In *Adaptive Designs:* IMS Lecture Notes–Monograph Series **25**, 1995, pp. 223-237.

Exact Computational Analyses for Adaptive Designs

Janis P. Hardwick[1] Quentin F. Stout[2]
Statistics Department EECS Department
University of Michigan, Ann Arbor, MI 48109

Abstract We show how to compute optimal designs and exact analyses of allocation rules for various sequential allocation problems. The problems we have solved include parameter estimation in an industrial scenario, and testing in a clinical trial. Our computational approach incorporates backward induction, dynamic programming, and a new technique of forward induction. By utilizing efficient algorithms and careful implementation, we are able to determine exact solutions to practical problems previously approached only through simulation or approximation.

Keywords: constrained dynamic programming, forward induction, backward induction, bayesian design, multiple criteria, clinical trials

1 Introduction

We discuss the exact analyses of sequential allocation rules for a variety of applications. Our interests include both estimation and testing problems, and we have examined allocation rules for selected applications in medicine and industry. In the former, we study the design and evaluation of optimal allocation rules for clinical trials with ethical costs [7, 8, 10]. In the latter, we consider the problem of estimating the failure probability of a product composed of two subparts [9]. Generally speaking, our goal is to show that, rather than relying on asymptotic approximations or simulations as complete solutions to such problems, one can analyze them exactly. Asymptotic approximations are important, and nicely compliment our work, but they often suffer from having error bounds that are excessive or unknown for sample sizes of practical interest. Simulations are a useful approach whenever exact calculations are infeasible, but there does not seem to be widespread awareness of the extent to

[1]Research supported in part by National Science Foundation under grants DMS-8914328 and DMS-9157715.

[2]Research supported in part by National Science Foundation/DARPA under grant CCR-9004727.

which sequential problems now admit of exact calculations. The increasing feasibility of exact computational analyses comes about from a combination of perpetually increasing computing power and improved algorithms.

To make descriptions more concrete, we utilize the terminology of clinical trials, observing *patients* on *treatments* T_1 and T_2. Responses to treatments are dichotomous, either *success* or *failure*, where, for each patient, the probability of success on treatment T_i is p_i, $i = 1, 2$, with all patient responses being independent. For most of our work we assume that there is prior information available concerning p_1 and p_2. This information is modeled in the form of a joint distribution function ξ on (p_1, p_2), taken to be the product of two independent beta random variables. We assume that response to treatment is rapid, so that the outcome of each patient is known before the next patient arrives.

We need to use a bit of computer science notation in our analyses. Given positive functions f and g defined on the natural numbers, we say $f(n) = \Theta(g(n))$ if there are positive constants c, d, N such that $c \cdot g(n) \leq f(n) \leq d \cdot g(n)$ for all $n \geq N$. We say $f(n) = \omega(g(n))$ if $\lim_{n \to \infty} f(n)/g(n) = \infty$.

1.1 States

Let n denote the horizon of the study, i.e., the largest possible sample size, and let m denote the number of patients observed so far. A crucial step in making the calculations practical is the use of *states*, where a state $\alpha = (s_1, f_1, s_2, f_2)$ is a collection of sufficient statistics

$$
\begin{aligned}
s_1 &= \#\text{ patients succeeding on treatment } T_1, \\
f_1 &= \#\text{ patients failing on treatment } T_1, \\
s_2 &= \#\text{ patients succeeding on treatment } T_2, \text{ and} \\
f_2 &= \#\text{ patients failing on treatment } T_2.
\end{aligned}
$$

Note that $m = s_1 + f_1 + s_2 + f_2$.

The use of states greatly reduces the number of essentially different allocation rules, where an *allocation rule* is an algorithm which decides, for each patient, which treatment the patient is assigned to. At any patient m, the allocation rule may depend only on the prior distribution and the full history of treatment assignments and patient outcomes observed up to this point. Thus, for a given prior distribution and fixed sample size n, there are $2^{2^n - 1}$ different deterministic allocation rules. However, since the sequence of observations gives no more information than is contained in the sufficient statistics, it suffices to restrict attention to allocation rules that depend only on the prior and the sufficient statistics. There are only $\Theta(n^4)$ states, which reduces the number of distinct deterministic allocation rules to $2^{\Theta(n^4)}$. This is still large but is nonetheless a significant improvement.

