{aligned}$$

which shows that the value function also has the desired properties. □

3.3 Normalized Scaling and State Space Reduction

Corollary 2 For some state $S \in \mathbb{S}$, let $\bar{S} = \frac{S - S^{(0)}}{S^{(n+1)} - S^{(0)}}$ so that $\bar{S}^{(0)} = 0$ and $\bar{S}^{(n+1)} = 1$. Then from the previous theorem, we have

$$V(S) = V\left((S^{(n+1)} - S^{(0)})\bar{S} + S^{(0)}\right) = \left(S^{(n+1)} - S^{(0)}\right)^{-r} V(\bar{S}) = R(S)V(\bar{S})$$

The corollary demonstrates the structure embedded in the bisection algorithm. From a theoretical standpoint, we can reduce our attention to all states within the interval $[0,1]$. This will help us identify the structure of optimal policies.

4 OPTIMAL POLICY STRUCTURE FOR DETERMINISTIC AND EQUAL PROCESSING TIMES

Suppose that the evaluations take a deterministic amount of time to compute. If we start the first batch of function evaluations simultaneously, then the jobs finish all at once, and all n points can be allocated. Using the recursion, we can prove that an optimal allocation at every step is for the points to be equally spaced in the interval.

Theorem 3 Suppose the function evaluations require a deterministic and equal amount of time to compute. Let $R(\cdot)$ be defined as before, with $r \in (0,1)$. If $\tilde{X}_j = (\tilde{X}_j^{(0)}, \tilde{X}_j^{(1)}, \ldots, \tilde{X}_j^{(n)}, \tilde{X}_j^{(n+1)})$ is defined by

$$\tilde{X}_j^{(i)} = A_j + \frac{i}{n+1}(B_j - A_j),$$

then $\tilde{X}$ is an optimal allocation for $S = (A_j, B_j)$, and this defines an optimal policy.

Proof. At decision epoch j we must allocate all n points because $Q_j = \{1, \ldots, n\}$ for all $j \geq 0$ with probability 1. Thus, it is only necessary for the state to keep track of the endpoints of the interval because there are no jobs that are left to continue. For an allocation X to be optimal for incomplete state S, it must attain the maximum in

$$\max_{X \in \mathbb{X}(S)} \mathbb{E}V(S) = \max_{X \in \mathbb{X}(S)} \sum_{i=0}^{n} \left(\frac{X^{(i+1)} - X^{(i)}}{X^{(n+1)} - X^{(0)}}\right) V\left((X^{(i)}, X^{(i+1)})\right)$$

Using the inverse scaling property,

$$\begin{aligned}\max_{X \in \mathbb{X}(S)} \mathbb{E}V(S) &= \max_{X \in \mathbb{X}(S)} \sum_{i=0}^{n} \frac{X^{(i+1)} - X^{(i)}}{\|S\|} \left(X^{(i+1)} - X^{(i)}\right)^{-r} V((0,1)) \\ &= \|S\|^{-1} V((0,1)) \max_{X \in \mathbb{X}(S)} \sum_{i=0}^{n} (X^{(i+1)} - X^{(i)})^{1-r}.\end{aligned}$$

In this maximization, $X \in \mathbb{X}(S)$ implies we can choose $S^{(0)} = X^{(0)} \leq X^{(1)} \leq \cdots X^{(n)} \leq X^{(n+1)} = S^{(n+1)}$ arbitrarily, since no jobs are ongoing. Changing variables to $u_i = X^{(i+1)} - X^{(i)}$ for $i = 0, 1, \ldots, n$, the problem becomes that of maximizing $\sum_{i=0}^{n} u_i^{1-r}$, subject to the constraints that $\sum_{i=0}^{n} u_i = \|S\|$ and $u_i \geq 0$ for every i. This is a concave maximization problem with local, and therefore global, minimum when $u_0 = u_1 = \cdots = u_n = \|S\|(n+1)^{-1}$, which corresponds to $\tilde{X}$. The maximum is unique when $r \in (0,1)$ since the objective function is strictly concave. When $r = 1$, $\tilde{X}$ is still optimal, but it is not the uniquely optimal solution. Lastly, from p.46 of Hernández-Lerma and Lasserre (1996), iteratively choosing the optimal allocation $\tilde{X}$ for every S yields an optimal policy. □

Hence equally-spaced allocation provides the optimal solution to the dynamic program for deterministic function evaluations. It is encouraging to see a result that is consistent with the classical bisection method.

