Submitted to *SIAM Journal on Scientific Computing*

# Using Forward Induction to Evaluate Sequential Allocation Procedures

Janis P. Hardwick
Statistics Department

Quentin F. Stout
EECS Department

University of Michigan, Ann Arbor, MI 48109

**Abstract:** Forward induction is a technique to speed the process of making multiple exact evaluations of a sequential allocation procedure, where the options are discrete and their outcomes follow a discrete distribution. Multiple evaluations are needed for determining criteria such as maxima or minima over regions (where the location of the extremal value is unknown in advance), for visualizing characteristics such as robustness, or for obtaining the distribution of a statistic rather than just its mean. By using an initial phase to determine the number of paths reaching each terminal state, the subsequent evaluations are far faster than repeated use of backward induction. Algorithms are given for fully sequential and staged sequential procedures, and the procedures can be either deterministic or random. The procedures can be generated by any technique (including dynamic programming), and the evaluations performed can be quite flexible and need not be related to the method of obtaining the procedure. While the emphasis is on forward induction, the techniques used to speed up the analyses of staged allocation procedures can also be used to improve backward induction for such procedures. If multiple evaluations need to be carried out, however, forward induction will still be far superior.

**Keywords:** backward induction, adaptive allocation, staged allocation, path counting, Bayesian, bandit problems

**AMS Classifications:** 62L10, 90C35, 68Q25, 62A15

Copyright ©1995.

Last modified: 19 Sep 1995.

Research supported in part by the National Science Foundation under grants DMS-9157715 and DMS-9504980.

# 1 Introduction

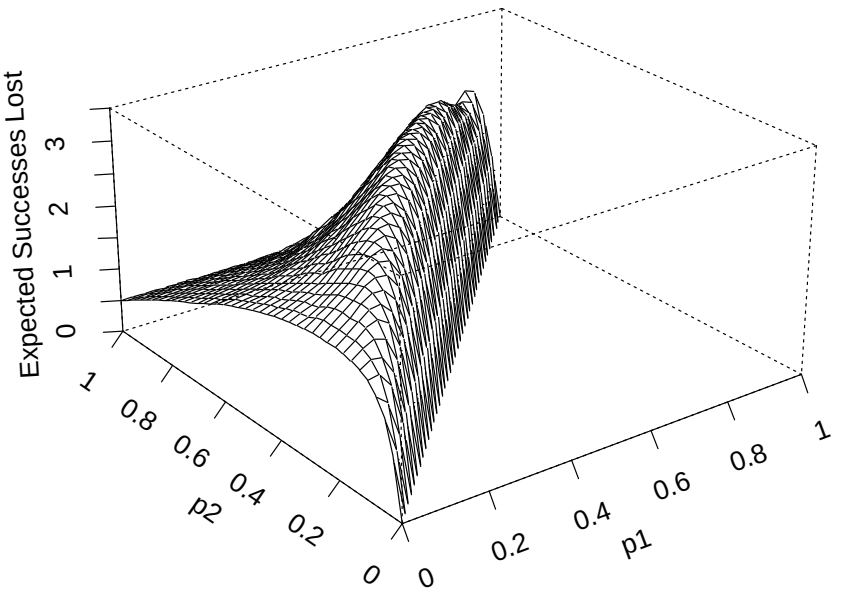
Sequential allocation procedures use accruing data to control their decisions, rather than making all decisions in advance of any data collection. This adaptive feature makes them more efficient than fixed allocation procedures, but also makes them more complex to analyze and optimize. Because of the analytical difficulties, sequential procedures are often overlooked, even when they offer substantial benefits. Fortunately, advances in computer technology are making computational analyses more practical and are expanding the range of sequential procedures that can be considered. The well-known techniques of dynamic programming and backward induction, both of which proceed from the end of the experiment towards the front, can be used for optimization and evaluation, respectively, when the options and outcomes are finite.

However, there are some situations which are still quite computationally challenging. For example, suppose one has a sequential procedure for allocating between two Bernoulli arms, and wants to evaluate a criterion such as the probability of correctly selecting the better arm if their success probabilities differ by $\delta > 0$ or more (see [3]). It is known that the minimum such probability occurs when the probabilities differ by exactly $\delta$, but in general it is not known which pair of probabilities will achieve the minimum. Thus one needs to evaluate some collection of success probabilities until the minimum has been located to sufficient accuracy. This is an example of *multiple evaluation*, where the same procedure needs to be evaluated several times.

Other examples of multiple evaluation occur when one tries to visualize criteria, such as expected failures, as a function of the success probabilities of the two arms. That is, one may not be satisfied with just a single such evaluation, but rather insist on seeing the entire surface in order to have a better understanding of the workings of the procedure. Or one may have a Bayesian procedure, created using some specified prior distribution, and want to examine its robustness by evaluating it using several different priors. Yet another example is when one wants to determine the distribution of the criterion, rather than just its mean.

While it may be possible to perform multiple evaluations by using exact analytical expressions, this is quite rare and it limits investigators to simple procedures optimized for analytical tractability, rather than for usefulness. In general, the standard approach for obtaining exact results has been simply to run backward induction multiple times, once per evaluation. This approach can be quite time-consuming and limit the number of evaluations attempted and/or the size of experiment that researchers can analyze. (This point is pursued further in Section 6.) Yet another approach is to run multiple simulation studies, one per evaluation, but this approach reduces accuracy and is time-consuming since each simulation study requires numerous passes.

Forward induction is a computational technique designed to alleviate these problems. It consists of two phases: initialization and evaluation. The initialization phase works from the front of the experiment towards its conclusion, determining the number of paths that reach each terminal state. The evaluation phase then performs each evaluation using only the terminal states and the path counts. As will be shown, this can significantly reduce the total number of calculations required. Forward induction, which is exact and quite general, is applicable to frequentist or Bayesian procedures and analyses as well as to a wide range of criteria. While the technique will be explained in the context of deterministic procedures, it can easily be adapted to randomized procedures as well (see Section 5.4). The primary requirements for its use are that there be a finite number of alternatives being allocated (called *arms*), each of which has a finite number of possible outcomes (the set of outcomes on different arms need not be the same), and that all observations




Expected successes lost, for solution to two-armed Bernoulli bandit problem with horizon 100 and uniform priors on $p_1$ and $p_2$ (function is symmetric, only 1/2 shown).

Figure 1: An Example Requiring Multiple Evaluations.

---

be independent of each other. From now on, these conditions are assumed.

