Asymmetric Information Measures: How to Extract Knowledge From an Expert So That the Expert's Effort is Minimal

Hung T. Nguyen[1], Elizabeth N. Kamoroff[2,3], and Vladik Kreinovich[3]

[1]Department of Mathematical Sciences, New Mexico State University
Las Cruces, NM 88003, USA, hunguyen@nmsu.edu
[2]Bell Northern Research (BNR), Durham, NC 27712, USA
[3]Department of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA, vladik@cs.utep.edu

Abstract

Knowledge acquisition is when we ask experts questions, and put the answers into the computer system. Since this is a very time-consuming task, it is desirable to minimize the effort of an expert.

As a crude estimate for this effort, we can take a number of binary (yes-no) questions that we ask. The procedure that minimizes this number is binary search.

This approach does not take into account that people often feel more comfortable answering "yes" than answering "no". So, to make our estimates more realistic, we will take into consideration that for a negative answer the effort is bigger.

This paper describes a procedure that minimizes the effort of an expert. We also estimate the effort of this "optimal" search procedure.

1 Introduction to the Problem

1.1 Informal Introduction

Knowledge acquisition is when we ask experts questions, and put the answers into the computer system. It is a very time-consuming and therefore expensive task. Thus, it is desirable to minimize the effort of an expert.

How do we estimate this effort? A reasonable way to do it is to estimate the effort by a number of questions. Of course, we can always ask just one question like "Please explain everything you know." A reasonable idea is to estimate the effort by the total number of *binary* questions (i.e., yes-no questions for which there are exactly two answers, "yes" and "no").

We will consider only the case when we know all possible *alternatives*, and we want to know which one of them is correct.

For example, suppose we know that we need to prescribe an analgesic to a patient, but we do not know which one to prescribe for this particular patient. If we have four alternatives, what is the right sequence of questions to ask in order to minimize the number of questions?

There exists a methodology called the *binary search* that helps to choose the minimal number of questions: if initially we had N mutually exclusive alternatives, then we ask the first "yes"-"no" question so that for half of these alternatives the answer is "yes", and for the other half the answer is "no". This way on each step we halve the set of alternatives, and in $\log_2(N)$ questions we narrow it down to one. It has been proved that by using this sequence of questions, we ask the smallest possible number of questions. This is explained in practically any book on algorithms and complexity; see, e.g. [2].

The main problem with this approach is that it does not take into consideration a well-known psychological fact that most people feel more comfortable answering "yes" than answering "no" (see, e.g., [1]). One of the reasons for this phenomenon is the following:

- The expert's time is valuable; because of this, the expert is usually asked to help only in most complex situations. For example, a medical expert would be normally called when an unusual situation happens. In this case, the expert expects to find competent people who, generally, know the answers to typical questions in his area of expertise, but who are puzzled by this particular unusual problem. In such situations, an expert is usually informed about the previous decisions and ideas of future decisions, and usually, he approves most of these decisions.
- If it so happens that half of the previous decisions were wrong, it usually means that the previous decision-makers were incompetent; in such situations, the expert feels that his valuable time was wasted because the appropriate solution is not to call a highly skilled expert, but rather to replace the existing decision makers with more competent people.

Similarly:

- When a knowledge engineer who interviews the expert asks him questions for which most answers are "yes", this shows that the knowledge engineer already has some preliminary knowledge of the area, and he is appropriately asking these questions to improve this knowledge.
- If, on the other hand, the knowledge engineer would start asking random questions, this would indicate that this engineer did not even bother to get some preliminary knowledge and therefore, the highly skilled expert is inappropriately used to answer questions some of which could be answered by simply consulting a textbook or a less skilled professional.

The larger the number of negative answers, the more discomfort the expert will experience, and the larger effort he will have to make to continue this interview.

In view of this phenomenon, instead of minimizing the total number of questions, it is more reasonable to minimize the effort of an expert, and in calculating this effort to assign more weight to "no" answers than to "yes" ones.

