In *SIAM Journal on Scientific Computing* **21** (1999), pp. 67–87.

Using Path Induction to Evaluate Sequential Allocation Procedures

Janis P. Hardwick
Statistics Department

Quentin F. Stout
EECS Department

University of Michigan, Ann Arbor, MI 48109

Abstract: Path induction is a technique used 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 parameter 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 standard evaluation techniques. 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 or ad hoc approaches), 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 path 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, path induction will still be far superior. For each parameter configuration to be evaluated, one reduces the time by a factor of n, where n is the size of the experiment, by using path induction rather than the standard technique of backward induction. In some settings the savings is significantly greater than n.

Keywords: backward induction, adaptive allocation, stage, group sampling, path counting, forward induction, Bayesian, bandit problems, stochastic optimization, design of experiments, machine learning

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

Last modified: 13 Aug 1999.

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

1 Introduction

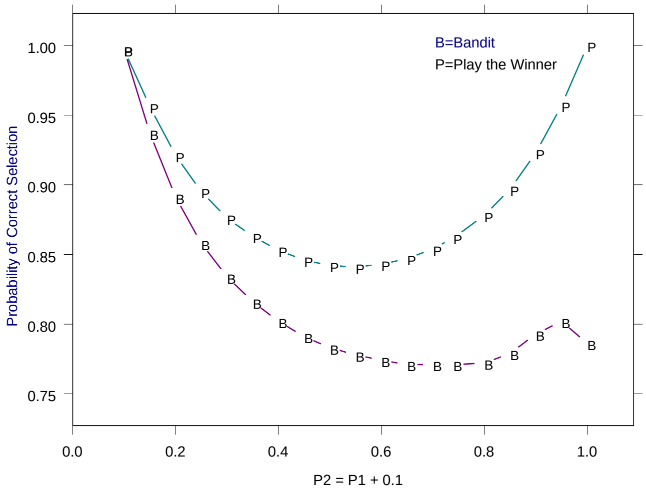
Sequential allocation procedures use accruing data to control their decisions, rather than making all decisions in advance of any data collection. For example, allocating jobs to servers based on observed queue lengths is a sequential allocation procedure. Another important example, which motivates the work in this paper, is the use of sequential allocation procedures to determine how best to assign patients to treatments in a clinical trial. Conventional clinical trial designs use fixed allocation schemes in which, for example, half of the patients are assigned to each of two treatments. Such methods, while excellent from a statistical point of view, may interfere with other trial criteria such as minimizing losses to patients. Sequential procedures allow researchers to address a variety of trial goals that would be impossible to accommodate under fixed allocation schemes.

To keep the example simple, let us assume that each patient response is an independent Bernoulli random variable, with outcomes either "live" or "die". Next, suppose that we want to design a trial in which we gather sufficient experimental information to arrive at a terminal decision regarding the best treatment while simultaneously reducing the average number of patients who die during the experiment itself. Let T_1 and T_2 be two therapies each having an unknown probability p_i, $i = 1, 2$, that any given patient will live or die. It is assumed that the patient responses are independent both of the therapies to which they are assigned and of one another. If we are allowed to utilize patient outcomes as the trial ensues, then our ability to direct more patients to the apparently better therapy increases over time. Thus, a goal such as minimizing patient deaths is now addressable. Similarly, in industrial settings one may utilize adaptive allocation procedures to address estimation problems where there are also cost considerations that vary with the options or outcomes.

Unfortunately, while the adaptive features of sequential procedures make them more flexible and, often, more efficient than fixed allocation procedures, they also make them more complex to analyze and optimize. Due to analytical difficulties, sequential procedures are often overlooked, even when they can offer substantial benefits. Fortunately, advances in computer technology are making computational analyses more practicable 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 toward the front, can be used for optimization and evaluation, respectively, when the options and outcomes are finite.