An example of where forward induction is useful appears in Figure 1. That figure shows the result of evaluating the expected successes lost by an allocation procedure $\mathcal{A}$ as a function of the success probabilities $p_1$ and $p_2$ for two Bernoulli arms. *Expected successes lost* is the expected number of successes that would occur if the better arm were used exclusively, minus the number that occurred using $\mathcal{A}$ (i.e., it is a form of regret). Here the sample size $n$ was 100, and the procedure $\mathcal{A}$ was created by assuming uniform distributions on $p_1$ and $p_2$ and using dynamic programming to maximize the expected successes. That is, $\mathcal{A}$ solves the *two-armed bandit with uniform discount* problem, an allocation procedure of considerable vintage [4, 6]. One of the interesting features of this situation is that $\mathcal{A}$ was created using a Bayesian design, but evaluated with respect to a non-Bayesian criterion. Other important aspects are that the criterion needed to be evaluated multiple times to produce the desired resolution, that analytical evaluations are apparently impossible, and that all calculations (including the determination of $\mathcal{A}$) took less than a minute on a Sun 10 workstation.

The implementation details, and relative advantages, of forward induction depend upon the procedure being evaluated. Forward induction will first be described in a fairly general setting in Section 2, and then specialized to fully sequential procedures (Section 3) and staged sequential procedures (Section 4). A few extensions of various forms are discussed in Section 5. This list of variations is not exhaustive, but indicates many useful approaches.

Throughout, the time and space requirements will be analyzed using standard "generalized O-notation", so that

- $f(n) = \Theta(g(n))$ means that there exist positive constants $C$, $D$, $N$ such that $Cg(n) \leq f(n) \leq Dg(n)$ for all $n \geq N$.
- $f(n) = O(g(n))$ means that there exist positive constants $D$, $N$ such that $f(n) \leq Dg(n)$ for all $n \geq N$.
- $f(n) = o(g(n))$ means that for every positive $D$ (no matter how small), there exists a positive constant $N_D$, such that $f(n) \leq Dg(n)$ for all $n \geq N_D$.

## 2 General Case

To describe forward induction, one needs the notion of the *states* of an experiment. Basically, a state is just a set of sufficient statistics describing a possible outcome (either intermediate or final) of the experiment. For example, if there are two arms, one of which is a coin and the other of which is a die, then a state is just an 8-tuple consisting of the number of heads observed, the number of tails observed, and the number of times each face of the die was observed.

In general, one wants the state space to be as concise as possible. However, in some cases one needs states with information beyond that of the sufficient statistics. For example, consider the following procedure:

> **Procedure $\mathcal{E}$:** Flip a coin three times. If the outcome is a head followed by two tails, then stop, but otherwise flip once more and stop.

In $\mathcal{E}$, the number of heads observed and number of tails observed is a sufficient statistic, but is not adequate to describe the workings of the experiment since it treats head-tail-tail the same as tail-head-tail, while the procedure treats them differently. Therefore states need to have enough information to describe the statistical outcome, and to determine the actions of the procedure. For $\mathcal{E}$, one might choose the state space to be the initial state 0 (this will be used throughout to denote the state where no observations have yet occurred), states (3,0), (2,1), (0,3), TTH, THT, and HTT representing the results of the initial flips, and states (4,0), (3,1), (2,2), (1,3), and (0,4), representing the outcomes after the fourth flip, where $(h,t)$ denotes the fact that $h$ heads and $t$ tails were observed. Of these states, HTT, (4,0), (3,1), (2,2), (1,3), and (0,4) are terminal.

In dynamic programming or backward induction, to evaluate a criteria one proceeds from the end towards the beginning of the procedure. For example, in the above experiment, suppose that the goal is find the expected number of heads observed. Let $C(\sigma)$ denote the expected value of the criteria, given that state $\sigma$ occurred. The goal, therefore, is to evaluate $C(0)$. For any terminal state $(h,t)$, $C(h,t) = h$, and for the terminal state $HTT$, $C(HTT) = 1$. Once the value is known for all immediate successors of a state $\sigma$, then it can be evaluated for that state by

$$C(\sigma) = \sum\{prob(\sigma,\sigma') \cdot C(\sigma') : \sigma' \text{ a successor of } \sigma\}, \tag{1}$$

where $prob(\sigma,\sigma')$ is the probability of reaching $\sigma'$ from $\sigma$ without passing through any other states along the way. One keeps applying this formula recursively until the initial state 0 is evaluated. This is illustrated in Figure 2, where it is assumed that a single flip of the coin has probability 0.6 of being a head.

For forward induction, one first determines the number of paths from 0 to each terminal node, where the number of paths is counted in a state model where observations occur one at a time, even

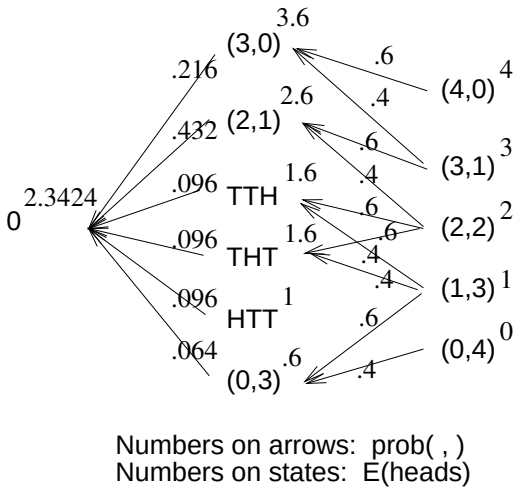



Figure 2: Backward Induction Evaluation of E(# Heads) in $\mathcal{E}$ where prob(head)=0.6

---

if the original state model does not contain all such intermediate states. Thus, for example, there are 3 paths from 0 to (2,1). Path calculations proceed in the reverse order of backward induction. Using $P(\sigma)$ to denote the number of paths to state $\sigma$, one starts with $P(0) = 1$. Once the $P$ values are known for the predecessors of state $\sigma$, its $P$ value can be determined via

$$P(\sigma) = \sum\{paths(\sigma', \sigma) \cdot P(\sigma') : \sigma' \text{ a predecessor of } \sigma\}, \tag{2}$$

where $paths(\sigma', \sigma)$ is the number of paths from $\sigma'$ to $\sigma$ that do not pass through any other states. The calculation of $P$ values for procedure $\mathcal{E}$ is shown in Figure 3, and an algorithm for the general case is given in Figure 4.

In Figure 4, the condition "if $P'(\sigma')$ evaluated" means that all predecessors of $\sigma'$ have added their contribution to $P'(\sigma')$. A similar condition, checking that all successors have contributed, would also occur in an implementation of backward induction. Usually it is trivial to determine if all successors or all predecessors of a state have been evaluated, since typically calculations proceed level by level, where the *level* of a state is the total number of observations. There might be cases where this is not appropriate, but then one could use standard traversal techniques for acyclic directed graphs to make such determinations, using space and time proportional to the number of states. (This is the same as the problem of determining a linear ordering compatible with a given partial ordering, a processes sometimes called *topological sorting*.) In the algorithm given in Figure 4, it is assumed merely that one has some efficient way of determining when all predecessors have contributed.

