Estimating the Expected Cost of Function Evaluation Strategies

Rebi Daldal, Zahed Shahmoradi, Tonguç Ünlüyurt[[1]](#footnote-1)

**Abstract** We propose a sampling based method to estimate the expected cost of a given strategy that evaluates a given Boolean function. In general, computing the exact expected cost of a strategy that evaluates a Boolean function obtained by some algorithm may take exponential time. Consequently, it may not be possible to assess the quality of the solutions obtained by different algorithms in an efficient manner. We demonstrate the effectiveness of the estimation method on random instances for algorithms developed for certain functions where the expected cost can be computed in polynomial time. We show that the absolute percentage errors are very small even for samples of moderate size. We propose that in order to compare strategies obtained by different algorithms, it is practically sufficient to compare the estimates when the exact computation of the expected cost is not possible.

**Keywords** function evaluation • sequential testing • cost estimation • Monte Carlo methods

Introduction

In this work, we consider the problem of estimating the expected cost of a given strategy that evaluates a given Boolean function. In general, a feasible strategy to evaluate a Boolean function can be described as a Binary Decision Tree (BDT) whose size can be exponential in the size of the description of the Boolean function. Consequently, computing the expected cost of the strategy may not be executed in polynomial time if the standard ways that are described below are used. In this work, we propose to estimate the expected cost of a strategy by sampling the input vectors as in Monte Carlo methods. We demonstrate that the estimation works well on special cases where the expected cost of a strategy can be computed efficiently.

The first component of the input of the problem is a Boolean function in variables described in some form, a cost vector whose ith component is the cost of learning the value of ith variable and a probability vector whose ith component is the probability that ith variable takes the value 1 with We assume that the variables take values independent of each other. The Boolean function can be described as a Disjunctive Normal Form, Conjunctive Normal Form or via an oracle that provides the value of the function at a certain binary vector in constant time. Certain Boolean functions can be described in a more concise way. As some typical examples, we can mention threshold functions, k-out-of-n functions or Read-Once functions. For more information for Boolean function representations one can refer to (Crama and Hammer, 2011).

In addition, as part of the input, we assume that we are given a strategy, that correctly evaluates this Boolean function. In order to evaluate the function at a certain binary vector, we need to learn the values of some of the variables. A strategy is a feasible solution for our problem if is an algorithm that describes which variable to learn next or outputs the correct value of the Boolean function, given the values of the variables that have been learnt so far. In general, can be represented by a Binary Decision Tree. Each internal node of this Binary Decision Tree corresponds to a variable that we will test and learn its value whereas a leaf node corresponds to declaring the value of the function as 0 or 1. For instance, let us consider the following Boolean function given by its Disjunctive Normal Form as, . In Figure 1, we provide a feasible strategy for evaluating this function. We will refer to the process of learning the value of as testing variable . In this strategy, is tested first. If is 1, then is tested next. On the other hand, if is 0, then is tested. Then depending on the values of the variables either the value of the function is determined or other variables are tested.

We can compute the expected value of the strategy provided in Figure 1, by summing up over all leaves, the product of the total cost from the root to that leaf and probability of ending up at that leaf. For instance, the total cost of the rightmost leaf is and the probability of reaching this leaf is . So the contribution of this leaf to the expected cost is (. The total expected cost can be computed by summing up the contributions of all the leaves. For this example, the total expected cost can be written as:

We denote by the set of leaves of the strategy tree. For each let us define to be the set of variables on the path from the root to leaf The cost of a leaf is defined as and the probability of a leaf l is defined as . Then the expected cost of a strategy can be defined as;

There are alternative ways to compute the expected cost of a strategy. A slightly different method than the one described above computes the probability that any variable is tested and sum up over all variables. Another method uses a recursive approach and it computes the expected cost of a given strategy by adding the cost of the variable at the root and the expected costs of the right and left subtrees of the root. The cost of a leaf node is 0. So one can recursively compute the expected cost. In fact, this approach could be used to find the optimal strategy by a dynamic programming formulation.