Still, there are many situations that remain computationally challenging. For example, suppose that in our clinical trial experiment, we have determined how to allocate the patients to minimize the expected number of deaths, and we now want to evaluate a characteristic of the procedure such as the probability that we will correctly identify the better therapy at the termination of the trial provided that the two success probabilities differ by at least $\delta > 0$. This problem combines the goals of two well-known design problems as well as both frequentist and Bayesian viewpoints. The first problem is the finite horizon, Bernoulli *two-armed bandit* problem **(2-AB)**, an allocation procedure of considerable vintage [4, 6]. In this problem one samples sequentially from either of two dichotomous populations ("arms") in an attempt to maximize the number of successes garnered after having taken a total of n observations. Bandit problems are typically approached from a Bayesian framework, that is, one in which it is assumed that the unknown parameters are themselves random variables. The distributions for the parameters are referred to as being *prior* distributions, and in the present problem it's quite common, at least initially, to assume that these distributions are uniform.

In this scenario, the problem that is combined with the bandit problem is a classical *best selection* problem, in which one seeks to optimize the probability of correctly selecting the population with the higher success probability (the *probability of correct selection* (P(CS))). The P(CS) is based on having observed a total of n responses and the assumption that the probabilities differ by at least δ; see [3] for more details



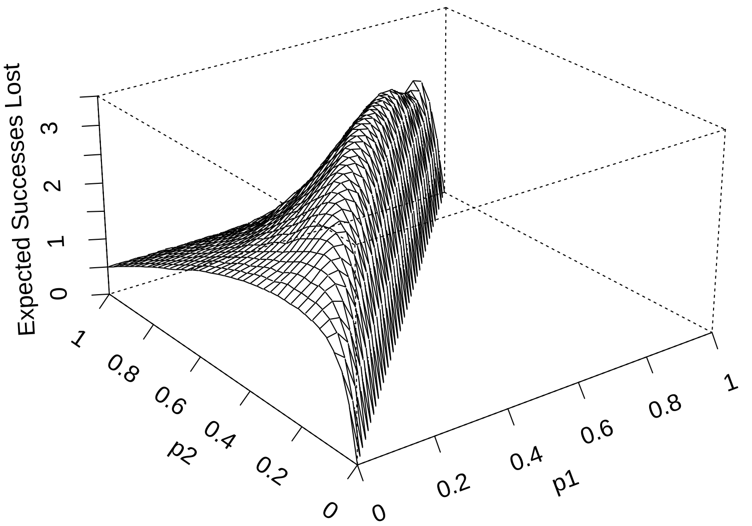
Sample size n=100, bandit prior distributions are uniform.

Figure 1: An Example Requiring Multiple Evaluations.

---

on these problems. In this best choice problem, it is known that one is least likely to make a correct selection when the success probabilities differ by exactly δ. However, in general, it is not known which pair of probabilities achieves the minimum. Thus, it is necessary to evaluate some collection of success probabilities until the minimum has been located to sufficient accuracy. One curve in Figure 1 represents the P(CS $\mid p_1, p_2$) for the optimal 2-AB strategy as p_1 and p_2 vary, subject to $p_2 - p_1 = 0.1$. The other curve in Figure 1 represents P(CS $\mid p_1, p_2$) for the simple ad hoc scheme "play the winner" (see Section 6.1). Note that each point in these plots represents a new evaluation of an allocation procedure.