The second part of forward induction is to use the path counts to make evaluations. Using the notation above, this can be computed as

$$C(0) = \sum\{P(\sigma) \cdot path\text{-}prob(\sigma) \cdot C(\sigma) : \sigma \text{ terminal}\}, \tag{3}$$

where $path\text{-}prob(\sigma)$ denotes the probability that the experiment followed any single path from 0 to $\sigma$. For example, $path\text{-}prob(h, t) = p^h(1-p)^t$, where $p$ is the probability of getting a head. This

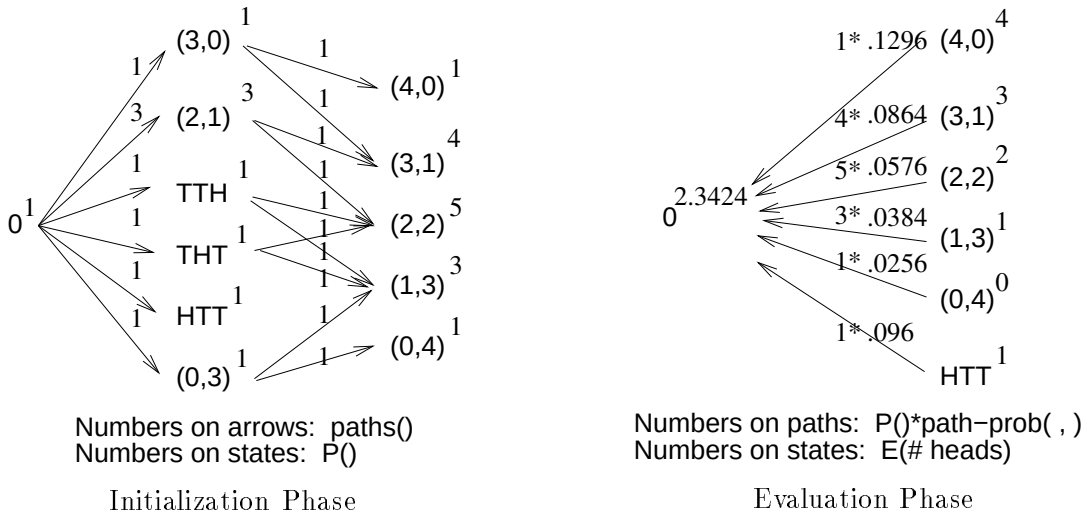



Figure 3: Forward Induction Evaluation of $\mathcal{E}$

---

is shown in Figure 3, again for the case where $C$ is the expected number of heads observed and the probability of a head is 0.6. One of the assumptions of forward induction is that the probability of a path from 0 to $\sigma$ depends only on $\sigma$, though this can be relaxed in certain circumstances (see Section 5.4).

To illustrate the use of forward induction, suppose that one wanted to plot the expected number of heads observed in $\mathcal{E}$, as a function of the probability $p$ of observing a head on a single toss of the coin (ignoring the fact that in this trivial problem the function can be determined exactly). To use backward induction, one would use Equation 1 to run through all states to compute $C$, repeating the entire process for each $p$ being evaluated. Using forward induction, one first computes $P$ by running through all the nodes using Equation 2. Then, for each $p$ value, one computes $C$ using Equation 3, which only involves terminal nodes. The advantages of forward induction include the facts that there are fewer terminal states than total states, and that only one graph traversal occurs.

Note that $path\!-\!prob(\sigma)$ can be computed from $\sigma$ alone, as can $C(\sigma)$. Throughout this paper, the time/space analyses assume that these can be computed for all terminal states in time that is proportional to the number of terminal states, perhaps with the aid of additional memory which is at most proportional to the number of terminal states. For example, for each $p$ one may compute and store $p^i$ and $(1-p)^i$ for $i = 0, \ldots, 4$ to speed up the evaluation of $path\!-\!prob$. It is also assumed that the set of successors or predecessors of any given state can be determined in time proportional to the size of the set, as can the values of $prob(\cdot,\cdot)$ and $paths(\cdot,\cdot)$.

The evaluations being performed can be frequentist, as described above, or Bayesian. In the Bayesian case $path\!-\!prob(\sigma)$, and perhaps $C(\sigma)$, would be an integral. For example, if there are $a$ Bernoulli arms, where the success parameter of the $i^{\text{th}}$ arm is modeled as a Beta random variable $\text{Be}(c_i, d_i)$, then

$$path\!-\!prob(s_1, f_1, \ldots, s_a, f_a) = \prod_{i=1}^{a} \frac{c_i^{\overline{s_i}} d_i^{\overline{f_i}}}{(c_i + d_i)^{\overline{s_i + f_i}}}$$

where $x^{\overline{k}} = x \cdot (x+1) \cdots (x+k-1)$.

for all states $\sigma$, $P(\sigma) = 0$
$P(0) = 1$
put initial state 0 in set $C$
while $C$ nonempty do {$C$ contains nonterminal states for which $P$ has been determined}
    remove an arbitrary state $\sigma$ from $C$
    for all successors $\sigma'$ of $\sigma$
        $P(\sigma') = P(\sigma') + paths(\sigma', \sigma) \cdot P(\sigma)$
        if $P(\sigma')$ completed, and $\sigma'$ nonterminal, then add $\sigma'$ to $C$
endwhile
{All terminal states now have their correct $P$ values}

Figure 4: Determining $P$ Values, General Case

---

Analyzing the time and space of repeated backward induction versus forward induction, one arrives at the following result:

**Theorem 2.1** *To perform $v$ evaluations of a deterministic procedure with $s$ states and $u$ successors per state, with $t$ of the states being terminal, takes*

| | Time | Space |
|---|---|---|
| Initialization | $\Theta(s \cdot u)$ | $\Theta(s)$ |
| Evaluation | $\Theta(v \cdot t)$ | $\Theta(t)$ |

*using forward induction, versus*

| | Time | Space |
|---|---|---|
| Evaluation | $\Theta(v \cdot s \cdot u)$ | $\Theta(s)$ |

*using simple backward induction.* □

Comparing the total time of forward induction, $\Theta(s \cdot u + v \cdot t)$, with that of backward induction, $\Theta(v \cdot s \cdot u)$, and noting that $t \leq s$ (and that all parameters are $\geq 1$), one sees that forward induction is never slower than backward induction, and in general should be far faster. Forward induction is particularly advantageous when $t \ll s$ or $u \gg 1$. Both situations are quite common in practice, and hence forward induction typically exhibits significant savings even when evaluating procedures of modest size. For example, for two Bernoulli arms and a sample size of $n$, it is shown that fully sequential procedures have $t = \Theta(s/n)$ (Section 3) and that few-stage procedures have $u = \Theta(n^2)$ (Section 4). Further, in Section 5.4, cases of random allocation are given where $u = \Theta(n^4)$. Typical applications might have $n$ in the hundreds and $v$ in the tens or hundreds, and thus forward induction can be several orders of magnitude faster.

Theorem 2.1 is quite general, but overly pessimistic for many important classes of procedures. In the following sections the classes of fully sequential and staged sequential procedures will be considered, and it will be shown that both forward and backward induction can be implemented more efficiently.