# 

**Figure 1-Example strategy.**

Although we have these formulations to compute the expected cost of a given strategy there is a practical problem about all these methods. All these methods use the Binary Decision Tree representation of the strategy and the size of the Binary Decision can be exponential in the input size. Consequently, the number of leaves of the Binary Decision Tree will be exponential input size. Here the input size can be the size of description of the Boolean function. So it may change from two integers in the case of k-out-of-n functions to an exponential size in the number of varibales in the case of a Disjunctive Normal Form. Consequently, all the methods mentioned above will be exponential time methods, in general to compute the expected cost of a given strategy described by a Binary Decision Tree.

On the other hand, a strategy can also be described as a black box (algorithm or function) that computes the next variable to test given the values of already tested variables or outputs the correct value of the function. This black box can run in polynomial time and one can execute this black box at most times to find out sequentially which variable to test next until the correct value of the function is found. Hence the black box is sufficient for practical execution of a strategy but it is not sufficient to compute the exact expected cost of the strategy. Obviously, one can execute the blackbox for all to construct the corresponding Binary Decision Tree and then use the methods explained above to compute the exact expected cost. As we have argued before, this requires exponential space and time in the worst case.

This problem has been considered in the literature motivated by applications in various areas including inspection in a manufacturing environment (Duffuaa and Raouf 1990) to project management (Reyck and Leus 2008) and to medical diagnosis (Greiner et al. 2006). A review can found in (Ünlüyurt 2004) describing various applications and results.

In this work, we propose to estimate the expected cost of a given strategy by using random samples and using the black box approach that we mentioned over the samples. So we will generate a number of samples using the probability vector Then for each binary vector in the sample, we will apply the given strategy (or the black box) to learn to the value of the function at that binary vector. We will estimate the expected cost by averaging the costs incurred over all samples. In order to evaluate the effectiveness of such an approach, we will use strategies that have been developed for certain Boolean functions whose exact expected cost can be computed efficiently in polynomial time. We will describe these systems in detail in the next section. Another way to evaluate the performance of our estimation procedure can be to measure practical convergence of the average value.

**Literature Review**

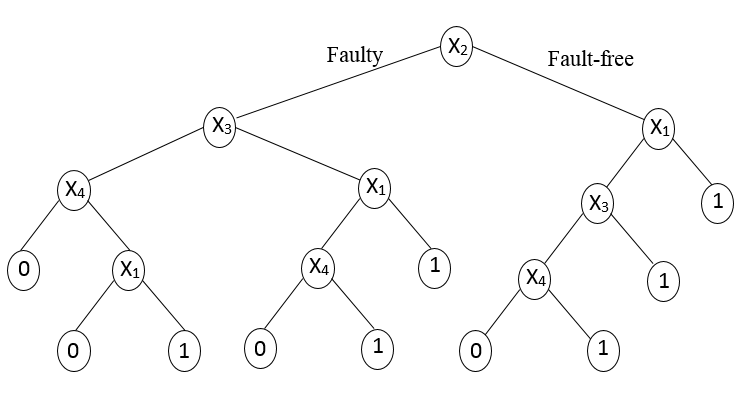
The problem of developing optimal strategies for evaluating Boolean Functions has long been studied in the literature in different contexts for various function classes. For some special classes of functions, algorithms that produce optimal or approximate strategies are known. We will not provide a full review of the results here but we will rather concentrate on the functions that are utilized in this work.

A series function (logical AND function) takes the value 1 if all of the variables are 1 and it takes the value 0 if at least one variable is 0. On the other hand, a parallel function (logical OR function) takes the value 0 if all the variables are 0 and it takes the value 1 if at least one variable is 1. For series and parallel functions, any feasible strategy is just a permutation of the variables. For instance, for a series function one would learn the variables one by one and stop when a variable is 0 or all the variables are learnt. The optimal permutation for a series system is the non-increasing order of ci/qi, where qi is the probability that a variable takes the value 0. The optimal solutions for series (parallel) systems are known for long time (see e.g. Mitten, 1960).