5 OPTIMAL POLICY STRUCTURE FOR IID EXPONENTIAL PROCESSING TIMES

We now characterize optimal policies in the presence of exponentially distributed processing times. Function evaluations no longer finish simultaneously, so we must reallocate points without the information from jobs that are still running.

5.1 A Simplified Recursion

Recall that each core $i = 1, 2, \ldots, n$ is equally likely to return a function value at every decision point. If Core i returns with the information that $X^* > X^{(i)}$, then the resulting incomplete state is $X_+^{(i)} = (X^{(i)}, X^{(i+1)}, \ldots, X^{(n)}, X^{(n+1)})$. Otherwise, if the process returns with $X^* < X^{(i)}$, then the resulting incomplete state is $X_-^{(i)} = (X^{(0)}, X^{(1)}, \ldots, X^{(i-1)}, X^{(i)})$. Therefore

$$\mathbb{P}\left(S_{j+1} = X_+^{(i)} \;\middle|\; S_j, X_j\right) = \left(\frac{1}{n}\right)\left(\frac{B_j - X_j^{(i)}}{B_j - A_j}\right) \quad \text{and} \quad \mathbb{P}\left(S_{j+1} = X_-^{(i)} \;\middle|\; S_j, X_j\right) = \left(\frac{1}{n}\right)\left(\frac{X_j^{(i)} - A_j}{B_j - A_j}\right).$$

We can now analyze the dynamic-programming recursion. Conditioning on the action X_j,

$$\begin{aligned} V(S_j) &= (1-\gamma)R(S_j) + \gamma \max_{X \in \mathbb{X}(S_j)} \mathbb{E}[V(S_{j+1}) \,|\, S_j, X_j = X] \\ &= (1-\gamma)R(S_j) + \gamma \max_{X \in \mathbb{X}(S_j)} \frac{1}{n} \sum_{i=1}^{n} \left[\left(\frac{X^{(i)} - A_j}{B_j - A_j}\right) V(X_-^{(i)}) + \left(\frac{B_j - X^{(i)}}{B_j - A_j}\right) V(X_+^{(i)}) \right]. \end{aligned}$$

Now assume S_j is normalized, i.e., $\|S_j\| = 1$. We want to express $V(S_j)$ in terms of the value of other normalized states. Scaling and shifting, we find that

$$\begin{aligned} V(S_j) &= (1-\gamma)(1) + \gamma \max_{X \in \mathbb{X}(S_j)} \frac{1}{n} \sum_{i=1}^{n} \left[X^{(i)} V(X_-^{(i)}) + (1 - X^{(i)}) V(X_+^{(i)}) \right] \\ &= (1-\gamma) + \gamma \max_{X \in \mathbb{X}(S_t)} \frac{1}{n} \sum_{i=1}^{n} \left[(X^{(i)})^{1-r} V(\bar{X}_-^{(i)}) + (1 - X^{(i)})^{1-r} V(\bar{X}_+^{(i)}) \right], \end{aligned}$$

where $\bar{X}_+^{(i)}$ and $\bar{X}_-^{(i)}$ are the respective normalized states for $X_+^{(i)} = (X^{(i)}, \ldots, X^{(n+1)})$ and $X_-^{(i)} = (X^{(0)}, \ldots, X^{(i)})$. Now we can restrict the state space to $\{S \in \mathbb{S} : \min(S) = 0, \max(S) = 1\}$. In fact, for the rest of the paper, we consider only normalized states and allocations.