Furthermore, note that the measure represented by the 2-AB curve is a frequentist analysis of a Bayesian bandit design. Had the P(CS $\mid p_1, p_2$) been integrated with respect to the prior distributions of p_1, p_2 or both, then the design and the analysis parts of the problem would be considered Bayesian. We emphasize the difference between design and analysis assumptions for a couple of reasons. First it shows that investigators can analyze their experimental data independently of the Bayesian assumptions used to generate the optimal design. Note that the design being evaluated may also be based on an ad hoc approach. Second, frequentist analyses of sequential designs often require very different computational approaches than do Bayesian analyses. While it has occasionally been possible to perform the sort of multiple evaluations needed for Figure 1 by using exact analytical expressions, this is quite rare and has tended to limit investigators to using simple procedures that are optimized for analytical tractability rather than for usefulness. Historically, 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 severely limits the number of evaluations attempted and/or the size of experiment that researchers can analyze. Typically, in fact, investigators will carry out only a single evaluation and obtain only a single output such as the minimum risk achievable when following an optimal strategy. This information, by itself, is of limited use in practice. It's true that such outputs are useful in allowing investigators to evaluate how close specific procedures are to the optimum and also to evaluate how close asymptotic approximations are to the true value. On the other hand, bringing adaptive techniques into practical use requires that researchers gain a significantly better understanding of the characteristics not only of the optimal procedures but also of the suboptimal procedures that may have



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

Figure 2: An Example Requiring Multiple Evaluations.

---

other features to recommend them. This is where *path induction*, a far more efficient algorithm, comes into play. Path induction supplies a method for efficiently obtaining multiple evaluations of a procedure, independent of the method used in generating the procedure. This allows us to rapidly compute the information shown in Figure 1.

Another example of an operating characteristic made readily accessible by path induction is given in Figure 2, where we show a surface plot of the expected successes lost as p_1 and p_2 vary over the parameter space, for the optimal solution to the 2-AB problem. *Expected successes lost* refers to the expected number of successes that would occur if the better arm were used exclusively, minus the number that occurred using the procedure. Issues related to the generation of Figure 2 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 the solution to the 2-AB) took less than one minute.

Unlike some other forward techniques, path induction consists of two phases: initialization and evaluation. The initialization phase works from the front of the experiment toward 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. Because only the evaluation phase needs to be reiterated when evaluating a procedure for a variety of different parameter configurations, path induction requires significantly fewer calculations than forward methods that are direct counterparts of backward induction. Such methods require repeating the entire process each time a new parameter configuration is evaluated.

In earlier work, we have referred to the technique of path induction as *forward induction*, an appropriate name but one that we have found can cause confusion. There are a number of scenarios that call for forward algorithmic procedures, any of which may reasonably be called forward induction. For example, Gittins's procedure for solving certain multiarmed bandit problems has also been termed forward induction [8]. This is a procedure in which a dynamic allocation index for each arm is calculated at each stage and the next experiment chosen is the one corresponding to the highest allocation index. Gittins's technique was a major breakthrough in the theory of bandit problems since it defined scenarios in which one could obtain optimal solutions by moving forward rather than backward through the state space. "Forward induction" arguments are also sometimes used to prove that various adaptive algorithms, such as queuing algorithms, make optimal choices. Similarly, there is a variety of other algorithmic techniques that utilize path computation and state indexing. For example, the *network algorithms* used for computing exact distributions in the statistical package STATXACT use paths and states [13]. However, these algorithms are quite different from those discussed here. A simulation study is yet another forward process, and in an effort to decrease the variance of a simulation or Monte Carlo study, some researchers utilize an initial forward induction stage to get the exact probability of reaching each state near the beginning, and take it on from there with the simulation study. This variance-reduction approach reduces the time needed to achieve a given accuracy. Still, it does not have the time-saving features of the path induction approach, since each change in parameters requires redoing the forward process.

Path induction is exact and applicable to a wide range of procedures, analyses and 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 6.4). 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 be independent of each other. Throughout this paper, these conditions are assumed.