One generalization of series (parallel) functions is k-out-of-n functions. A k-out-of-n function takes the value 1 iff at least k of the n variables take the value 1. Consequently, for a k-out-of-n function to take the value 0, we need that at least n-k+1 variables are 0. Clearly, a series function is an n-out-of-n function and a parallel function is a 1-out-of-n function. The optimal evaluation strategy for k-out-of-n functions are known (Chang et al. 1990) and in addition it is possible to efficiently compute the expected cost of the optimal expected cost. Essentially, the algorithm sorts the variables in non-decreasing order of their and ratios. Let us assume and are the corresponding permutations. In other words,

Figure 2: Example strategy for a 2 out of 4 system

and

Then the next variable to learn is in where and Once the value of a variable is learned depending on the value of the variable, we either have a k-1 out of n-1 system or a k out of n-1 system. So we can apply the same procedure until we obtain the correct value of the function. As a matter of fact, this algorithm was proposed in (Ben-Dov 1981) but the proof in that article was incomplete. A special data structure referred to as “A Block Walking Diagram “allows us to compute the exact expected cost of this optimal algorithm in time. As an example, we consider 2-out-of-4 system where We show the strategy corresponding to this example in Figure 2.

Another generalization of a series (parallel) function is referred to as Series Parallel functions (SPS). (These are also known as Read-Once functions in Boolean functions literature) Without loss of generality, let us refer to a series function as a 1-level deep Series-Parallel function. Then we can define 2-level deep Series-Parallel function as a parallel connection of 1-level deep Series-Parallel functions. We can continue like this to define more complicated Series-Parallel functions. So a k-level deep Series-Parallel function is either a series or a parallel connection of k-1 level deep Series-Parallel functions.

The optimal solution for 2-level deep Series-Parallel functions is known (Boros and Ünlüyurt 2000, Greiner et al. 2006). Essentially, the optimal solution is a generalization of the optimal solution for a series (parallel) function. We replace each series function by a single variable whose cost is the optimal expected cost of evaluating that series function and whose probability of being 1 is the probability that the series function is 1. We now have a parallel function and we just implement the optimal algorithm for a parallel function. So in fact, we learn whether the series functions are 0 or 1 one by one according to the optimal ratio.

During this process, we continue with the same series function until it is evaluated and we never switch to other series functions without completely evaluating a series sub-system. One can generalize this algorithm to more complicated Series-Parallel functions. For more general functions, this generalization does not provide optimal results. In fact, it can produce very bad results (Boros and Ünlüyurt 2000, Ünlüyurt and Boros 2009). On the other hand, it is possible to compute the expected cost of this strategy in polynomial time by a recursive procedure. Essentially, one has to consider the function at the deepest level and replace it with a single variable according to its optimal solution. Then we end up with a 1 level less deep Series-Parallel function and if we continue applying the same procedure, at every stage the depth will decrease.

We consider the 2-level deep SPS in figure 3 and an example strategy for that SPS in figure 4.