In addition to expressing the value function in terms of the current state, we can also express it in terms of the action X. One can view X as the post-decision state, and $W(X)$, the expected downstream reward resulting from action X, as a so-called Q-factor or post-decision value function; see Powell (2011). Formally, we define $W(X)$ as

$$W(X) = \frac{1}{n}\sum_{i=1}^{n}(X^{(i)})^{1-r}V(\bar{X}_{-}^{(i)}) + (1-X^{(i)})^{1-r}V(\bar{X}_{+}^{(i)}). \tag{5}$$

For $z \in [0,1]$, let $\beta(z) = z^{1-r} + (1-z)^{1-r}$, with $r \in (0,1]$ defined as in the reward function R. In an abuse of notation, for an allocation X, we define $\beta(X) = n^{-1}\sum_{i=1}^{n}\beta(X^{(i)})$, and $\beta((0,1)) = 1$. Using the original definition V, we can write W as a recursion, yielding

$$W(X) = (1-\gamma)\beta(X) + \frac{\gamma}{n}\sum_{i=1}^{n}(X^{(i)})^{1-r}\left[\max_{Y\in\mathbb{X}(\bar{X}_{-}^{(i)})} W(Y)\right] + (1-X^{(i)})^{1-r}\left[\max_{Z\in\mathbb{X}(\bar{X}_{+}^{(i)})} W(Z)\right]. \tag{6}$$

5.2 Golden Section Policy

The above recursions and shifting/scaling properties reveal the structure inherent in the Multisection Algorithm. We now want to leverage this structure in deriving an optimal policy. When we have two cores, i.e., $n=2$, we can describe a policy that reduces the interval size by a constant factor in every step, and guarentees that the (normalized) allocation at every step is identical.

Let $X = X_0 = \pi((0,1))$. When $n=2$, we have $X = (0,a,b,1)$, with $X^{(1)} = a$ and $X^{(2)} = b$. By normalizing, we have $\bar{X}_{+}^{(1)} = \left(0, \frac{b-a}{1-a}, 1\right)$ and $\bar{X}_{-}^{(2)} = (0, \frac{a}{b}, 1)$. We only need to consider these states because $\bar{X}_{-}^{(1)} = \bar{X}_{+}^{(2)} = (0,1)$, meaning that both cores need to be reallocated. If we want $\pi(\bar{X}_{+}^{(i)}) = \pi(\bar{X}_{-}^{(i)}) = X$ for $i = 1,2$ and some allocation X, i.e., that the allocation is the same for both states, then we require that $(0, \frac{b-a}{1-a}, 1) \subset (0,a,b,1)$ and $(0, \frac{a}{b}, 1) \subset (0,a,b,1)$. One way to satisfy this is for $a = (b-a)/(1-a)$ and $b = a/b$. This system of equations has the unique solution

$$X^{(1)} = a = \frac{1}{2}\left(3-\sqrt{5}\right) \qquad X^{(2)} = b = \frac{1}{2}\left(\sqrt{5}-1\right). \tag{7}$$

Consider a policy π such that $\pi((0,1)) = \pi((0,a,1)) = \pi((0,b,1)) = (0,a,b,1)$ and $\pi(hS+c) = h\pi(S)+c$. We call this policy the Golden Section Policy because it uses the same queries as the Golden Section Search Algorithm (Kiefer 1953) for finding the extremum (either maximum or minimum) of a unimodular function. In both situations, Golden Section uses symmetry to maintain the same spread of queries. In the context of multisection, the construction of the Golden Section policy implies that if $X_j = X$, then $\pi(\bar{X}_{+}^{(i)}) = \pi(\bar{X}_{-}^{(i)}) = X$ for $i = 1,2$, implying that $X_{j+1} = X_j$ for all $j \geq 0$. This property can be used to show that Golden Section is optimal under a risk-neutral reward function.

5.2.1 On the Fringe: Optimal When Risk-Indifferent

We now show that under risk indifference, i.e., $r = 1$, Golden Section is optimal.

Theorem 4 Let π be the Golden Section policy as defined above. If $r = 1$, then π is optimal.