1.2 Criteria

Our research is typically comprised of two parts: the *design* of allocation rules, optimized with respect to a specific criterion; and the *evaluation* of these and other, possibly ad hoc, rules according to multiple criteria. For clinical trials, criteria of interest include

- number of failures
- number of expected successes lost [1]
- number of patients assigned to the inferior treatment
- sample size
- cost
- probability of correct selection, P(CS)
- robustness.

These criteria have multiple variants. For example, 'probability of correct selection' may refer to the evaluation of the function $\mathrm{P}(\mathrm{CS}|p_1, p_2)$ for all treatment probabilities (p_1, p_2) in a specified region of the parameter space. On the other hand, it may refer to the same concept, but evaluated from a Bayesian viewpoint in which $\mathrm{P}(\mathrm{CS}|p_1, p_2)$ is integrated with respect to a prior distribution on (p_1, p_2). A third possibility, often encountered in the ranking and selection literature [2], is to consider a minimax-like definition where there is an indifference region $|p_2 - p_1| < \Delta$, and one is concerned with the smallest value of $\mathrm{P}(\mathrm{CS}|p_1, p_2)$ outside this region. That is, the measure of interest is

$$\min_{|p_2 - p_1| \geq \Delta} \mathrm{P}(\mathrm{CS}|p_1, p_2) \ .$$

It is to this version of the P(CS) criterion that we refer later on in this paper.

For other applications there may be different sets of criteria of interest or variations of the same criteria. For example, in the fault tolerance problem studied in [9, 14, 16], the goal was to minimize the mean squared error of the estimate of the fault tolerance. In industrial applications some of the "ethical" criteria, such as number of uses of the inferior choice, may be far less important, while time or cost criteria may become more important and/or complex. As an example, the cost may include a component that represents the setup costs of switching from one alternative to another. While we emphasize clinical terminology and criteria in this paper, many industrial criteria can be accommodated by the same techniques.

2 Backward Induction and Dynamic Programming

The phrases "backward induction" and "dynamic programming" are often used in a somewhat confusing, overlapping manner. Here we use *dynamic programming* only for the process of optimization. We use *backward induction* for the process of evaluation. This seems to be the accepted usage for dynamic programming, but "backward induction" often seems to be used in both contexts.

To illustrate these terms, consider the following setup: suppose we are interested in some criterion $\mathcal{C}$, such as number of failures incurred during the trial, and suppose γ is an allocation rule that we wish to evaluate with respect to $\mathcal{C}$. We use $C_\gamma(\alpha)$ to denote the expected value of $\mathcal{C}$, conditional on the experiment reaching α given the prior ξ and using rule γ. We assume that we can evaluate $C_\gamma(\alpha)$ whenever α is a terminal state, and our goal is to evaluate $C_\gamma(0,0,0,0)$.

In general, for a nonterminal state $\alpha = (s_1, f_1, s_2, f_2)$, if γ were to assign the next patient to T_1 then we would have the recursive equation

$$\begin{aligned} C_\gamma(s_1, f_1, s_2, f_2) &= \mathbf{E}^m(p_1) \cdot C_\gamma(s_1+1, f_1, s_2, f_2) + \\ &\quad (1-\mathbf{E}^m(p_1)) \cdot C_\gamma(s_1, f_1+1, s_2, f_2) \end{aligned} \tag{1}$$

where $\mathbf{E}^m(p_1)$ is the posterior expected value of p_1, given prior ξ and data α. Similarly, if γ were to assign the next patient to T_2 then we would have the recursive equation

$$\begin{aligned} C_\gamma(s_1, f_1, s_2, f_2) &= \mathbf{E}^m(p_2) \cdot C_\gamma(s_1, f_1, s_2+1, f_2) + \\ &\quad (1-\mathbf{E}^m(p_2)) \cdot C_\gamma(s_1, f_1, s_2, f_2+1)\ . \end{aligned} \tag{2}$$

Equations 1 and 2 are the fundamental equations of *backward induction*, showing that one can evaluate $C_\gamma(s_1, f_1, s_2, f_2)$ if one knows the values on the right-hand sides and which treatment γ would allocate. By starting with terminal states and working backwards, one is ultimately able to evaluate the initial state $C_\gamma(0,0,0,0)$. With obvious modifications, this also applies if γ randomly allocates patients to treatments using probabilities depending on α and/or ξ.