In this paper, we will formalize and solve this problem.

Comment. Our preliminary results first appeared in [3].

1.2 Towards Formalizing the Problem

In order to formalize the problem of selecting the *best* search procedure, we must first formalize the notion of a search procedure.

Initially, we have some finite set of alternatives, between which we will choose. We will denote this set by S (S for *s*et).

If this set contains more than one alternative, then we must ask an expert a question (and the question is supplied by this search procedure). The effect of this question is that the original set of alternatives is separated into two subsets $S = A(0) \cup A(1)$:

- the set of all alternatives for which the answer is "yes"; we will denote this set by $A(1)$ (1 stands for "yes", just like in the majority of computers);
- the set of all alternatives, for which the answer is "no"; we will denote this set by $A(0)$.

After asking this question, we thus know whether the (initially unknown) alternative $a \in S$ belongs to the set $A(0)$ or to the set $A(1)$. This is the only result of asking the question, so for our purposes, it does not matter how exactly this question was formulated: what matters is how the answer to this question divides the set of possible alternatives, i.e., what are the sets $A(0)$ and $A(1)$.

In principle, it is possible to ask a question in such an ambiguous way that for certain alternatives $a \in S$, both answers "yes" and "no" are possible, i.e., in mathematical terms, $A(0) \cap A(1) \neq \emptyset$. However, we are looking for an optimal (fastest) ways of eliciting knowledge. So, if we ask, instead, a new question in which

$A'(0) = A(0)$ and $A'(1) = S - A(0)$, then, since $A'(1) \subset A(1)$, this new questions narrows down an alternative even better. Therefore, since we are interested in finding the fastest elicitation method, it is sufficient to consider unambiguous questions for which $A(a) \cap A(1) = \emptyset$.

It is also possible, in principle, to have trivial questions to which the answer is always "yes" or always "no", i.e., for which either $A(1) = S$ and $A(0) = \emptyset$, or $A(0) = S$ and $A(1) = \emptyset$. Such trivial question does not add any information and can therefore be skipped. Therefore, since we are interested in the fastest knowledge elicitation, it makes sense to consider only pairs $\langle A(0), A(1)\rangle$ for which both sets $A(0)$ and $A(1)$ are non-empty.

If each of the sets $A(0)$ and $A(1)$ contains only one alternative, then there is no need to ask any further questions. If one (or both) of the resulting sets $A(0)$ and $A(1)$ contains more than one alternative, then we must continue asking questions. If the answer to the first question was "yes" (i.e., if we are in the set $A(1)$), then after the second question, the set $A(1)$ is divided into two subsets:

- The set of all alternatives that correspond to answer "yes" to both questions; we will denote this set by $A(11)$.
- The set of all alternatives for which the answers to the first two questions are correspondingly "yes" and "no"; we will denote this set by $A(10)$.

In general, for every sequence $\omega_1\omega_2\ldots\omega_k$ of 0's and 1's, $A(\omega_1\omega_2\ldots\omega_k)$ will denote the set of all alternatives which are possible after we received answers $\omega_1, \omega_2, \ldots, \omega_k$ to the first k questions. In particular, for an empty sequence Λ, we have $A(\Lambda) = S$.

Similarly to the above text, we can argue that since we select an optimal search procedure, it is sufficient to consider, for every ω, only subdivisions $A(\omega) = A(\omega 0) \cup A(\omega 1)$ for which $A(\omega 0) \cap A(\omega 1) = \emptyset$, $A(\omega 0) \neq \emptyset$, and $A(\omega 1) \neq \emptyset$.