# 3 Fully Sequential Procedures

A *fully sequential* procedure is one in which allocation decisions are made one observation at a time. This includes the most efficient procedures possible, such as those found by dynamic programming. To simplify analysis and description, a fixed sample size $n$ will be assumed, and only the case of two Bernoulli arms will be considered. Extensions to other numbers and types of arms are trivial, and extensions to variable sample sizes are discussed in Section 5.2.

For two Bernoulli arms, there is a well-known approach for indexing the states and keeping the storage space small. The natural states are $(s_1, f_1, s_2, f_2)$, denoting the number of successes and failures observed on each arm. This forms an index into a 4-dimensional array, where each dimension has extent $0 \ldots n$. However, due to the constraint that $s_1 + f_1 + s_1 + f_2 \leq n$, only a corner of the array is utilized. To compute the values of $P$, it is easiest to go level by level, from level 0 to level $n$. If $m$ is the level of the states currently being examined, then $f_2 = m - s_1 - f_1 - s_2$, so one need specify only $s_1$, $f_1$, and $s_2$. This allows one to reduce to a 3-dimensional array, rather than a 4-dimensional array, where now entries are reused between levels. It is still true that only a corner of this array is being utilized, and by a slightly more complicated mapping from this corner onto a 1-dimensional array one could eliminate unused entries, reducing from $(n+1)^3$ entries to approximately $n^3/6$. However, this will not be detailed here. This approach can be used for dynamic programming, backward induction, or forward induction.

One must be a bit careful not to overwrite entries before their values have been used, and this requires that the loops for $s_1$, $f_1$, and $s_2$ in Figure 5 go in reverse order. These three loops can be nested in any order, as long as the upper limits are properly adjusted. The specific order used should be determined by the language, since cache performance is best if the innermost loop moves through items stored in adjacent locations. For Fortran that means that it should be the first index of the array, while for C it would mean the last index. The algorithm in Figure 5 assumes Fortran ordering.

Note that one simplification for fully sequential procedures is that $paths(\sigma', \sigma) = 1$ if $\sigma'$ is a successor of $\sigma$.

Once the $P$ values have been determined for the terminal states, then evaluations can proceed as before. Terminal states are those at level $n$, so there are only approximately $n^3/6$ of them, compared to the approximately $n^4/24$ total states. Assuming both forward and backward induction use the space-reduction scheme described above gives the following result:

**Theorem 3.1** *For a fully sequential procedure of sample size n, where there are two Bernoulli arms, v evaluations can be completed in*

| | Time | Space |
|---|---|---|
| Initialization | $\Theta(n^4)$ | $\Theta(n^3)$ |
| Evaluation | $\Theta(v \cdot n^3)$ | $\Theta(n^3)$ |

*by using forward induction, versus*

| | Time | Space |
|---|---|---|
| Evaluation | $\Theta(v \cdot n^4)$ | $\Theta(n^3)$ |

*by using backward induction.* □

$P(0,0,0) = 1$
do $m = 0,\ n-1$ {go through states in levels, from start to end}
    {$P$ values are correct for all states at level $m$}
    do $s_2 = m,\ 0,\ -1$
        do $f_1 = m - s_2,\ 0,\ -1$
            do $s_1 = m - s_2 - f_1,\ 0,\ -1$
                $f_2 = m - s_2 - f_1 - s_1$
                if decision$(s_1, f_1, s_2, f_2)$ = arm 1 then
                    $P(s_1+1, f_1, s_2) = P(s_1+1, f_1, s_2) + P(s_1, f_1, s_2)$
                    $P(s_1, f_1+1, s_2) = P(s_1, f_1+1, s_2) + P(s_1, f_1, s_2)$
                    $P(s_1, f_1, s_2) = 0$
                else {decision = arm 2}
                    $P(s_1, f_1, s_2+1) = P(s_1, f_1, s_2+1) + P(s_1, f_1, s_2)$
                    {$P(s_1, f_1, s_2)$ retains its value}

Figure 5: Determining $P$ for Deterministic Fully Sequential Allocation (Fortran loop ordering)

# 4 Staged Allocations

Fully sequential procedures are the most powerful, but are often rejected due to various concerns. For example, they require knowing the outcomes of previous allocations before the next allocation can be decided, and this prohibits the use of concurrent observations. Also, they are often difficult to randomize, which introduces possibilities such as selection bias [2]. Due to these concerns, investigators often prefer to use procedures that proceed in stages, where outcomes from previous stages are used to decide the number of observations on each arm for the next stage, but within a stage one can incorporate concurrency and constrained randomization. Typically the number of stages is quite small, three or fewer.

Despite the basic simplicity of such procedures, their optimization and analysis is surprisingly complicated. Some design optimization issues were addressed in [9], and analysis is addressed here. To simplify analyses, the total sample size $n$ will be fixed, and it is assumed that $k \ll n$. This is the case of interest, and by restricting things in this manner, one needn't consider cases such as $k = n$ (i.e., fully sequential allocation) which create added difficulties.

In a *1-stage allocation*, the only decision is the number of observations $n_i$ to be taken on each arm $i$, where $\sum n_i = n$. In a *$k$-stage allocation*, $k > 1$, there is an initial allocation of the number of observations $n_i^1$ on each arm $i$ during the first stage. After these observations have occurred, they can be utilized to make the allocation decisions in the remaining $(k-1)$-stage allocation with sample size $n - \sum n_i^1$. Note that stage sizes, as well as the number of observations on each arm, can depend on the accruing observations. However, often investigators prefer to know the stage sizes in advance, even if the allocation within each stage is data dependent. This restricted allocation is called *allocation with fixed stage sizes*, in contrast to the general *allocation with arbitrary stage sizes*.

One of the important differences between fully sequential and staged allocation is the number of successors of each stage. For example, for $a$ Bernoulli arms, in the fully sequential case each state has two successors, while in staged allocation, if the allocation at some state is $n_i$ observations on

arm $i$, for $i \in \{1, \ldots, a\}$, then that state has $\prod(n_i + 1)$ successors, which can be $\Theta((n/a)^a)$. As was noted in Section 2, large numbers of successors make forward induction particularly attractive. For $k$-stage allocation with two Bernoulli arms, the total number of states can be $\Theta(k \cdot n^4)$, so Theorem 2.1 shows that forward induction can be initialized in $\Theta(k \cdot n^6)$ time and evaluated in $\Theta(v \cdot n^3)$ time, versus $\Theta(v \cdot k \cdot n^6)$ time for evaluation via backward induction. While the evaluation time of forward induction cannot be reduced in the general case, the initialization time can be. To simplify discussion, for the rest of this section only the case of two Bernoulli arms will be analyzed.