The implementation details, and relative advantages, of path induction depend upon the procedure being evaluated. Path 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 5). A few extensions of various forms are discussed in Section 6. 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 path 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 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:

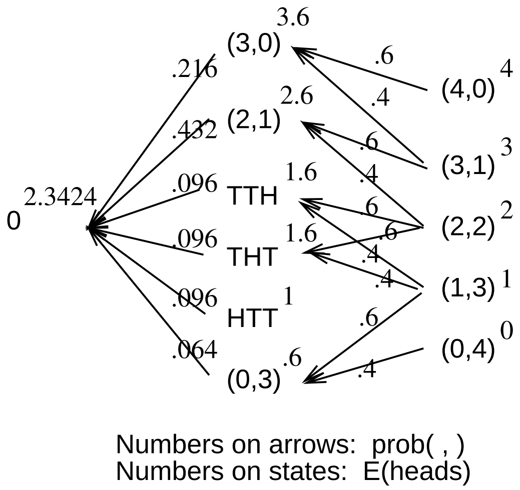


Figure 3: Backward Induction Evaluation of E(# Heads) in $\mathcal{E}$, Where prob(head)=0.6

> **Procedure $\mathcal{E}$:** Flip a coin three times. If the outcome is a head followed by two tails, then stop; 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 this 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 been taken), states (3,0), (2,1), (0,3), TTH, THT, and HTT to represent the results of the initial flips, and states (4,0), (3,1), (2,2), (1,3), and (0,4) to represent 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.

To evaluate a criteria via dynamic programming or backward induction, one proceeds from the end towards the beginning of the procedure. For example, in the above experiment, suppose that the goal is to find the expected number of heads observed. Let $C(\sigma)$ denote the expected value of the criteria, given that state σ 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 σ, 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 σ' from σ 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 3, where it is assumed that a single flip of the (biased) coin has probability 0.6 of being a head.

For path 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 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 σ, one starts with $P(0) = 1$. Once the P values are known for the predecessors of state σ, its

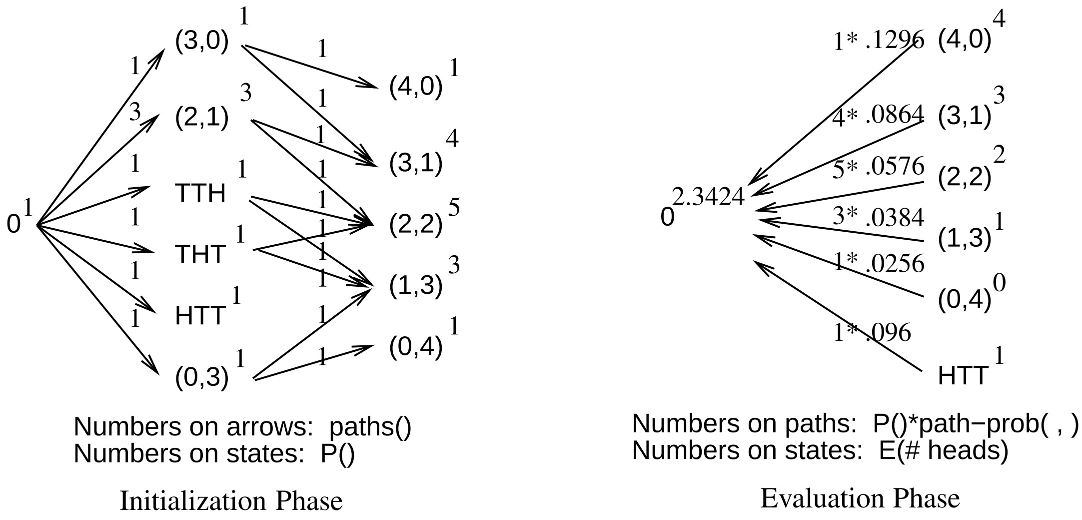


Figure 4: Path Induction Evaluation of $\mathcal{E}$

own 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 σ' to σ that do not pass through any other states. The calculation of P values for procedure $\mathcal{E}$ is shown in Figure 4, and an algorithm for the general case is given in Figure 5.

In Figure 5, the condition "if $P(\sigma')$ evaluated" means that all predecessors of σ' 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 5, it is assumed merely that one has some efficient way of determining when all predecessors have contributed.