For each search procedure P and for every alternative $a \in S$, there exists a sequence $\omega = \omega_1\ldots\omega_k$ for which $A(\omega) = \{a\}$; we will denote this sequence by $\omega(a, P)$. We will assign, to each "no" answer, a weight W_0, and to each "yes" answer, a weight $W_1 < W_0$. Then, for each alternative a, we can compute the total effort by adding the weights corresponding to all the answer from the sequence $\omega(a, P)$. For different alternatives, the effort may be different. As a numerical characteristic of the quality of a search procedure, we will take the worst-case effort, i.e., the largest of the efforts corresponding to different alternatives. Our goal is to find the search procedure for which this effort is the smallest possible.

Now, we are ready for the formal definitions.

1.3 Formal Definitions

Denotations.

- *By $\{0,1\}^*$ we mean the set of all finite sequences $\omega = \omega_1\omega_2\ldots\omega_k$ of 0's and 1's. The i-th element of the sequence ω will be denoted by ω_i. An empty sequence will be denoted by Λ.*
- *By $|S|$ we mean the number of elements in a set S.*

Definition 1. *Let S be a finite set. Elements of the set S will be called alternatives. By a search procedure P for the set S we mean a pair $P = \langle\Omega, A\rangle$, where:*

- *Ω is a finite subset of $\{0,1\}^*$,*
- *A is a function from Ω to the set $2^S - \{\emptyset\}$ of all nonempty subsets of S,*

and the following two properties hold:

- *$\Lambda \in \Omega$ and $A(\Lambda) = S$;*
- *For every $\omega \in \Omega$:*
  - *if $|A(\omega)| = 1$, then no extension of ω belongs to Ω;*
  - *otherwise (i.e., if $|A(\omega)| > 1$), both extensions $\omega 0$ and $\omega 1$ belong to Ω, and we have $A(\omega) = A(\omega 0) \cup A(\omega 1)$, $A(\omega 0) \cap A(\omega 1) = \emptyset$, $A(\omega 0) \neq \emptyset$, and $A(\omega 1) \neq \emptyset$.*

Comment. One can easily prove that for every search procedure P and for every alternative $a \in S$, there exists a sequence $\omega \in \Omega$ for which $A(\omega) = \{a\}$; this sequence will be denoted by $\omega(a, P)$. We will also say that this sequence ω *leads* to a.

Definition 2. *Let $W_0 > W_1$ be two positive real numbers.*

- *We will call W_0 the weight of a "no" answer and W_1 the weight of a "yes" answer.*
- *By the weight of a binary sequence $\omega = \omega_1\omega_2\ldots\omega_k$ we mean the sum $W(\omega) = \sum_{i=1}^{k} W_{\omega_i}$.*
- *For every search procedure P and for every alternative $a \in S$, the weight $W(\omega(a, P))$ of the sequence which leads to a will be called the weight of the alternative a in the procedure P.*
- *By an effort of a procedure P we mean $E(P) = \max_{a \in S} W(\omega(a, P))$.*
- *Let $N = |S|$, W_0, and W_1 be given. We say that P_0 is an optimal search procedure if*

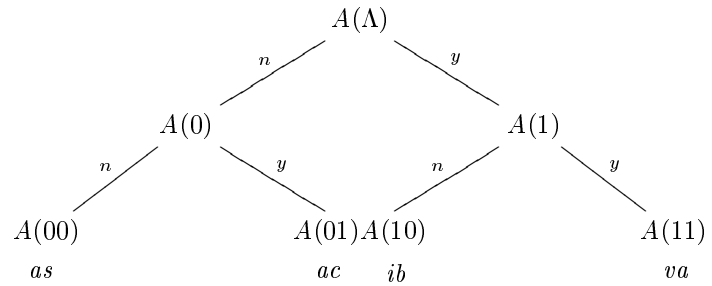
$$E(P_0) = \min_{P:|P|=N} E(P),$$

where the minimum is taken over all search procedures over S. This minimum will be denoted by $T(N)$.

1.4 Example 1: binary search (optimal for $W_0 = W_1$)

To illustrate the above definitions, let us consider a simple example in which a doctor has to choose between $N = 4$ possible analgetics: aspirin (as), acetaminophen (ac), ibuprofen (ib), and valium (va).