First, for $k = 1$ or $k = 2$, the state space is greatly reduced. When $k = 1$, which corresponds to fixed allocation, there are only $\Theta(n^2)$ terminal stages. There is essentially no difference between backward or forward induction, with both completing $v$ evaluations in $\Theta(v \cdot n^2)$ time. For $k = 2$, the stage sizes are fixed, and while there may be $\Theta(n^3)$ terminal states, there are only $\Theta(n^2)$ states at the end of the first stage. Thus backward induction can completed in $\Theta(v \cdot n^4)$ time, using only $\Theta(n^2)$ storage, while a very simple implementation of forward induction would take $\Theta(n^4)$ time and $\Theta(n^3)$ space for initialization, and $\Theta(v \cdot n^3)$ time and $\Theta(n^3)$ space for evaluation.

To reduce the initialization time of forward induction when $k > 2$, some notation will prove useful. Let $n_i^j(\sigma)$ denote the number of observations assigned to arm $i$ when the $j^{\text{th}}$ stage starts at state $\sigma$, and let $S^j$ denote the set of states at the end of stage $j$ (equivalently, $S^j$ is the set of states at the start of stage $j + 1$), where $S^0$ is defined to be state 0. As will be shown, the savings possible will depend on whether the stage sizes are fixed or not.

Algorithms will be given showing more efficient ways to determine $P$ values. Note, however, that by reversing the order of calculation, one would also have new algorithms for performing a single evaluation of backward induction. In the algorithm analyses, *simple backward induction* refers to the process of computing $C$ for $S^k$, then $S^{k-1}$, and so on until $S^0$ is reached, where at each stage one computes $C$ for all states by using Equation 1 of Section 2. To the best of our knowledge, this is the technique used in practice, and we know of no literature showing better algorithms. An *improved implementation of backward induction* refers to one based on reversing the order of evaluation of the forward induction initiation. A single evaluation of an improved backward induction algorithm has the same time and space requirements as a forward induction initialization algorithm, except that it does not need space to store the $P$ values for the terminal states. The details of such an implementation are not given, but follow in a straightforward manner.

### 4.1 Fixed Stage Sizes

When the stage sizes are fixed, one can greatly reduce the time and space required to an amount below that given by Theorem 2.1. Let $m^j$ denote the level of the end of stage $j$, i.e., the number of observations in the $j^{\text{th}}$ stage is $m^j - m^{j-1}$, where $m^0$ is defined to be 0. At the end of the evaluation of stage $j - 1$, the $P$ values have been determined for all states in $S^{j-1}$. To determine the $P$ values for the $j^{\text{th}}$ stage, additional states are utilized, representing intermediate results. These states are identical to those used in the fully sequential case, but are used in a slightly different manner. An array $P'$ is used to compute the path counts for these states during the middle of the stage, and then at the end the counts for the states in $S^j$ are copied to $P$.

Conceptually the evaluation of each stage can be viewed as having two steps, one representing observations on arm 1, followed by one representing observations on arm 2. In the first step, for each state $\sigma = (s_1, f_1, s_2, f_2)$ in $S^{j-1}$, let $t = n_1^j(\sigma)$. Then, for all $s \in \{0, \ldots, t\}$, $\binom{t}{s} P(\sigma)$ is added to $P'(\sigma_s)$, for $\sigma_s = (s_1 + s, f_1 + t - s, s_2, f_2)$. Note that $\sigma_s$ represents starting at $\sigma$ and obtaining all

$P(0) = 1$
for $j = 1$ to $k$ do {go through stages from start to end}
{P has been determined for all states in $S^{j-1}$}
    sort $\sigma \in S^{j-1}$ by $i = n_2^j(\sigma)$ into lists $S^{j-1}(i)$
    for all states $\sigma$ of level $m^{j-1}$, $P'(\sigma) = 0$ {$P'$ used for path counts in midstage}
    for all states $\sigma \in S^{j-1}(m^j - m^{j-1})$, $P'(\sigma) = P(\sigma)$
    for $i = m^j - m^{j-1} - 1$ downto 0 do {determine $P'$ for level $m^j - i$}
        {paths in $P'$ will undergo $i + 1$ more observations on arm 2}
        $m = m^j - i$
        for $s_2 = m$ downto 0 do {add an observation on arm 2}
            for $f_1 = m - s_2$ downto 0 do
                for $s_1 = m - s_2 - f_1$ downto 0 do
                    $P'(s_1, f_1, s_2+1) = P'(s_1, f_1, s_2+1) + P'(s_1, f_1, s_2)$
                    {$P'(s_1, f_1, s_2)$ retains its value}
        $t = m - m^{j-1}$
        for all states $(s_1, f_1, s_2, f_2) \in S^{j-1}(i)$ {make $t$ observations on arm 1}
            for $s = 0$ to $t$ do
                $P'(s_1+s, f_1+t-s, s_2) = P'(s_1+s, f_1+t-s, s_2) + \binom{t}{s} P(s_1, f_1, s_2)$
    for all states $\sigma \in S^j$, $P(\sigma) = P'(\sigma)$

Figure 6: Deterministic Staged Allocation, Fixed Stage Sizes

---

$t$ of the required observations on arm 1, of which $s$ are successes, while obtaining no observations on arm 2.

After the first step is finished, only observations on arm 2 are needed. One proceeds as in the fully sequential case, updating counts from $S^{j-1}$ through $S^j$, except that at each intermediate state the updates occur as if the procedure selected arm 2. When the second step is finished, the $P$ values for all states in $S^j$ are correct.

The actual implementation, shown in Figure 6, is a bit more complicated, due to the desire to minimize space as well as time. Steps 1 and 2 are intermingled, as opposed to being in sequence. This is discussed in the proof of the following theorem.

**Theorem 4.1** *For a $k$-stage deterministic procedure with two Bernoulli arms, $k \geq 2$, with fixed sample size $n$ and fixed stage sizes, $v$ evaluations can be completed using*

| | Time | Space |
|---|---|---|
| Initialization | $\Theta(n^4)$ | $\Theta(n^3)$ |
| Evaluation | $\Theta(v \cdot n^3)$ | $\Theta(n^3)$ |

*by using forward induction, versus*

| | Time | Space |
|---|---|---|
| Evaluation ($k = 2$) | $\Theta(v \cdot n^4)$ | $\Theta(n^2)$ |
| Evaluation ($k \geq 3$) | $\Theta(v \cdot n^5)$ | $\Theta(n^3)$ |

*by using simple backward induction, or*

| | Time | Space |
|---|---|---|
| Evaluation | $\Theta(v \cdot n^4)$ | $\Theta(n^3)$ |

*by using an improved implementation of backward induction.*

Proof: For the initialization phase of forward induction, if at each stage one did step 1 and then step 2, one would need $\Theta(n^4)$ space to store the results of step 1. To reduce this to $\Theta(n^3)$, during the $j^{\text{th}}$ stage the $i$-loop of Figure 6 effectively proceeds from level $m^{j-1}$ through level $m^j$, performing step 2. As each new level $l$ is reached, path counts are added for those states which would have added their arm 1 observations to level $l$ during step 1. Thus the step 1 contributions are only added as they would have been reached. This requires creating the lists $S^{j-1}(i)$, but by using bin sorting this can be done in $\Theta(|S^{j-1}| + m^j - m^{j-1})$ time and $\Theta(m^j - m^{j-1})$ space (in addition to a pointer per state, used for the lists).