The primary difference between general backward induction and *dynamic programming* is that dynamic programming determines the allocation rule γ that optimizes $\mathcal{C}$. This means that, at α, γ should choose the treatment that optimizes the value obtained in equations 1 and 2. One implication of this difference is that, at any state α, dynamic programming does roughly twice as many computations as does backward induction, since dynamic programming evaluates both equations and backward induction evaluates only one.

Dynamic programming, and to a lesser extent backward induction, is well-known to statisticians, especially in the area of adaptive designs. For example, for quite some time it has been known that dynamic programming can be used to solve the *finite* *horizon uniform bandit problem*, which is the problem of maximizing the expected number of successes [4]. In practice, however, dynamic programming and backward induction are often viewed as being too computationally slow or memory intensive, so they are often dismissed as being infeasible for interesting problems. One of the ongoing goals of our work is to show that this is not always true, and that computational advances make advanced sequential methods more practical.

2.1 Mixed Evaluations

As dynamic programming is designing an allocation rule, it is also evaluating it with respect to the criterion being optimized. Backward induction, however, has no such restriction, and thus can be used to evaluate rules on arbitrary criteria. For example, in [9], dynamic programming was used to find optimal allocation rules for the product of means problem mentioned in the introduction. Given a prior distribution ξ on the reliability parameters of the individual parts, dynamic programming was used to create an optimal rule γ^{ξ}. To evaluate the robustness of the design with respect to departures from ξ, the procedure γ^{ξ} was also evaluated utilizing different priors ξ', using backward induction for the evaluation.

Another example of mixed evaluations occurs when dynamic programming, coupled with a prior distribution, is used in a Bayesian design to create an allocation rule with the minimal expected number of failures, and the rule is then evaluated with respect to a maximin criterion like P(CS), described in Section 1.2. Similarly, one can evaluate ad hoc rules, such as play-the-winner/switch-on-loser(PW/SL) [15] or alternating allocation, on any of the criteria listed in Section 1.2.

While the notion of using backward induction for arbitrary evaluation of allocation rules is not new, nor profound, it has rarely been used. Two of the few works with which we are familiar that apply this to sequential allocation are [5, 13].

2.2 Wavefront Ordering

Equations 1 and 2 show that $C_\gamma(s_1, f_1, s_2, f_2)$ can be evaluated once $C_\gamma(s_1+1, f_1, s_2, f_2)$, $C_\gamma(s_1, f_1+1, s_2, f_2)$, $C_\gamma(s_1, f_1, s_2+1, f_2)$, and $C_\gamma(s_1, f_1, s_2, f_2+1)$ have been evaluated. Such dependencies are quite common in computational problems, and are known as *wavefront dependencies*. Any evaluation ordering is possible as long as these wavefront dependencies are satisfied, but typically one evaluates all states with $m+1$ patients before evaluating any states with m patients.

If m is fixed, then one of the components of the state can be eliminated. If f_2 is chosen, then we can denote $C_\gamma(s_1, f_1, s_2, f_2)$ by $C_\gamma^m(s_1, f_1, s_2)$, and the wavefront dependencies reduce to the fact that $C_\gamma^m(s_1, f_1, s_2)$ must be evaluated after $C_\gamma^{m+1}(s_1+1, f_1, s_2)$, $C_\gamma^{m+1}(s_1, f_1+1, s_2)$, $C_\gamma^{m+1}(s_1, f_1, s_2+1)$, and $C_\gamma^{m+1}(s_1, f_1, s_2)$. We can reuse the array holding C_γ^{m+1} to hold C_γ^m, as long as we obey these dependencies. They are just wavefront dependencies on fewer variables, coupled with the fact that the new value at (s_1, f_1, s_2) depends on the previous value at the same location.

Since there are many legal evaluation orderings, one can optimize to satisfy other computational goals. Usually, the most important of these is the fact that computer memory is organized hierarchically, with registers, cache, main memory, and disks. As data is used it is moved from larger, slower memory to smaller, faster memory in this hierarchy, and, except when moving to registers, it is moved in blocks of contiguous memory addresses. Therefore it is best to utilize contiguous blocks, and this fact becomes increasingly important as processor speeds continue to develop faster than memory speeds.