For this example, the standard binary search will lead, e.g., to the following search procedure P_1:



Before we ask any questions ($k = 0$), $S = \{as, ac, ib, va\}$. After we ask the first question ($k = 1$), the $N = 4$ alternatives are partitioned into two sets:

- the set $A(0) = \{as, ac\}$ corresponding to the "no" answer, and
- the set $A(1) = \{ib, va\}$ corresponding to the "yes" answer.

Here, $A(1) \cup A(0) = S$, $A(1) \cap A(0) = \emptyset$, and $|A(1)| = |A(0)| = 2$.

After the second question is asked ($k = 2$):

- the set $A(0)$ is partitioned into the two sets $A(00)$ and $A(01)$ so that $A(00) \cup A(01) = A(0)$ and $A(00) \cap A(01) = \emptyset$, and
- the set $A(1)$ is partitioned into the two sets $A(10)$ and $A(11)$ such that $A(10) \cup A(11) = A(1)$ and $A(10) \cap A(11) = \emptyset$.

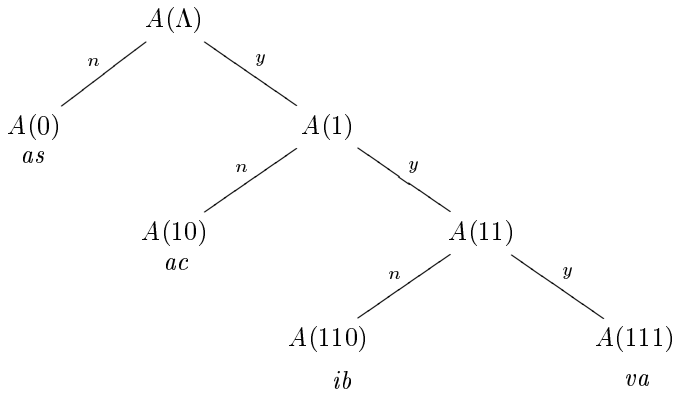
After two questions have been asked, we have $A(00) = \{as\}$, $A(01) = \{ac\}$, $A(10) = \{ib\}$, and $A(11) = \{va\}$. No further questions are needed.

Binary search is the best algorithm for the case of equal weights, when, e.g., $W_0 = W_1 = 1$. In this case, $W(\omega(a, P_1)) = 2$ for all four alternatives a, so $E(P_1) = 2$. For these weights, P_1 is the optimal procedure.

1.5 Example 2: a search procedure which is better than binary search ($W_0 > W_1$)

If $W_0 > W_1$, the binary search procedure P_1 may be not optimal. Indeed, e.g., when $W_1 = 1$ and $W_0 = 3$, we have $W(\omega(as, P_1)) = 6$, $W(\omega(ac, P_1)) = 4$, $W(\omega(ib, P_1)) = 4$, and $W(\omega(va, P_1)) = 2$; hence, the effort $E(P_1)$ (which is defined as the largest of the weights of all alternatives) is equal to $E(P_1) = 6$.

We can decrease the effort by applying a different search procedure P_2:



For this search procedure, $W(\omega(as, P_2)) = 3$, $W(\omega(ac, P_2)) = 4$, $W(\omega(ib, P_2)) = 5$, $W(\omega(va, P_2)) = 3$, and therefore $E(P_2) = 5$.

1.6 What we are planning to do

In this paper, we describe two results:

- first, we describe the algorithm which finds, for every N, W_1, and $W_0 > W_1$, the optimal search procedure;
- second, we will describe an even faster algorithm which computes the *asymptotically* optimal search procedure.

2 Description of the optimal search procedure

Let us first describe the result which helps to compute $T(N)$:

Proposition 1. $T(N) = \min\limits_{0<N_+<N} \{\max\{W_1 + T(N_+), W_0 + T(N - N_+)\}\}$.

Comment. For reader's convenience, all the proofs are placed in the last section of this paper.