Proof. From the recursion in $W(\cdot)$, we see for any allocation $X \in \arg\max_{Z\in\mathbb{X}((0,1))} W(Z)$,

$$W(X) = (1-\gamma)\beta(X) + \frac{\gamma}{n}\sum_{i=1}^{n}(X^{(i)})^{1-r}\left[\max_{Z_1\in\mathbb{X}(\bar{X}_{-}^{(i)})} W(Z_1)\right] + (1-X^{(i)})^{1-r}\left[\max_{Z_2\in\mathbb{X}(\bar{X}_{+}^{(i)})} W(Z_2)\right].$$

If we relax the constraints in the two maxima, we see that

$$\begin{aligned}W(X) &\leq (1-\gamma)\beta(X) + \frac{\gamma}{n}\sum_{i=1}^{n}(X^{(i)})^{1-r}\left[\max_{Z_1\in\mathbb{X}((0,1))} W(Z_1)\right] + (1-X^{(i)})^{1-r}\left[\max_{Z_2\in\mathbb{X}((0,1))} W(Z_2)\right]\\ &= (1-\gamma)\beta(X) + \gamma W(X)\frac{1}{n}\sum_{i=1}^{n}[(X^{(i)})^{1-r} + (1-X^{(i)})^{1-r}]\\ &= (1-\gamma)\beta(X) + \gamma W(X)\beta(X).\end{aligned}$$

By iteratively substituting the above inequality for $W(X)$, we can express the upper bound as the infinite geometric series

$$W(X) \leq (1-\gamma)\beta(X)\sum_{p=0}^{\infty}\gamma^{p}\beta(X)^{p}, \tag{8}$$

which converges when $\gamma\beta(X) < 1$ and diverges to infinity otherwise. But if we consider the value of the Golden Section policy, with Y being the Golden Section allocation,

$$\begin{aligned}W^{\pi}(Y) &= (1-\gamma)\beta(Y) + \frac{\gamma}{n}\sum_{i=1}^{n}(Y^{(i)})^{1-r}W^{\pi}(\pi(\bar{Y}_{-}^{(i)}))) + (1-Y^{(i)})^{1-r}W^{\pi}(\pi(\bar{Y}_{+}^{(i)}))\\ &= (1-\gamma)\beta(Y) + \frac{\gamma}{n}\sum_{i=1}^{n}(Y^{(i)})^{1-r}W^{\pi}(Y) + (1-Y^{(i)})^{1-r}W^{\pi}(Y),\end{aligned}$$

and we can also write the post-decision value of choosing the Golden Section as

$$W^{\pi}(Y) = (1-\gamma)\beta(Y)\sum_{p=0}^{\infty}\gamma^{p}\beta(Y)^{p}. \tag{9}$$

If $r = 1$, we have $\beta(X) = 2$ for any allocation X, which makes (8) and (9) both equal. If $\gamma < 1/2$, then the upper bound equals $2(1-\gamma)/(1-2\gamma)$. If $\gamma \geq 1/2$, then the series diverges to infinity. In either case, choosing the Golden Section policy π achieves the upper bound, and is therefore optimal. $\square$

When $r < 1$, the Golden Section Policy still performs very well. In the $W(\cdot)$ recursion, the immediate reward obtained from assigning queries with allocation X is given by $\beta(X)$. Since the Golden Section Policy requires only one allocation, it is sufficent to use this as a metric for the performance of the entire policy. Now suppose X is the Golden Section Allocation. If we compare $\beta(X)$ versus $\max_z \beta(z) = 2^r$, which is achieved by the Bisection Method, there is an extremely small gap. At its smallest, the ratio $\beta(X)/2^r \geq 0.993$, achieved when $r = 0.502$. And from numerically solving discretized versions of the dynamic program, there is strong computational evidence that the Golden Section Policy is optimal for any $0 < r < 1$ with γ sufficiently large.

We can also compare the Golden Section Policy to the traditional Bisection Method in terms of $W(\cdot)$. Let w_1 and w_2 be the respective post-decision state values for Bisection and Golden Section policies. Using (9) and the definition of γ in Section 3.1, we can derive closed form expressions for w_1 and w_2 in terms of α, λ, and r. In general, Golden Section achieves a higher expected reward because it effectively uses the additional information, and the loss from cutting off-center is relatively small. There are values of parameters (eg. $\alpha = 0.5$, $\lambda = 1.2$, $r = 0.5$) for which Golden Section performs infinitely better under this metric. However, in cases where λ is small, it is more advantageous to use Bisection.