Fortran and S store an array A[0:i,0:j,0:k] as A[0,0,0], A[1,0,0], ..., A[i,0,0], A[0,1,0], A[1,1,0], ..., A[i,1,0], ..., A[i,j,0], A[0,0,1], ..., A[i,j,k]. Thus, for Fortran or S, an optimal evaluation ordering is

```
do 100 m=n,0,-1
    do 100 s2=0,m
        do 100 f1=0,m-s2
            do 100 s1=0,m-s2-f1
                 C(s1,f1,s2) = ...
```

Several other languages, such as C and Pascal, store matrices so that later indices vary faster, so for them it is best to reverse the ordering of the inner 3 loops, making s2 the variable of the innermost loop. For large arrays, an improper ordering can increase the running time by an order of magnitude.

2.3 Time/Space Analysis

Since the number of states is $\Theta(n^4)$ and it takes only a constant amount of compute time per state in either backward induction or dynamic programming, the total time required is $\Theta(n^4)$. By reusing space as outlined in the previous section, the total memory requirements are $\Theta(n^3)$. This analysis assumes that the posterior distribution can be computed in constant time. While this is not necessarily true, in the independent case at least, the prior, and hence posterior, distribution is a product of distributions on the individual treatments, so that only $\Theta(n^2)$ posterior means need to be computed for each treatment. Thus, while the computation of posterior means may be nontrivial, they do not dominate the computation time.

The previous paragraph is a worst-case analysis in which it is assumed that all states need to be evaluated. However, in many situations this is not true. For example, in evaluating PW/SL, one finds that only $\Theta(n^3)$ states are reachable since they must satisfy $|f_1 - f_2| \le 1$. Because of this, the time can be reduced to $\Theta(n^3)$, and space can be reduced to $\Theta(n^2)$. The simple nature of PW/SL can be exploited even further to get analytic evaluations without requiring backward induction, but the point is that often, special knowledge of the behavior of an allocation rule can be exploited to reduce the time and space needed to evaluate the rule.

Several researchers, including Berry [5, 6], have noted that space can be reduced in the fixed horizon, nonstopping rule problem by utilizing the constraint that $s_1 + f_1 + s_2 + f_2 \leq n$. This implies that only about 1/6 of the entries in the array are accessed. While this is a common situation, standard computer languages do not provide a mechanism for allocating only the appropriate corner of the array. Hence to utilize this information a programmer must declare a 1-dimensional array of size approximately $n^3/6$, and then explicitly determine where each state is stored. While not difficult, this either decreases program readability and ease of modification, or increases compute time. Further, the increased use of virtual memory diminishes the importance of this space reduction, since only the utilized memory locations will be moved into main memory (or cache). On personal computers, however, this may still be useful since PC's typically lack virtual memory capabilities and have less memory.

One interesting computer science question is whether dynamic programming is an optimal algorithm for minimizing expected failures. That is, we know that dynamic programming produces optimal allocation rules for minimizing failures, but a computational complexity question arises, namely,

> Is dynamic programming the fastest way to produce an allocation rule minimizing expected failures?

Thus far, no faster algorithm has been determined, but there is no proof that one does not exist. Experimental observations, reported in [8], found that the allocation rule that minimizes failures actually reaches $\Theta(n^4)$ states. Still, even a proof that there are $\Theta(n^4)$ reachable states does not demonstrate that $\Theta(n^4)$ time is needed to determine the minimum possible expected number of failures. Unfortunately, despite the widespread use of dynamic programming in many different fields, there are few problems for which it is known that dynamic programming provides the optimal solution in the fastest possible time.

3 Constrained Dynamic Programming

A useful variation of dynamic programming is to constrain the choices available at each stage, in order to achieve some secondary goal in addition to the goal of optimizing a primary criterion. For example, one may be concerned about a drift in the treatment probabilities over time, desiring that no treatment be used more than k times in a row before using the other treatment. In this scenario, there are situations in which only one treatment is allowable, so the choices available for dynamic programming are constrained. The notion of a 'state' becomes more complicated since it must now include an indication of the most recent treatment and the number of consecutive times it has been used.

A different form of constraint was considered in [3, 10], where a class of optimal equal allocation rules were considered. For a fixed sample size, one can show that the P(CS) is maximized by allocating equally to the two treatments. Often "equal allocation" is implemented as "alternating allocation", but all ways of allocating equally have the same probability of correct selection, and there are $2^{\Theta(n^4)}$ different deterministic equal allocation rules. A natural question arises, namely,

> Are some equal allocation rules better than others with respect to a second criterion, and if so how do we determine an optimal rule?