This proposition leads to the following dynamic programming-type algorithm for computing $T(N)$ for a given N: In order to compute $T(N)$, we compute $T(1), T(2), \ldots, T(N)$ as follows:

- when $N = 1$, we take $T(1) := 0$ (if there is only one alternative, no questions need to be asked).
- if we already know $T(1)$ through $T(n-1)$, then we can use the formula from Proposition 1 to compute $T(n)$.

To compute $T(N)$, we need N iterations, on each of which we perform $\le c \cdot N$ arithmetic operations; thus, we can compute $T(N)$ in time $O(N^2)$.

Since we are interested not only in finding $T(N)$, we can, at each step, not only compute $T(n)$, but record the value $N_+(n)$ for which the expression $\max\{W_1 + T(N_+), W_0 + T(n - N_+)\}$ attains its minimum. Then, in designing a procedure, we divide a set $A(\omega)$ of size n into sets $A(\omega 1)$ of sizes $N_+(n)$ and $A(\omega 0)$ of size $n - N_+(n)$. We arrive at the following algorithm for designing the optimal search procedure:

Algorithm A. *Suppose that we are given a set S with $|S| = N$ alternatives, and the weights $W_0 > W_1$. Then, to develop the search procedure, we first sequentially compute the values $T(1), N_+(1), \ldots, T(N), N_+(N)$. After that, we construct the search procedure as follows.*

The search procedure is defined as a pair consisting of the set Ω of binary sequence and a function $A : \Omega \to 2^S$. In our algorithm, we will start with an empty set Ω and then add sequences to this set. As we add a sequence ω to the set Ω, we will also define the corresponding value $A(\omega)$. To be more precise, at each stage of our algorithm, we will have two sets:

- *a set Ω, with a function $A : \Omega \to 2^S$, and*
- *a set Ω_0 of sequences ω which we have already added to Ω and for which we still need to analyze "children" $\omega 0$ and $\omega 1$.*

We start with $\Omega = \Omega_0 = \emptyset$. Then, we add the empty sequence Λ to the set Ω, take $A(\Lambda) = S$, and add Λ to the set Ω_0. At each step of the algorithm, if $\Omega_0 \neq \emptyset$, we do the following for each sequence $\omega \in \Omega_0$:

- *if $|A(\omega)| = 1$, then no further questions are necessary, so we simply delete the sequence ω from the set Ω_0;*
- *if $N(\omega) = |A(\omega)| > 1$, then we:*
  - *add both children $\omega 1$ and $\omega 0$ to the set Ω;*
  - *select the corresponding question so that for $N_+(N(\omega))$ of the alternatives the answer is "yes", and for $N - N_+(N(\omega))$ of the alternatives the answer is "no", i.e., so that $|A(\omega 1)| = N_+(N(\omega))$ and $|A(\omega 0)| = N - N_+(N(\omega))$;*
  - *delete ω from the set Ω_0 of all sequences for which we are need to analyze children, and add the children $\omega 0$ and $\omega 1$ to this set.*

The construction ends when $\Omega_0 = \emptyset$.

Proposition 2. *For every N, W_0, and $W_1 < W_0$, the Algorithm* **A** *designs the optimal search procedure.*

Example. Let us show how this algorithm works for the above example, in which $N = 4$, $W_0 = 3$, and $W_1 = 1$.

According to the above Algorithm **A**, we first compute the values $T(n)$ and $N_+(n)$:

- We take $T(1) = 0$.
- When $n = 2$, the only possible value for N_+ is $N_+ = 1$, so

$$T(2) = \min_{0<N_+<2}\{\max\{1 + T(N_+), 3 + T(2 - N_+)\}\} =$$

$$\max\{1 + T(1), 3 + T(1)\} = \max\{1, 3\} = 3.$$

  Here, $N_+(2) = 1$.