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

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

where $pathprob(\sigma)$ denotes the probability that the experiment followed any single path from 0 to σ. For example, $pathprob(h,t) = p^h(1-p)^t$, where p is the probability of getting a head. This is shown in Figure 4, 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 path induction is that the probability of a path from 0 to σ depends only on σ, although this can be relaxed in certain circumstances (see Section 6.4).

To illustrate the use of path 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

for all states σ, $P(\sigma) = 0$
$P(0) = 1$
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 σ from C
    for all successors σ' of σ
        $P(\sigma') = P(\sigma') + paths(\sigma', \sigma) \cdot P(\sigma)$
        if $P(\sigma')$ completed, and σ' nonterminal, then add σ' to C
endwhile
$\{$All terminal states now have their correct P values$\}$

Figure 5: Determining P Values, General Case

---

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. One could also use a forward induction process which fixes a value of p and goes from the start state to the terminal states, finding the exact probability of reaching each state. As in backward induction, this entire process would be repeated for each value of p. Using path 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 involves only terminal nodes. The advantages of path induction include the facts that there are fewer terminal states than total states and that only a single graph traversal occurs.

Note that $pathprob(\sigma)$ can be computed from σ 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 $pathprob$. 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 $pathprob(\sigma)$, and perhaps $C(\sigma)$, would be an integral. For example, recall that a random variable x has a *beta* distribution with parameters (a, b) if x has the following density function:

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} \quad \text{for} \quad 0 < x < 1 \quad \text{and} \quad a, b > 0.$$

Note that when $(a, b) = (1, 1)$, the beta distribution is simply a uniform distribution. Now, suppose that we have a Bernoulli arms and that the success probability for the i^{th} arm is modeled as beta with parameters (c_i, d_i). Then

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

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

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 path induction, versus

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

using simple backward induction.

Proof: For a single pass through the graph (for either backward or path induction), there are $s \cdot u$ edges, each of which requires a constant amount of work. The initialization of path induction requires only one such pass, but backward induction will require v passes, which gives the time bounds. As for space bounds, at most a constant amount of space per state is needed for a pass through the graph (beyond the space, if any, needed to store the graph itself), and this space is reused if multiple passes are needed. For the evaluation phase of path induction, only a constant amount of space per terminal state is needed. These space and time analyses are based on the assumptions mentioned earlier concerning the computational requirements for $paths$, $prob$, and $pathprob$. □

Comparing the total time of path 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 ≥ 1), one sees that path induction is never slower than backward induction and in general should be far faster. Path induction is particularly advantageous when $t \ll s$ or $u \gg 1$. Both situations are quite common in practice, and hence path 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 5). Furthermore, in Section 6.4 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 path 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 path 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 6.2.

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 $f_2 = m - s_1 - f_1 - s_2$; thus 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 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 path 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 6 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 6 assumes Fortran ordering.

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

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 path 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 path induction, versus

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

by using backward induction. □

4 Example of Robustness Analysis Using Path Induction

Since procedures resulting from the assumptions imposed by a Bayesian design are directly tied to the choice of prior distribution, it is useful to be able to evaluate the impact of the initial assumptions on the data analysis process. We noted in Section 1 that uniform distributions are often assumed as priors of the parameters of Bernoulli random variables. One can assess the "robustness" of such an assumption by re-evaluating the procedure using several different priors. If important characteristics of the procedure remain virtually unchanged, then the initial choice of prior would not be deemed a sensitive design parameter. This is useful because results obtained using Bayesian designs are often criticized as being overly dependent on the assumptions in the prior distribution. Here we show how path induction can be used to help appraise the robustness of the prior assumptions, and we apply the approach to a nonlinear estimation problem.

```
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 s2 = m, 0, −1
        do f1 = m − s2, 0, −1
            do s1 = m − s2 − f1, 0, −1
                f2 = m − s2 − f1 − s1
                if decision(s1, f1, s2, f2) = arm 1 then
                    P(s1+1, f1, s2) = P(s1+1, f1, s2) + P(s1, f1, s2)
                    P(s1, f1+1, s2) = P(s1, f1+1, s2) + P(s1, f1, s2)
                    P(s1, f1, s2) = 0
                else {decision = arm 2}
                    P(s1, f1, s2+1) = P(s1, f1, s2+1) + P(s1, f1, s2)
                    {P(s1, f1, s2) retains its value}
```