In many cases, an equal allocation rule that is optimal for a second criterion is one that has, in some way, been terminated early. One important form of early termination is curtailment, where an allocation rule γ can be *curtailed* (stopped with no change in decision) at a state α if all terminal states that γ can reach from α will yield the same decision.

For example, suppose equal allocation is being used with a horizon of n (even), where the terminal decision rule is dichotomous, deciding in favor of T_1 if $s_1 > s_2$, in favor of T_2 if $s_2 > s_1$, and randomly declaring a winner if $s_1 = s_2$. Then one may stop and declare T_1 the winner at any state α where $s_1 > n/2 - f_2$, since any state reachable from α by equal allocation will have no more than $n/2 - f_2$ successes on T_2. Notice that, by definition, P(CS) is not changed by curtailment. One can be more opportunistic and also declare T_1 the winner whenever $s_1 = n/2 - f_2$ and $m < n$. Strictly speaking this is not standard curtailment since one reachable terminal state is a tie with T_2 being declared the winner with probability 0.5; however, for equal allocation rules, it can easily be shown that this more aggressive curtailment does not affect the P(CS) no matter what the treatment probabilities are. This has apparently been noted by several researchers, and follows from a more general result in [12].

One can use dynamic programming to determine the optimum n-horizon equal allocation rule for criteria such as earliest expected curtailment (smallest expected sample size), fewest expected losses, least cost, etc. For each state, one first determines if the state is terminal, either because $m = n$ or because curtailment is possible. If it is terminal then the criterion is evaluated, while otherwise equations 1 and 2 are evaluated for each legal treatment, and the optimum is chosen. A treatment is *legal* at a state if and only if it has not already been used $n/2$ times, since any more uses cannot result in an equal allocation.

Curtailment can be viewed as a *pruning* operation, reducing the number of states that will be reached (or, for some states, reducing the number of paths that reach

Only $p_1 \geq p_2$ shown, $n = 200$, both treatment priors are Be(1,1)

Figure 1: Expected failures, $\gamma_{EA\mu F} - \gamma_{\mu F}$

them). This can be coupled with a *grafting* operation, where, once a state is reached where the terminal decision is known, then the experimenter is free to choose treatments arbitrarily for the remainder of the study (presumably choosing the winner). The use of grafting allows one to compare allocation rules fairly by extending them to a common horizon. For example, Figure 1 shows the difference between expected failures for the pruned-and-grafted equal allocation of minimal failures (denoted $\gamma_{EA\mu F}$), and the expected failures of the allocation rule solely designed to minimize failures (the classical uniform horizon two-armed bandit, denoted $\gamma_{\mu F}$), both with a horizon of 200.

An interesting open problem is to examine the optimal tradeoff curve of failures versus P(CS) for a given horizon. It is easy to show that this curve is convex, and that the $\gamma_{\mu F}$ rule gives the optimal tradeoff at one end. (Technically, $\gamma_{\mu F}$ is not quite unique, since in a few states the evaluations of expected failures for both treatments are the same, even though the treatments are not equivalent. The different choices in such a situation will yield slightly different P(CS), but the total variation is minuscule and one can determine the solution with maximal P(CS).) We conjecture that $\gamma_{EA\mu F}$ is the point at the other end, but we have yet to complete a proof of this. Beyond this we know of no feasible procedure for producing other points along the curve, although certain ad hoc allocation rules, such as those described in [7], give extremely good tradeoffs, nearly equal to $\gamma_{EA\mu F}$ in probability of correct selection and nearly equal to $\gamma_{\mu F}$ in expected failures.

4 Forward Induction

Often it is useful to evaluate an allocation rule and criterion for multiple (p_1, p_2) pairs. For example, suppose we wish to determine the P(CS) (as defined in Section 1.2) for a given indifference region $|p_1 - p_2| < \Delta$. For an arbitrary allocation rule it is not known which pair of treatment probabilities (p_1, p_2) yields the smallest $\mathrm{P}(\mathrm{CS}|p_1, p_2)$, although it is known that the minimum occurs along the line $|p_1 - p_2| = \Delta$, and thus multiple evaluations are needed to locate the minimum. Multiple evaluations are also needed if we want to produce the data for plots such as those in Figure 1. In such cases, a significant time reduction can be obtained by a path-counting technique we call *forward induction.*