5.2.2 Larger Numbers of Cores

For a general number of cores n, it is difficult to explicitly find policies similar to Golden Section. Even when $n = 3$, one cannot rely on a single allocation to define a policy. (For the case of three dimensions, finding such a policy is equivalent to solving a system with three variables and six nonlinear equations, none of which are redundant.) A fundamental idea behind Golden Section is that reducing the number of states visited over time is beneficial for the long-run performance of the algorithm. We might then relax the condition that we only visit a single state, i.e., $X_{j+1} = X_j$ for every j, and instead design a policy that visits a *finite* number of states. This might be a more achievable goal in higher dimensions, even though it is not guaranteed to yield an optimal policy.

For example, consider a policy for the case $n = 3$ defined by two allocations $Y = (0, 1/3, 1/2, 2/3, 1)$ and $Z = (0, 1/4, 1/2, 3/4, 1)$. The potential incomplete states arising from Y or Z, (exploiting the fact that both Y and Z are symmetric about $1/2$ so as to reduce the number of incomplete states), are

$$\begin{aligned} \bar{Y}_+^{(1)} &= (0,1/4,1/2,1) & \bar{Y}_-^{(1)} &= (0,1) & \bar{Y}_+^{(2)} &= (0,1/3,1) \\ \bar{Z}_+^{(1)} &= (0,1/3,2/3,1) & \bar{Z}_-^{(1)} &= (0,1) & \bar{Z}_+^{(2)} &= (0,1/2,1), \end{aligned} \tag{10}$$

which means that no matter what job finishes first, we will always be able to choose action Y or action Z to reallocate the idle processors. In other words, starting from Y or Z, for every $j \geq 0$ there always exists some $X_j \in \{Y, Z\} \cap \mathbb{X}(S_j)$. Such a choice of policy yields a Markov Chain on the allocation space $\mathbb{X}((0,1))$ with only two states. This kind of policy is desirable because it easy to describe and easy to analyze. Even though such a policy is reminiscient of Golden Section, both actions are not equally preferable. The queries of Y are positioned closer to the midpoint than those of Z, which means Y is likely to reduce the interval length by a greater amount. Because of this, in cases where we have a choice between Y and Z, selecting Y is preferable, and one can verify that $W(Y) > W(Z)$. Even so, the existence of a policy with a small action space is promising. We suspect that similar policies exist for larger n, although finding them has proven to be difficult.

5.3 To Stack or Not to Stack

The memoryless property of the exponential distribution means that the residual processing time for a core is always exponentially distributed with the same mean. This observation suggests that each time a core completes, we might re-assign all cores that are still running as well as those that have been made redundant. Our formulation in Section 2 did not allow this, because we required that $X \subset S$, i.e., that the action leaves cores running that are still in process. We continue to require that non-redundant cores continue to run, but there is a subtlety in our setup that has a substantial impact on the form of the optimal policy. With exponential processing times, with probability 1 we see a single core, i say, return a value at each decision point, so that either $(\ell_j, u_j) = (0, i)$ or $(\ell_j, u_j) = (i, n+1)$ in the discussion immediately following (2). If, for example, Core i tells us that $X^* > X^{(i)}$, then just after (2) our formulation allows us to reassign all cores with indices less than i. In practice, we have the option of doing more. If Core $i+1$ is also evaluating at $X^{(i)}$, then we can also reassign Core $i+1$, along with any other cores $j > i$ for which $X^{(j)} = X^{(i)}$. In our present formulation this is not allowed.

We made this modeling choice because it seemed, to us, to be realistic to not "stack cores," i.e., to not assign them to evaluate the same point. However, in situations such as in cloud-computing environments where cores can be quite unreliable and the speed of processing can be highly variable from one core to another, we may have incentive to evaluate very similar points on multiple cores, i.e., to adopt stacking policies. We now adapt our formulation to allow such policies.