- To find $T(3)$, we must compare two different values $N_+ = 1$ and $N_+ = 2$:

$$T(3) = \min_{0<N_+<N}\{\max\{1 + T(N_+), 3 + T(3 - N_+)\}\} =$$

$$\min\{\max\{1 + T(1), 3 + T(2)\}, \max\{1 + T(2), 3 + T(1)\}\} =$$
$$\min\{\max\{1, 6\}, \max\{4, 3\}\} = \min\{6, 4\} = 4.$$

  Here, the minimum is attained when $N_+ = 2$, so $N_+(3) = 2$.

- To find $T(4)$, we must consider three possible values $N_+ = 1$, $N_+ = 2$, and $N_+ = 3$, so

$$T(4) = \min_{0<N_+<N}\{\max\{1 + T(N_+), 3 + T(4 - N_+)\}\} =$$

$$\min\{\max\{1 + T(1), 3 + T(3)\}, \max\{1 + T(2), 3 + T(2)\}, \max\{1 + T(3), 3 + T(1)\}\} =$$
$$\min\{\max\{1, 7\}, \max\{4, 6\}, \max\{5, 3\}\} = \min\{7, 6, 5\} = 5.$$

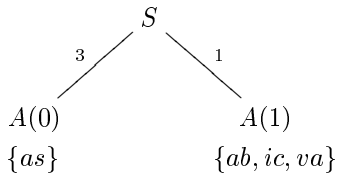
  Here, the minimum is attained when $N_+ = 3$, so $N_+(4) = 3$.

Now, we can start forming the optimal search procedure. We start with $\Omega = \Omega_0 = \emptyset$. Then, we add the empty sequence Λ to the set Ω, take $A(\Lambda) = S$, and add Λ to the set Ω_0.

The set Ω_0 consists of only one sequence $\omega = \Lambda$. For this sequence, $N(\Lambda) = |A(\Lambda)| = |S| = 4 > 1$, so we:

- add both its children 1 and 0 to the set Ω; now, $\Omega = \{\Lambda, 0, 1\}$;
- since $N_+(N(\Lambda)) = N_+(4) = 3$, we select the corresponding question so that $|A(1)| = 3$ and $|A(0)| = 4 - 3 = 1$; for example, we can take $A(1) = \{ab, ic, va\}$ and $A(0) = \{as\}$;
- we delete Λ from the set Ω_0, and add its children 0 and 1 to this set; now, $\Omega_0 = \{0, 1\}$.

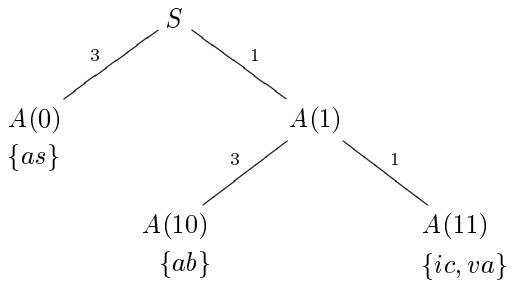
At this point, the subdivision looks as follows:



Now, in accordance with the Algorithm **A**, we have to consider both sequences 0 and 1 from the set Ω_0. For $\omega = 0$, we have $|A(0)| = 1$, so we simply delete this sequence 0 from the set Ω_0. For $\omega = 1$, $N(1) = |A(1)| = 3 > 1$, so we:

- add both its children 11 and 10 to the set Ω; now, $\Omega = \{\Lambda, 0, 1, 10, 11\}$;
- since $N_+(N(11)) = N_+(3) = 2$, we select the corresponding question so that $|A(11)| = 2$ and $|A(10)| = 3 - 2 = 1$; for example, we can take $A(11) = \{ic, va\}$ and $A(10) = \{ab\}$;
- we delete 1 from the set Ω_0, and add its children 10 and 11 to this set; now, $\Omega_0 = \{10, 11\}$.