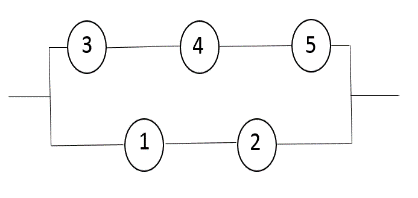


Figure 3: A 2-level deep SPS

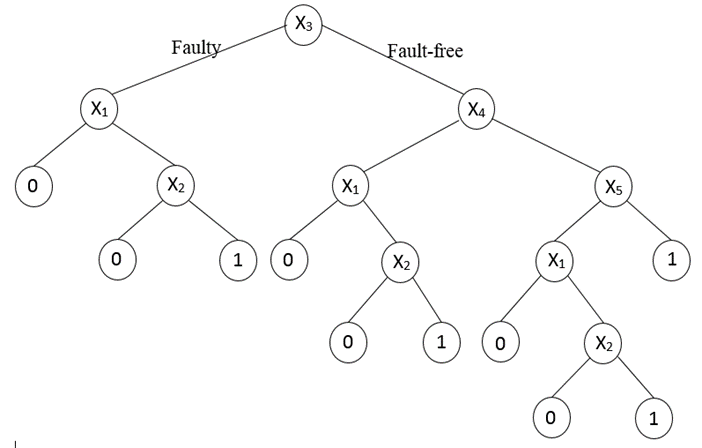


Figure 4: Example strategy for the SPS in Figure 3

**Proposed Methodology and Numerical Results**

We propose a simple sampling based method to estimate the expected cost of a given strategy. We sample binary vectors according to given the probability distributions and compute the average of the costs incurred for each binary vector. Since we can compute the expected cost of the optimal strategy for k-out-of-n functions and 2-level deep Series-Parallel functions efficiently, we utilize these systems to demonstrate the convergence of the estimation method. We also conduct experiments on a possibly non-optimal strategy for general Series Parallel Systems. All computations are performed using MATLAB.

## K-out-of-n systems

For k-out-of-n functions, we generate random instances with n=50, 100, 150 and 200. The k values are taken as n/8, n/4, n/2, 3n/4, 7n/8. Costs are uniformly generated from 0 to 10. There are three sets of probabilities: Uniform (0,1), Uniform (0.9,1) and Uniform (0,0.1) in order to represent a variety of instances. So in total we have 4\*5\*3\*5=300 problem instances. As mentioned before, we implemented the Block Walking Diagram proposed in (Chang et al. 1990) to compute the exact optimal cost in polynomial time. In Table 1, we present the average absolute errors of our estimation process for different number of samples from 200 to 1000. We also show the results in terms of *n* and the probability distribution since these factors seem to be interesting. In all cases, the estimation error is below 1%. We observe that the worst performance is when the probability distribution is uniform between 0 and 1. The absolute estimation errors decrease as the number of samples increases as expected. The decrease in the estimation errors stabilize as we approach 1000 samples. It is interesting and somewhat counterintuitive to observe that the estimation errors do not deteriorate when n increases. In fact, in our experiments the better estimates are for larger n.

## 2-Level deep Series Parallel Systems,

Similar to k-out-of n functions, we created random 2 level deep Series-Parallel functions. The number of subsystems is determined from 10 to 40. Each subsystem contains a random number of variables between 10 and 20. So on average we have 325 variables. Again we generated the costs uniformly between 1 and 10 and probabilities are determined in three different ways as described in k-out-of n functions. For each set of fixed parameters, we generate 5 independent instances. The average absolute estimations are presented in Table 2. For these systems, the worst estimations turn out to be when probabilities are drawn between 0.9 and 1. This indicates that the performance of estimation procedure is dependent on the type of the function that we are dealing with. We again observe no deterioration of the estimation error as the number of variables increase.

Table 1: Results for k-out-of-n systems

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **n** | Prob | **200** | **400** | **600** | **800** | **1000** |
| **50** |  | 0.93% | 0.79% | 0.52% | 0.51% | 0.46% |
|  | **(0,0.1)** | 0.78% | 0.39% | 0.40% | 0.28% | 0.17% |
|  | **(0,1)** | 1.40% | 1.57% | 0.87% | 0.91% | 0.96% |
|  | **(0.9,1)** | 0.59% | 0.42% | 0.30% | 0.34% | 0.24% |
| **100** |  | 0.70% | 0.55% | 0.36% | 0.36% | 0.31% |
|  | **(0,0.1)** | 0.41% | 0.29% | 0.25% | 0.14% | 0.14% |
|  | **(0,1)** | 1.29% | 1.10% | 0.62% | 0.68% | 0.59% |
|  | **(0.9,1)** | 0.39% | 0.26% | 0.22% | 0.25% | 0.18% |
| **150** |  | 0.50% | 0.35% | 0.35% | 0.27% | 0.31% |
|  | **(0,0.1)** | 0.33% | 0.20% | 0.18% | 0.15% | 0.17% |
|  | **(0,1)** | 0.90% | 0.55% | 0.72% | 0.50% | 0.59% |
|  | **(0.9,1)** | 0.26% | 0.29% | 0.16% | 0.17% | 0.17% |
| **200** |  | 0.48% | 0.35% | 0.33% | 0.26% | 0.25% |
|  | **(0,0.1)** | 0.30% | 0.24% | 0.16% | 0.18% | 0.16% |
|  | **(0,1)** | 0.93% | 0.63% | 0.66% | 0.46% | 0.46% |
|  | **(0.9,1)** | 0.22% | 0.17% | 0.17% | 0.14% | 0.13% |
|  | **Total** | 0.65% | 0.51% | 0.39% | 0.35% | 0.33% |