Figure 6: Determining P for Deterministic Fully Sequential Allocation (Fortran loop ordering)

In this problem, we wish to estimate the mean of some polynomial function of the parameters of two Bernoulli populations. (The polynomial used in Figure 7 is the product $p_1 * p_2$.) Problems of this nature may arise, for example, when estimating the fault tolerance of a system by testing its components individually. Since it is rarely optimal to test an equal number of components from each population, the question of how to gather a sample of fixed size n arises. Here, as in the clinical trial example, we assume that population parameters, p_1 and p_2, are independent beta random variables. Suppose that we take $\theta(p_1, p_2)$ to be a polynomial function of the population parameters and we use the Bayes estimator (i.e., the posterior mean of θ given the data), $\hat{\theta}$, to estimate θ, then our remaining problem is to determine how to sample from the different populations to minimize the integrated mean squared error (or *Bayes risk*) of $\hat{\theta}$.

As in the 2-AB example, the optimal sampling procedure for this problem can be generated via dynamic programming. However, the evaluation of operating characteristics of the procedure, such as its sensitivity to the design parameters, are better carried out using path induction. This type of evaluation involves two sets of prior parameters: the *design* or $\mathcal{D}$–parameters $= [(\mathcal{D}_1, \mathcal{D}_2); (\mathcal{D}_3, \mathcal{D}_4)]$ and the *analysis* or $\mathcal{A}$–parameters $= [(\mathcal{A}_1, \mathcal{A}_2); (\mathcal{A}_3, \mathcal{A}_4)]$, where the subscripts 1 and 2 refer to the beta parameters for arm one and 3 and 4 refer to the beta parameters for arm 2. The efficiency analysis goes as follows:

(1) Generate the optimal procedure for the $\mathcal{D}$–parameters.

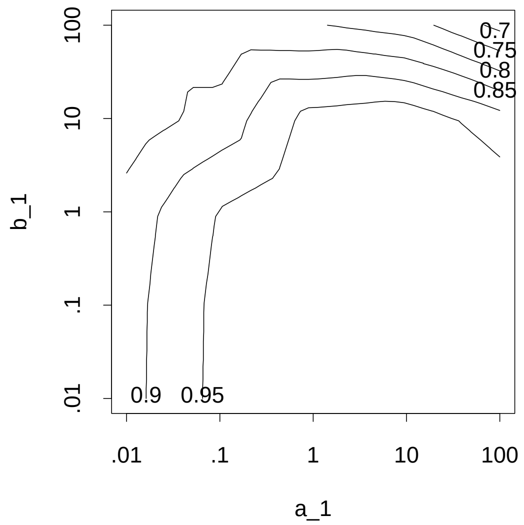
(2) For each $\mathcal{A}$–parameter configuration,

  (2.1) compute the operating characteristic (in this case, the Bayes risk) of the procedure from (1), relative to the $\mathcal{A}$–parameter.

  (2.2) Generate the optimal procedure for the $\mathcal{A}$–parameter and compute its Bayes risk.

  (2.3) Define the *relative efficiency* to be the ratio of the risk obtained in (2.1) with that obtained in (2.2).

We determined an optimal procedure for the problem of estimating the product using the $\mathcal{D}$–parameters set equal to $[(1,1); (1,1)]$. Then, using path induction, we computed the relative efficiency, (2.3), for 25



Relative efficiency of solution to nonlinear estimation problem
with sample size $n = 100$ when analysis prior differs from design prior.