The total time of all stages for step 2 is $\Theta(n^4)$, since the step is essentially identical to the fully sequential case. For step 1 (i.e., the loops corresponding to making arm 1 observations), at stage $j$ it can take time proportional to the number of states at level $m^{j-1}$ times the length of the stage. This product is proportional to the number of states in the fully sequential model from level $m^j$ to $m^{j+1}$. Thus the time of step 1 also is at most proportional to the time of fully sequential allocation, as was claimed.

Note that the evaluation phase of forward induction is exactly the same as for the fully sequential case, and hence the time and space are as in Theorem 3.1.

For backward induction, with the exception of the time analysis of simple backward induction for $k \geq 3$, all of the analyses follow immediately from Theorem 2.1 or the observations concerning reversing forward induction. For the remaining entry, since a stage can have length $\Theta(n)$, each initial state may have $\Theta(n^2)$ successors. There can be $\Theta(n^3)$ states at the start of the stage, yielding stages taking $\Theta(n^5)$ time. Thus one might conclude that a single evaluation could be as bad as $\Theta(k \cdot n^5)$, rather than the $\Theta(n^5)$ claimed. However, this does not occur because, for $n \geq 2$, no matter how large $k$ is, the sum over all states of the number of successors is less than $n^5$, which implies that the time is $\Theta(n^5)$. This simple observation can be proven by induction. □

### 4.2 Arbitrary Stage Sizes

When the stage sizes are data dependent, there is far less information that can be exploited. One can use the basic approach employed for fixed stage sizes, but must make significant changes to account for two difficulties:

1. One can reach the same set of observations at different stages.

2. During a single stage, different paths can reach the same intermediate set of observations but continue on for different number of observations on arm 2.

Many procedures with arbitrary stage sizes exhibit neither difficulty, but to handle all possible procedures the algorithm herein must take them into account.

Because of 1, the specification of a state must incorporate stages as well as sufficient statistics. This will be done implicitly, since only the set of states corresponding to the current stage will be

$P(0) = 1$
for $j = 1$ to $k$ do
{$P$ has been determined for all states in $S^{j-1}$}
    sort $\sigma \in S^{j-1}$ by $i = n_2^j(\sigma)$ into lists $S^{j-1}(i)$
    for all states $\sigma$ of all levels, $P'(\sigma) = 0$ {$P'$ used for path counts in midstage}
    for all states $\sigma \in S_n^{j-1}$, $P'(\sigma) = P(\sigma)$
    for $i = n - 1$ downto 0 do {all paths in $P'$ need $i + 1$ observations on arm 2}
        for $m = n - 1$ downto 0 do {update $P'$ for all states using arm 2 observations}
            for all states $(s_1, f_1, s_2, f_2)$ of level $m$ do
                $P'(s_1, f_1, s_2+1, f_2) = P'(s_1, f_1, s_2+1, f_2) + P'(s_1, f_1, s_2, f_2)$
                $P'(s_1, f_1, s_2, f_2+1) = P'(s_1, f_1, s_2, f_2+1) + P'(s_1, f_1, s_2, f_2)$
        for all states $\sigma = (s_1, f_1, s_2, f_2) \in S^{j-1}(i)$ {add arm 1 observations from $\sigma$}
            $t = n_1^j(\sigma)$
            for $s = 0$ to $t$ do
                $P'(s_1+s, f_1+t-s, s_2, f_2) = P'(s_1+s, f_1+t-s, s_2, f_2) + \binom{t}{s} P(s_1, f_1, s_2, f_2)$
    for all states $\sigma \in S^j$, $P(\sigma) = P'(\sigma)$

Figure 7: Deterministic Staged Allocation, Arbitrary Stage Sizes, $k \geq 4$

represented at any one time. This allows storage to be reused, just as the fact that fully sequential evaluation proceeds level by level allows one to reuse space there. However, since essentially any set of sufficient statistics might be the outcome (or intermediate state) of a given stage, the space must accommodate all possible states, not just those at a specified level.

Next, because of 2, during a stage it is not possible to do just a single pass through the levels of the states, making arm 2 observations for all states and adding arm 1 observations when their level is reached. Instead, the $i$-loop in Figure 7 counts down through the number of arm 2 observations needed, which can be arbitrary and which are made for all stages at each iteration, not just the stages at a given level. For a state $\sigma$ in $S^{j-1}$, the arm 1 observations from $\sigma$ are added to the path counts when $i = n_2^j(\sigma)$. Thus each stage takes $\Theta(n^4)$ space, and $\Theta(n^5)$ time.

Figure 7 represents the general case, with $k \geq 4$. If $k = 1$ or $k = 2$, then the procedure must have fixed stage sizes, and thus the results of Section 4.1 apply. The case of $k = 3$ is intermediate between the fixed stage size and arbitrary stage size situations since each stage either has a fixed beginning or a fixed ending. This can be exploited, as is shown in the following theorem.

**Theorem 4.2** *For a $k$-stage deterministic procedure with two Bernoulli arms, $k \geq 3$, with fixed sample size $n$ and arbitrary stage sizes, $v$ evaluations can be completed using*

| | Time | Space |
|---|---|---|
| Initialization ($k = 3$) | $\Theta(n^4)$ | $\Theta(n^4)$ |
| Initialization ($k \geq 4$) | $\Theta(k \cdot n^5)$ | $\Theta(n^4)$ |
| Evaluation | $\Theta(v \cdot n^3)$ | $\Theta(n^3)$ |

*by using forward induction, versus*

| | Time | Space |
|---|---|---|
| Evaluation | $\Theta(v \cdot k \cdot n^6)$ | $\Theta(n^4)$ |

*by using simple backward induction, or*

| | Time | Space |
|---|---|---|
| Evaluation ($k = 3$) | $\Theta(v \cdot n^4)$ | $\Theta(n^4)$ |
| Evaluation ($k \geq 4$) | $\Theta(v \cdot k \cdot n^5)$ | $\Theta(n^4)$ |

*by using a more sophisticated implementation of backward induction.*

Proof: For forward induction, for $k = 3$, a straightforward implementation can determine the path counts for all states in $S^1$ in $\Theta(n^2)$ time, and for all states in $S^2$ in $\Theta(n^4)$ time and space. To finish the path counts for the last stage, an approach similar to that used for the fixed stages can be used. Because $k = 3$ and the total sample size is fixed, $n_2^3(\sigma)$ determines the level from which only arm 2 observations are added until the end of the experiment, no matter what level the stage 2 state $\sigma$ was on. This is almost the same as the information for fixed stage sizes. Thus one can proceed as in the fixed stage size case, going level by level towards level $n$. However, when arm 1 observations from state $\sigma$ are added to the intermediate path counts, the number of such observations is determined by $n_1^3(\sigma)$, not by the level at which they are added. The intermediate space needed is again only $\Theta(n^3)$, but the total space is $\Theta(n^4)$ because that is how many states there may be in $S^2$.