## General SPSs

As mentioned before optimal algorithms are not known for general SPSs. On the other hand, the algorithm that provides optimal solutions for 1-level deep SPSs and for 2-level deep SPSs can be generalized and used for general SPSs. This does not always provide an optimal strategy but an important property of such a strategy is that the exact expected cost can be computed by a non-trivial recursive algorithm. The non-triviality comes from the fact that while the values of the variables are learnt, the depth of the resulting SPS can decrease and/or some sub-systems may disappear. Consequently, the resulting strategy can be considered as more complicated than the case when the depth is only 2. We also tested our estimation procedure for random general SPSs whose depth vary from 2 to 10. It is also not a straightforward task to generate general random SPSs. We set the the maximum number of components at any level to 5 and 25% of the components constitute other subsystems. We vary the depth from 2 to10. We generate 5 independent instances for each combination of parameters and we generate all probabilities uniformly. So we have in total 45 instances. We report the absolute percentage gaps with respect to sample size and depth of the system in Table 3. We observe that the quality of estimations does not deteriorate too much as the depth of the systems increase. For these systems we kept the sample sizes larger to achieve good results. Still the whole estimation process takes almost no time.

Table 2 Results for 2-level deep SPSs

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **n** | **Prob** | **200** | **400** | **600** | **800** | **1000** |
| **10** |  | 1.94% | 1.50% | 1.24% | 0.91% | 0.69% |
|  | **(0,0.1)** | 0.69% | 0.61% | 1.04% | 0.42% | 0.59% |
|  | **(0,1)** | 2.16% | 2.26% | 0.69% | 0.71% | 0.66% |
|  | **(0.9,1)** | 2.95% | 1.65% | 1.99% | 1.61% | 0.82% |
| **20** |  | 1.26% | 1.08% | 1.02% | 0.54% | 0.78% |
|  | **(0,0.1)** | 0.55% | 0.31% | 0.33% | 0.28% | 0.14% |
|  | **(0,1)** | 1.17% | 1.60% | 0.92% | 0.77% | 0.71% |
|  | **(0.9,1)** | 2.06% | 1.33% | 1.80% | 0.57% | 1.48% |
| **30** |  | 1.13% | 1.16% | 0.85% | 0.36% | 0.56% |
|  | **(0,0.1)** | 0.88% | 0.17% | 0.38% | 0.25% | 0.16% |
|  | **(0,1)** | 1.52% | 1.71% | 0.94% | 0.34% | 0.64% |
|  | **(0.9,1)** | 0.98% | 1.59% | 1.22% | 0.50% | 0.89% |
| **40** |  | 1.16% | 1.27% | 1.02% | 0.63% | 0.74% |
|  | **(0,0.1)** | 0.22% | 0.14% | 0.36% | 0.26% | 0.16% |
|  | **(0,1)** | 0.77% | 1.13% | 0.60% | 0.35% | 0.68% |
|  | **(0.9,1)** | 2.50% | 2.54% | 2.11% | 1.28% | 1.39% |
| **Total** |  | 1.37% | 1.25% | 1.03% | 0.61% | 0.69% |