Figure 7: An Example of a Robustness Analysis

---

different versions of the $\mathcal{A}$–parameter configurations: $\mathcal{A}_1 = 0.01$, 0.1, 1, 10, 100 $\quad \mathcal{A}_2 = 0.01$, 0.1, 1, 10, 100 $\quad \mathcal{A}_3 = 1$ $\quad \mathcal{A}_4 = 1$. Figure 7 is an interpolated surface plot based on the relative efficiencies for the 25 grid points. The data point .865 in Figure 7, for example, which corresponds to $\mathcal{A}_1 = 0.01$, $\mathcal{A}_2 = 0.01$, $\mathcal{A}_3 = 1$, $\mathcal{A}_4 = 1$ represents the relative efficiency of the procedure generated assuming the uniform distribution but evaluated using the specified $\mathcal{A}$–parameters versus the optimal procedure both generated and evaluated using the $\mathcal{A}$–parameters.

The standard approach (namely backward induction) used for undertaking the repeated evaluations needed for step (2.1) requires 25 iterations of an algorithm taking $\Theta(N^4)$ time. Using the path induction algorithm requires only a single run of an initialization phase, taking $\Theta(N^4)$ time, and 25 iterations of an evaluation phase taking only $\Theta(N^3)$ time. This difference can be a significant help as the sample sizes grow.

5 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 from each arm for the next stage. Within a stage, however, 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 [10], and analysis is addressed here. To simplify analyses, the total sample size n will be fixed, and it is assumed that there are k stages, with $k \ll n$. 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 are approached slightly differently.

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 path 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 path 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 needed for evaluation via backward induction. While the evaluation time of path 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 path 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 be completed in $\Theta(v \cdot n^4)$ time, using only $\Theta(n^2)$ storage, while a very simple implementation of path 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 path 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^{th} stage starts at state σ, 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.

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. It appears that this is the technique most commonly used in practice, as 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 path induction initiation. A single evaluation of an improved backward induction algorithm has the same time and space requirements as a path 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.

```
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 σ ∈ S^{j-1} by i = n_2^j(σ) into lists S^{j-1}(i)
    for all states σ of level m^{j-1}, P'(σ) = 0  {P' used for path counts in midstage}
    for all states σ ∈ S^{j-1}(m^j − m^{j-1}), P'(σ) = P(σ)
    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) ∈ 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) + (t choose s) P(s_1, f_1, s_2)
    for all states σ ∈ S^j, P(σ) = P'(σ)
```

Figure 8: Deterministic Staged Allocation, Fixed Stage Sizes

5.1 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^{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^{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 σ_s represents starting at σ and obtaining all 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 8, 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 5.1 *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 path 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 path 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^{th} stage the i-loop of Figure 8 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 added only 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 path 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 path 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. □

5.2 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:

```
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 σ ∈ S^{j-1} by i = n_2^j(σ) into lists S^{j-1}(i)
    for all states σ of all levels, P'(σ) = 0  {P' used for path counts in midstage}
    for all states σ ∈ S_n^{j-1}, P'(σ) = P(σ)
    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 σ = (s_1, f_1, s_2, f_2) ∈ S^{j-1}(i)  {add arm 1 observations from σ}
            t = n_1^j(σ)
            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) + (t choose s) P(s_1, f_1, s_2, f_2)
    for all states σ ∈ S^j, P(σ) = P'(σ)
```

Figure 9: Deterministic Staged Allocation, Arbitrary Stage Sizes, $k \geq 4$

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 the first difficulty, 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 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 the second difficulty, 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 9 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 σ in S^{j-1}, the arm 1 observations from σ 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 9 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 5.1 apply. The case of $k = 3$ is intermediate between the fixed stage size and arbitrary stage size situations since each stage has either a fixed beginning or a fixed ending. This can be exploited, as is shown in the following theorem.

Theorem 5.2 *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 path 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 path 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 σ 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 toward level n. However, when arm 1 observations from state σ 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 9 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.