For $k \geq 4$, the analysis follows by the comments preceding the theorem. □

The algorithm in Figure 7 was written to emphasize conciseness, and does not incorporate the simplifications for the first, second, and last stages mentioned in the proof of the theorem. While this does not affect the O-notational time and space for $k \geq 4$, in practice one may notice significant savings by incorporating such simplifications.

# 5 Extensions

Beyond those mentioned above, there are a variety of refinements and extensions possible for forward induction. In this section, a few of the more important ones will be considered.

## 5.1 State Reduction

In practice, one of the most useful refinements is the elimination of unreachable states. For example, for the fully sequential strategy known as "Play the Winner" [12, 15] (or, more accurately, "Play the Winner/Switch on Loser") for two Bernoulli arms, if a success occurs on one arm then the arm is repeated, while if a failure occurs then the other arm is tried. For this strategy, the number of failures on the two arms cannot differ by more than one, and hence there are only $\Theta(m^2)$ possible states at level $m$, rather than the $\Theta(m^3)$ possible in the general case. By incorporating this knowledge, the initialization and evaluation algorithms can be improved by a factor of $n$. Additional state reductions can occur if there are stopping rules, as discussed in the following section.

## 5.2 Stopping Rules

Often one needs to be able to evaluate allocations with variable sample sizes, rather than the fixed sample sizes analyzed above. This can occur for both multiarm and single arm designs (without optional stopping, single arm procedures are just fixed allocations). The use of stopping rules

can easily be incorporated into the algorithms of the proceeding sections, though they may affect the time and space analyses. For example, while it seems that all practical stopping rules have $O(n^3)$ terminal states, there exist peculiar ones, such as "stop if both arms have an even number of successes, or level $n$ has been reached", that have $\Theta(n^4)$ terminal states. Since the time of the evaluation phase of forward induction is proportional to the number of terminal states, this implies that there are (unuseful) procedures that will take $\Theta(v \cdot n^4)$ time for evaluation.

In some cases, stopping rules can reduce the number of terminal states to $o(n^3)$, which, when coupled with the state reduction approach mentioned above, results in faster evaluation. For example, suppose one is evaluating Vector at a Time allocation (taking one observation on each arm) and a stopping rule that halts if the number of successes on one arm is $r$ more than on the other arm, or if level $n$ is reached (see [3, 7]). Here the number of terminal states is only $\Theta(r \cdot n)$, permitting dramatically faster evaluation by forward induction.

### 5.3 Expanding Horizons

One unusual feature of forward induction, as compared to backward induction or dynamic programming, is that it can occasionally be used efficiently to help design allocation procedures where the sample size is not known in advance. For example, one may have some criteria, such as a specified level of power or a specified width of a confidence interval, that one is trying to attain by adjusting the sample size. Often this is done by evaluating successively larger sample sizes until a size having the desired criteria is reached. The simplest way to do this is to start over for each larger sample size, but this may be quite time-consuming, especially if each sample size requires multiple evaluations.

Fortunately, many sequential allocation rules, such as Alternating Allocation, Play the Winner (Section 5.1), Randomized Play the Winner (Section 5.4), Vector at a Time (Section 5.2), Thompson's rule [13], and most myopic rules, have the property that the allocation decision at a given state is independent of the sample size. For such rules, calculations of $P$ for small sample sizes can be continued for larger sample sizes, so that one need not start over for each new sample size. In the general case, to determine $P$ for a horizon-independent rule at sample sizes $n_1 < n_2 < \ldots < n_t$ takes only $\Theta(n_t^4)$ time, as opposed to the $\Theta(\sum_{i=1}^{t} n_i^4)$ time that would be required if one started anew for each larger sample size.

Neither backward induction nor dynamic programming seem capable of expanding from small sample horizons to larger ones, and thus the relative advantage of forward induction is magnified in such settings.

### 5.4 Random Allocation

For some allocation procedures, the successors of a given state are not deterministic. This requires some adjustments, because the paths to a given state need not all have the same probability, and the number of successors of a given state may be greatly enlarged.

To illustrate the former effect, in a simple version of Randomized Play the Winner [14], there is an urn with balls marked "arm 1" and "arm 2". A ball is randomly selected and returned to the urn, and that arm is tried. If the arm succeeds, then another ball of the same type is added to the urn, while if it fails then a ball of the opposite type is added. In this model, if the urn starts with 1 ball of each type, then the state (1,0,0,1) can be reached via an arm 1 ball followed by an arm 2 ball, or via arm 2 and arm 1. In the former case the probability is $\frac{1}{2} \cdot p_1 \cdot \frac{1}{3} \cdot (1 - p_2)$, while in the

latter case it is $\frac{1}{2} \cdot (1 - p_2) \cdot \frac{2}{3} \cdot p_1$. The unequal probabilities can be dealt with by adjusting the path updating rules to account for the extra source of randomization. At state $\sigma$, if there is probability $q(\sigma, i)$ of trying arm $i$, then $q(\sigma, i)P(\sigma)$ is added to the states representing an additional success or failure on arm $i$.

The expansion in the number of successors is also straightforward to accommodate, but it can greatly increase the time required to analyze staged allocation. (The time for randomized fully sequential allocation with $a$ arms is at most $a$ times larger than for the deterministic case). For example, for two Bernoulli arms, if the first stage size is fixed at $m$, but the number of observations on arm 1 is a random function varying from 0 to $m$, then all $\Theta(m^3)$ states at level $m$ are successors of the initial state 0, as opposed to the $\Theta(m^2)$ successors that occur for deterministic allocation. If the stage length is also random in $0 \ldots m$, then the number of successors can reach $\Theta(m^4)$. These increases in the number of successors cause corresponding increases in the time required.

Note that when the stage length is random then one can encounter situations where $\tau$ is an immediate successor of $\sigma$, and there is a state $\sigma'$ which is an immediate successor of $\sigma$ and an immediate predecessor of $\tau$. Such situations cannot exist when deterministic allocation is used, but in this more general setting one must be careful to define $prob(\sigma, \tau)$ and $paths(\sigma, \tau)$ with respect to reaching $\tau$ in a single stage, without passing through $\sigma'$.

### 5.5 Switching Costs

One of the assumptions in the above analyses is that the criterion to be evaluated is determined by the terminal state. However, if the natural states are used, then there are criteria such as switching costs which cannot be evaluated. In a basic *switching cost model* [1, 11], there is a fixed cost every time the experiment switches from one arm to another. For example, in a two-armed Bernoulli experiment, one might reach state (2,0,2,0) via 1, 2, or 3 switches, and the costs would vary correspondingly. Thus the natural state space is inadequate to determine the costs.