Table 3 Results for general SPSs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Depth/Sample size** | **2000** | **4000** | **6000** | **8000** | **10000** |
| 2 | 0.45% | 1.18% | 0.97% | 0.67% | 0.40% |
| 3 | 2.31% | 2.16% | 1.98% | 1.20% | 0.95% |
| 4 | 3.37% | 2.84% | 1.29% | 1.05% | 1.07% |
| 5 | 2.61% | 1.23% | 1.42% | 1.50% | 1.29% |
| 6 | 0.62% | 0.72% | 0.34% | 0.35% | 0.52% |
| 7 | 3.11% | 1.17% | 1.21% | 0.76% | 0.71% |
| 8 | 2.54% | 1.57% | 1.18% | 0.82% | 0.61% |
| 9 | 3.94% | 2.27% | 1.29% | 0.96% | 0.85% |
| 10 | 1.08% | 0.88% | 0.76% | 0.74% | 0.74% |
| **Total** | 2.23% | 1.56% | 1.16% | 0.89% | 0.79% |

## Conclusion:

In this work, we demonstrate the effectiveness of a sampling based estimation method for estimating the expected cost of a strategy that evaluates certain Boolean functions. Our results indicate that it is possible to estimate the expected cost of a strategy within a good accuracy even by 1000 to 10000 samples for certain classes of functions very efficiently in terms of time. So one can adapt this approach for more general functions when it is not possible to compute the expected cost efficiently. These estimations can be used to compare different algorithm. A similar analysis can be conducted for more complicated Series-Parallel functions since for some (not necessarily optimal) strategies for which the expected cost can be computed efficiently. As another continuation of this work, one may investigate the effectiveness of using a similar approach in order to compare different algorithms in a statistical manner.

**Acknowledgment:**

The authors gratefully acknowledge the support provided by TUBITAK 1001 programme, project number 113M478.

**References:**

Ben-Dov, Y. (1981), “Optimal testing procedures for special structures of coherent systems”, Management Science, 27(12):1410-1420.

Boros, E., T. Ünlüyurt, (2000), “Sequential testing of series parallel systems of small depth”, In: Computing Tools for Modeling, Optimization and Simulation, 39-74, Laguna and Velarde eds., Kluwer Academic Publishers, Boston.

Chang, M., W. Shi, W.K. Fuchs, (1990), “Optimal diagnosis procedures for *k*-out-of-*n* structures”, IEEE Transactions on Computers 39(4) , 559-564.

Crama, Y., P.L. Hammer, Boolean Functions: Theory, Algorithms and Applications, Cambridge University Press, 2011.

Duffuaa, S., A. Raouf, (1990). An optimal sequence in multicharacteristics inspection. Journal of Optimization, Theory and Applications, 67(1), 79–87.

Greiner, R., R. Hayward, M. Jankowska, M. Molloy, (2006). “Finding optimal satisficing strategies for and-or trees”, *Artificial Intelligence*, 170, 19-58.

Mitten L.G.,(1960). “An analytic solution to the least cost testing sequence problem”, Journal of Industrial. Engineering. ,January–February, 17.

Reyck, B. D.,, R. Leus, (2008). R&D-project scheduling when activities may fail. IIE Transactions, 40(4),367–384.

Ünlüyurt, T., E. Boros, (2009). “A note on “Optimal resource allocation for security in reliable systems””, European Journal of Operational Research, 199,2, 601-603.

Ünlüyurt, T. (2004). Sequential testing of complex systems: A review. Discrete Applied Mathematics, 142(1–3), 189–205.

1. R. Daldal • Z. Shahmoradi • T. Ünlüyurt (🖂) •

   Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey

   e-mail: [tonguc@sabanciuniv.edu](mailto:tonguc@sabanciuniv.edu)

   Rebi Daldal is currently with University of California, Berkeley.

   Zahed Shahmoradi is currently with University of Houston. [↑](#footnote-ref-1)