Suppose now that when Core i concludes that $X^* > X^{(i)}$, we reassign all cores j for which $X^{(j)} \leq X^{(i)}$, *including those with indices greater than i.* Similarly, when $X^* \leq X^{(i)}$, we reassign all cores j for which $X^{(j)} \geq X^{(i)}$. Now the transition kernel is no longer continuous because, for example, there is a large difference between stacking two adjacent points, and instead evaluating two points that are minutely but positively separated. Hence, the argument in Theorem 1 no longer applies, and we are no longer assured that an optimal value function exists and satisfies the Bellman recursion. *We conjecture that this is so, and proceed in this section under this conjecture.*

5.3.1 Optimal Stacking Policy Structure

It turns out that now the optimal policy for any set of parameters is one where all cores stack at the midpoint of the interval.

Theorem 5 Stacking queries $X_j^{(i)} = \frac{1}{2}(B_j - A_j)$ for every core i is the optimal allocation for independent exponentially distributed waiting times with common rate.

Proof. Since $V(S_0)$ is the optimal allocation when *all* n cores are assigned, $V(\bar{X}_-^{(i)}) \leq V(S_0)$ because the value function is more constrained. Therefore,

$$\begin{aligned}
V(S_0) &= (1-\gamma) + \gamma \max_{X \in \mathbb{X}(S_0)} \frac{1}{n} \sum_{i=1}^{n} \left[(X^{(i)})^{1-r} V(\bar{X}_-^{(i)}) + (1 - X^{(i)})^{1-r} V(\bar{X}_+^{(i)}) \right] \\
V(S_0) &\leq (1-\gamma) + \gamma \max_{X \in \mathbb{X}(S_0)} \frac{1}{n} \sum_{i=1}^{n} \left[(X^{(i)})^{1-r} V(S_0) + (1 - X^{(i)})^{1-r} V(S_0) \right] \\
&= (1-\gamma) + V(S_0)\gamma \max_{X \in \mathbb{X}(S_0)} \frac{1}{n} \sum_{i=1}^{n} \left[(X^{(i)})^{1-r} + (1 - X^{(i)})^{1-r} \right]. \\
V(S_0) &\leq (1-\gamma) + V(S_0)\gamma 2^r,
\end{aligned}$$

with equality if all points $X^{(i)}$ are stacked at $1/2$. Hence, stacking at $1/2$ is optimal regardless of the value of γ. Moreover, if $\gamma < 2^{-r}$ then the optimal value is finite. □

ACKNOWLEDGMENTS

Peter Frazier was supported by NSF CAREER CMMI-1254298, NSF IIS-1247696, AFOSR YIP FA9550-11-1-0083, and AFOSR FA9550-12-1-0200. Shane Henderson's work was partially supported by National Science Foundation Grant CMMI-1200315.

REFERENCES

Hernández-Lerma, O., and J. B. Lasserre. 1996. *Discrete-time Markov control processes*. Springer.

Kiefer, J. 1953. "Sequential minimax search for a maximum". *Proceedings of the American Mathematical Society* 4 (3): 502–506.

Powell, W. B. 2011. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, Volume 842. John Wiley & Sons.

AUTHOR BIOGRAPHIES

STEPHEN PALLONE is a PhD student in the School of Operations Research and Information Engineering at Cornell University. His research interest is in developing robust convex optimization algorithms that can take advantage of parallel architecture and utilize noisy information. His email address is `<snp32@cornell.edu>` and his website is http://people.orie.cornell.edu/snp32.

PETER I. FRAZIER is an assistant professor in the School of Operations Research and Information Engineering at Cornell University. His research interest is in dynamic programming, Bayesian statistics, and optimal learning. His web page is http://people.orie.cornell.edu/pfrazier.

SHANE G. HENDERSON is a professor in the School of Operations Research and Information Engineering at Cornell University. His research interests include discrete-event simulation and simulation optimization, and he has worked for some time with emergency services. He co-edited the Proceedings of the 2007 Winter Simulation Conference. His web page is http://people.orie.cornell.edu/shane.