4.1 Forward Induction Equations

To explain forward induction, let γ be any allocation rule, and let $P_\gamma(\alpha)$ denote the number of different paths by which γ can reach state α. Then $P_\gamma(0,0,0,0) = 1$, and for any state α that is not reachable by γ, $P_\gamma(\alpha) = 0$. For each state (s_1, f_1, s_2, f_2),

$$\begin{aligned} P_\gamma(s_1, f_1, s_2, f_2) &= P'_\gamma(s_1 - 1, f_1, s_2, f_2) + P'_\gamma(s_1, f_1 - 1, s_2, f_2) \\ &\quad + P''_\gamma(s_1, f_1, s_2 - 1, f_2) + P''_\gamma(s_1, f_1, s_2, f_2 - 1) \ , \end{aligned} \tag{3}$$

where the $'$ indicates that the term is included only if γ, being in that state, would choose treatment T_1, and the $''$ indicates that the term is included only if γ, being in that state, would choose treatment T_2. For example, let γ be the PW/SL allocation rule starting with treatment T_1. Then we have $P_\gamma(1,0,0,0) = 1$, $P_\gamma(0,1,2,0) = P''_\gamma(0,1,1,0) = 1$, and $P_\gamma(2,1,0,1) = P'_\gamma(1,1,0,1) + P''_\gamma(2,1,0,0) = 3$.

To utilize the P_γ values, notice that, given γ and (p_1, p_2), each sequence of outcomes along any path arriving at state $\alpha = (s_1, f_1, s_2, f_2)$ is equally likely, occurring with probability $p(\alpha; p_1, p_2)$ which is given by

$$p(\alpha; p_1, p_2) = p_1^{s_1} \cdot (1 - p_1)^{f_1} \cdot p_2^{s_2} \cdot (1 - p_2)^{f_2} \ .$$

Thus, the probability that γ will reach α if the treatment probabilities are (p_1, p_2), is $P_\gamma(\alpha)\, p(\alpha; p_1, p_2)$. For a given criterion $\mathcal{C}$ and treatment probabilities (p_1, p_2), the expected value of $\mathcal{C}$ given the procedure γ is

$$\sum_{\alpha \text{ terminal}} C_\gamma(\alpha) P_\gamma(\alpha)\, p(\alpha; p_1, p_2) \ . \tag{4}$$

The same approach can be used to evaluate $\mathcal{C}$ given γ for a specific prior ξ - just replace $p(\alpha; p_1, p_2)$ with $\int p(\alpha; p_1, p_2)\, \mathrm{d}\xi$.

With simple modifications, one can also handle the case where γ is not a deterministic function of the state and prior. In this case, each term on the right-hand side of equation 3 is multiplied by the probability that γ, being in the indicated state, would choose the relevant treatment. One example of such an allocation rule is randomized play the winner (RPW), introduced in [17].

4.2 Time/Space Analysis

Equation 3 shows that P_γ obeys wavefront dependencies which are the reverse of the ones observed for backward induction. By reversing the order of evaluation in Section 2.2, one can evaluate P_γ for arbitrary γ in $\Theta(n^3)$ space and $\Theta(n^4)$ time. As is the case with dynamic programming and backward induction, certain allocation rules such as RPW actually reach $\Theta(n^4)$ states, but others such as PW/SL and alternating allocation reach only $\Theta(n^3)$ states, permitting a reduction in time and space.

The advantage of forward induction over backward induction occurs when an allocation rule is being evaluated multiple times. This may arise because a single criterion is being evaluated for multiple (p_1, p_2) pairs or for multiple priors, or because several different criteria are being evaluated. If the rule is being evaluated k times, then repeated use of backward induction takes $\Theta(kn^4)$ time. Forward induction takes only $\Theta(n^4 + kn^3)$ time, since P_γ is computed once and repeated evaluations are performed only at terminal states, as shown in equation 4. This explicitly assumes that the number of terminal states is $\Theta(n^3)$, which is true if a fixed sample size is used. If a stopping rule is used then the time may be even further reduced, though there exist bizarre stopping rules that increase the number of terminal states to $\Theta(n^4)$. For example, stopping whenever s_1 and s_2 are even will have such an effect. However, for problems such as those examined here, we know of no stopping rule of statistical interest that has $\omega(n^3)$ terminal states, so it is fair to say that forward induction represents a significant practical improvement over backward induction whenever multiple evaluations are needed.