To evaluate such criteria, the state space needs to be expanded, much as it is expanded to handle allocation procedures that depend on the path taken to a given state. One natural approach is to add to each state information concerning the arm most recently tried and the number of switches that have occurred so far. Thus the (2,0,2,0) state might be replaced by the six states

(2,0,2,0: arm 1, 1) (2,0,2,0: arm 1, 2) (2,0,2,0: arm 1, 3)
(2,0,2,0: arm 2, 1) (2,0,2,0: arm 2, 2) (2,0,2,0: arm 2, 3).

The resulting evaluations for forward or backward induction would be straightforward, but would suffer because the state space has been greatly expanded. For example, for a fully sequential two-armed Bernoulli experiment of sample size $n$, there would be $\Theta(n^5)$ states, with $\Theta(n^4)$ of them being terminal. This would increase the time and space requirements by a factor of $n$.

A better approach for evaluating mean switching costs, which only expands the time and space by constant factors, is to add the information about the arm most recently tried to each state, and to keep two path counts. One path count, $P$, is as before, and the other, $P_w$, is a weighted path count, where each path is multiplied by the number of switches that occurred. Thus, at any state $\sigma$, $P_w(\sigma)/P(\sigma)$ is the average number of switches occurring on a path from 0 to $\sigma$. To determine $P_w$, suppose one is at state $\sigma_i$ and is updating information for a successor $\tau_j$, where the subscript indicates the arm most recently tried. Then

$$P_w(\tau_j) = \begin{cases} P_w(\tau_j) + paths(\sigma, \tau) \cdot P_w(\sigma_i) & \text{if } i = j \text{ i.e., no switch occured} \\ P_w(\tau_j) + paths(\sigma, \tau) \cdot (P_w(\sigma_i) + P(\sigma_i)) & \text{otherwise} \end{cases}$$

# 6 Final Remarks

Forward induction is an aid to evaluating sequential procedures and complements other approaches such as exact or asymptotic analyses, backward induction, and dynamic programming. It is fairly flexible, and is suitable for a variety of procedures (frequentist/Bayesian, fully sequential/staged sequential, fixed sample size/optional stopping, deterministic/randomized) and a variety of evaluation criteria. Several of these have been discussed here, along with various algorithms optimized for the worst case. In many situations, additional information about a procedure can be used to significantly improve upon the worst-case analyses given here. All worst-case analyses assumed that the maximum possible number of states needed to be considered, but often this is not true. Any major reduction in the number of states examined will allow a major reduction in the time or space required.

Depending on the procedure and sample size, forward induction can be orders of magnitude faster than previous exact computational approaches for performing multiple evaluations of a procedure. For example, for several years we have been able to use forward induction on workstations [8] to perform calculations that had previously been done using backward induction on supercomputers [5]. This speed encourages more extensive analysis and visualization of designs, helping users achieve better optimizations and tradeoffs among multiple criteria.

As another example, the computational results in [10] could only be carried out for $n = 25$, while concurrently the authors obtained the results in [8] for $n = 150$. These results involve nearly identical fully sequential models, though the latter paper included criteria that required approximately 100 evaluations per procedure, while the former required only a couple of evaluations per procedure. (Note that the algorithms used in [10] were quite standard, and our only reasons for singling out this paper are the fact that the results can be directly compared, and it explicitly noted that the sample size was limited by computational constraints.) In terms of the algorithms used in [10], the work in [8] solved problems approximately $(150/25)^4 \cdot 100 \approx 100,000$ times harder. While a factor of 10 or so advantage was obtained by using a workstation (a modest Sun 3/60) as opposed to a (presumed) personal computer, and some advantage was obtained by having an implementation that tried to maximize processor efficiency, most of the advantage came from using a far more efficient algorithm, namely forward induction instead of backward induction.

Finally, although the emphasis has been on showing the utility of forward induction over backward induction whenever multiple evaluations are required, we also show how to improve the implementation backward induction for analyzing staged allocation procedures. The algorithms for determining $P$ values appearing in Section 4 can be reversed to yield algorithms for backward induction. These algorithms are faster than standard implementations by a factor of $n$ or more, and are apparently superior to any previously appearing in the literature. However, it will still be true that one should use forward induction, instead of the improved backward induction, if multiple evaluations are required.

## References


[1] Agrawal, R., Hegde, M.V., and Teneketzis, D. (1988), "Asymptotically efficient adaptive allocation rules for the multiarmed bandit problem with switching costs", *IEEE Trans. Auto. Control* **33**, pp. 899–906.

[2] Bather, J. (1995), "Response adaptive allocation and selection bias", *Adaptive Designs*, N. Flournoy and W.F. Rosenberger, eds., IMS Lecture Notes-Monograph Series **25**, pp. 23–35.

[3] Bechhofer, R.E., Kiefer, J., and Sobel, M. (1968), *Sequential Identification and Ranking Procedures*, Univ. Chicago Press.

[4] Bellman, R. (1961), *Adaptive Control Processes: A Guided Tour*, Princeton Univ. Press.

[5] Berry, D.A. and Eick, S.G. (1995), "Adaptive assignment versus balanced randomization in clinical trials— a decision-analysis", *Stat. in Medicine* **14**, pp. 231–246.

[6] Bradt, R.N., Johnson, S.M. and Karlin, S. (1956), "On sequential designs for maximizing the sum of $n$ observations", *Ann. Math. Stat.* **27**, pp. 1060–1074.

[7] Buringer, H., Martin, H. and Schriever, K.H. (1980), *Nonparametric Sequential Selection Procedures*, Birkhauser.

[8] Hardwick, J. and Stout, Q.F. (1991), "Bandit strategies for ethical sequential allocation", *Computing Science and Statistics* **23**, pp. 421–424.

[9] Hardwick, J. and Stout, Q.F. (1995), "Few-stage allocation rules", *Computing Science and Statistics* **27**, to appear.

[10] Jones, P.W. (1992), "Multiobjective Bayesian bandits", *Bayesian Statistics 4*, J.M. Bernardo, J.O. Berger, A.P. Dawid, and A.F.M. Smith, eds., Oxford Univ. Press.

[11] Kolonko, M. and Benzing, H. (1985), "The sequential design of Bernoulli experiments including switching costs", *Operations Research* **33**, pp. 412–426.

[12] Robbins, H. (1952), "Some aspects of the sequential design of experiments", *Bull. Amer. Math. Soc.* **58**, pp. 527–535.

[13] Thompson, W.R. (1933), "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples", *Biometrika* **25**, pp. 275–294.

[14] Wei, L.J. and Durham, S. (1978), "The randomized play the winner rule in medical trials", *J. Amer. Statist. Assoc.* **73**, pp. 840–843.

[15] Zelen, M. (1969), "Play-the-winner rule and the controlled clinical trial", *J. Amer. Statist. Assoc.* **64**, pp. 131–146.