6 Extensions

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

6.1 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" [14, 17] (or, more accurately, "play the winner/switch on a 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.

6.2 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, lthough 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 path 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 from each arm at each step) 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 $\Theta(r \cdot n)$, which permits dramatically faster evaluation by path induction.

6.3 Expanding Sample Size

One unusual feature of path induction, as compared to backward induction or dynamic programming, is that it can occasionally be used 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 ad hoc sequential allocation rules, such as alternating allocation, play the winner (Section 6.1), randomized play the winner (Section 6.4), vector-at-a-time (Section 6.2), Thompson's rule [15], 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 sample size-independent rule at sample sizes $n_1 < n_2 < \cdots < n_t$ 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 seems capable of expanding from small sample sizes to larger ones, and thus the relative advantage of path induction is magnified in such settings.

6.4 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 [16], 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 σ, 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 one can encounter situations where τ is an immediate successor of σ, and there is a state σ' which is an immediate successor of σ and an immediate predecessor of τ. 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 τ in a single stage, without passing through σ'.

6.5 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, 12], 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 path 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 σ, $P_w(\sigma)/P(\sigma)$ is the average number of switches occurring on a path from 0 to σ. To determine P_w, suppose one is at state σ_i and is updating information for a successor τ_j, where the subscript indicates the arm most recently tried. Then

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

Higher order moments can be computed similarly, adding one new array per degree.

7 Final Remarks

Path induction is an aid to evaluating sequential procedures and complements other approaches such as exact or asymptotic analyses, backward induction, and dynamic programming.

Path induction 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 corresponding reduction in the time or space required.

Depending on the procedure and sample size, path induction can be orders of magnitude faster than previous exact computational approaches for performing multiple evaluations of a procedure. For example, for several years now we have been able to use path induction on workstations [9] 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 of the restrictiveness of the computational space and time constraints, Jones [11], in 1992, computed results for the 2-AB problem that we described in Section 1. In this paper [11], the author acknowledges that his calculations could only be carried out for a sample of size $n = 25$ and he explicitly noted that this was due to computational constraints. Working contemporaneously, the we obtained the results in [9] for $n = 150$. These results were for a sequential model nearly identical to the one used by Jones, although the calculations for [9] included criteria that required approximately 100 evaluations per procedure, while the former required only a couple of evaluations per procedure. Thus, in terms of the algorithms used in [11], the work in [9] solved problems approximately $(150/25)^4 \cdot 100 \approx 100,000$ times harder.[1] While an advantage of a factor of 10 or so 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 path induction instead of backward induction.

Finally, although the emphasis here has been to show the utility of path induction over backward induction whenever multiple evaluations are required, we also show how to improve the implementation of backward induction for analyzing staged allocation procedures. The algorithms for determining P values appearing in Section 5 can be reversed to yield algorithms for backward induction. These algorithms are faster than standard implementations by a factor of n or more, and apparently are superior to any previously appearing in the literature. However, it will still be true that one should use path induction, instead of the improved backward induction, if multiple evaluations are required.

[1]Note that the algorithms used in [11] were essentially the same as those used by all other researchers. Our only reasons for singling out this paper are that the results can be directly compared and that the paper explicitly noted that the sample size was limited by computational constraints.

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] Gittins, J.C. (1979) "Bandit processes and dynamic allocation indices", *J. Roy. Statist. Soc. Ser. B* **41**, pp. 148–177.

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

[10] Hardwick, J. and Stout, Q.F. (1995), "Determining optimal few-stage allocation rules", *Computing Science and Statistics* **27** (1995), pp. 342–346.

[11] 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.

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

[13] Mehta, C. and Patel, N. (1983) "A network algorithm for performing Fisher's exact test in $r \times c$ contingency tables", *J. Amer. Statist. Assoc.* **78**, pp. 427–434.

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

[15] 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.

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

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