Additional time savings can be achieved by forward induction when multiple horizons are being examined (a common problem when one is searching through parameter spaces to find an appropriate rule). If the allocation rule is independent of the horizon, then calculations of P_γ for small horizons can be continued for larger horizons, so one need not start over for each new horizon. This computational approach was used to help determine the parameters of the allocation rules presented in [7]. The advantages of this approach, and various implementation details of forward induction, are discussed in [11].

5 Extensions and Conclusions

This work, and that of others such as in [5, 13], shows that exact computational design and analysis is possible for some practical sequential allocation problems. This ability allows one to create better, more flexible designs, optimized and analyzed for specific situations. Since sequential designs often offer significant cost or ethical advantages over their nonadaptive counterparts, but are often perceived as being too complex and difficult to design, making them computationally tractable is an important step towards their more widespread adoption. This approach is also pursued in [7], where the results were obtained using the computational techniques described in this paper.

The work described here has concentrated on an important, but relatively simple, scenario in which response is dichotomous and immediate, and only two treatments are being evaluated. Computationally acceptable extensions may be developed to handle some variations, such as grouped sequential designs or situations with more than a binary terminal decision (such as adding a "No difference" outcome). Other variations increase the computational requirements more rapidly, and their exact solution requires significant advances in algorithm efficiency and/or computational power. Important examples of such variations are delayed responses, polychotomous responses, and the evaluation of more than two treatments. For example, if there are t treatments being evaluated, with r responses, and a delay of d between treatment and response, then dynamic programming, backward induction, or forward induction will take $\Theta(t^d n^{tr})$ time and $\Theta(t^d n^{tr-1})$ space.

To handle such computationally intensive variations, we are currently developing algorithms for parallel computers, on which hundreds or thousands of operations can be performed concurrently. Parallel programming is more complicated than serial programming, so usually it is utilized only when the problem is important and serial computers are inadequate. For adaptive design calculations, efficient parallel programs have to take the wavefront dependencies into account to maximize the number of concurrent operations while minimizing data movement among processors.

References

[1] Bather, J.A. (1985), "On the allocation of treatments in sequential medical trials" *Int.'l Stat. Review* **53**, pp. 1-13.

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

[3] Bechhofer, R.E. and Kulkarni, R.V. (1982), "Closed adaptive sequential procedures for selecting the best of $k \geq 2$ Bernoulli populations", in *Proc. 3rd Purdue*

Symp. Stat. Decision Theory and Related Topics I (ed. S.S. Gupta and J. Berger), Academic Press, pp. 61-108.

[4] Bellman, R. (1957), *Dynamic Programming*, Princeton University Press.

[5] Berry, D.A. and Eick, S.G. (1987), "Decision analysis of randomized clinical trials: comparison with adaptive procedures" (unpublished manuscript).

[6] Berry. D.A. and Fristedt, B. (1986), *Bandit Problems: Sequential Allocation of Experiments*, Chapman and Hall.

[7] Hardwick, J. (1994), "A modified bandit as an approach to ethical allocation in clinical trials", this volume.

[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. (1993a), "Optimal allocation for estimating the product of two means", *Computing Science and Statistics* **24**, pp. 592-596.

[10] Hardwick, J. and Stout, Q.F. (1993b), "Optimal adaptive equal allocation rules", *Computing Science and Statistics* **24**, pp. 597-601.

[11] Hardwick, J. and Stout, Q.F. (1993c), "Exact computational analyses for sequential allocation problems", to appear.

[12] Jennison, C. (1983), "Equal probability of correct selection for Bernoulli selection procedures", *Commun. Stat. Theor. Meth. A* **12**, pp. 2887-2896.

[13] Jones, P.W. (1992), "Multiobjective Bayesian bandits", *Bayesian Statistics* **4**, pp. 689-695.

[14] Page, C. (1993), "Adaptive allocation for estimation", to appear.

[15] Robbins, H. (1952), "Some aspects of sequential design of experiments", *Bull. Amer. Math. Soc.* **58**, pp. 527-536.

[16] Shapiro, C. Page (1985), "Allocation schemes for estimating the product of positive parameters", *J. Amer. Statist. Assoc.* **80**, pp. 449-454

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