At this point, the subdivision looks as follows:



In accordance with the Algorithm **A**, we have to consider both sequences 10 and 11 from the set Ω_0. For $\omega = 10$, we have $|A(10)| = 1$, so we simply delete this sequence 10 from the set Ω_0. For $\omega = 11$, $N(11) = |A(11)| = 2 > 1$, so we:

- add both its children 110 and 111 to the set Ω; now, $\Omega = \{\Lambda, 0, 1, 10, 11, 110, 111\}$;
- since $N_+(N(11)) = N_+(2) = 1$, we select the corresponding question so that $|A(111)| = 1$ and $|A(110)| = 2 - 1 = 1$; for example, we can take $A(111) = \{ic\}$ and $A(10) = \{va\}$;
- we delete 11 from the set Ω_0, and add its children 110 and 111 to this set; now, $\Omega_0 = \{110, 111\}$.

At this point, the subdivision looks exactly like the Procedure P_2.

In accordance with the Algorithm **A**, we have to consider both sequences 110 and 111 from the set Ω_0. For both sequences ω, we have $|A(\omega)| = 1$, so we simply delete both sequences from the set Ω_0. Now, $\Omega_0 = \emptyset$, so the algorithm stops.

The resulting procedure is thus optimal. For this search procedure, $E(P_2) = 5$.

3 Description of the asymptotically optimal search procedure

The above algorithm computes the optimal search procedure in $O(N^2)$ time. It is feasible, but for large N, N^2 computational steps may be still too many. So, to decrease the number of computation steps, we will describe an asymptotically optimal search algorithm which can be designed even faster. This algorithm is based on the following estimate:

Denotation. *By $f(n) \asymp g(n)$ we mean that there exists a constant $C > 0$ such that $|f(n) - g(n)| \le C$ for all n.*

Proposition 3. *For each W_0 and W_1, $T(N) \asymp K \cdot \log_2(N)$, where $K \ge 0$ is the solution of the following equation: $2^{-W_0/K} + 2^{-W_1/K} = 1$.*

When $W_0 = W_1$, this equation has a clear solution $K = W_0 = W_1$. In this case, the total effort is asymptotically equal to the effort of a single question ($W_0 = W_1$) times the number of questions (i.e., the amount of information necessary to determine the alternative).

In general, it is easy to prove that this equation has a solution, and that this solution is unique: its left-hand side is increasing in K. For $K = 0$, the left-hand side is equal to 0; for $K \to \infty$, it tends to 2. Since a continuous function attains all intermediate values, it has to be equal to 1 for some K; since the left-hand side is strictly increasing, it is equal to 1 for only one K.

In order to find this K, we can, e.g., use bisection (see, e.g., [2]) to find $\alpha = 2^{-W_1/K}$ from the equation $\alpha + \alpha^w = 1$, where $w = W_0/W_1$. Then, we can compute K as $K = -W_1/(\log_2(\alpha))$. Let us give two examples:

- When $W_0 = W_1 = 1$, we have $w = 1$, and the equation turns into $2\alpha = 1$. So, $\alpha = 0.5$, and $K = 1$.
- When $W_0 = 2W_1$, we have $w = 2$, and the equation turns into $\alpha + \alpha^2 = 1$. Its solution is the well-known *golden ratio* $\alpha = \frac{\sqrt{5}-1}{2} \approx 0.618$. So, in such a situation, the optimal portion of "yes" answers coincides with the golden ratio.

The algorithm itself is similar to the above Algorithm **A**, with the only difference that instead of the exact value $N_+(n)$, we use an approximate value $\lfloor \alpha \cdot n \rfloor$:

Algorithm B. *Suppose that we are given a set S with $|S| = N$ alternatives, and the weights $W_0 > W_1$. Then, to develop the search procedure, we first find α. After that, we construct the search procedure as follows.*

The search procedure is defined as a pair consisting of the set Ω of binary sequence and a function $A : \Omega \to 2^S$. In our algorithm, we will start with an empty set Ω and then add sequences to this set. As we add a sequence ω to the set Ω, we will also define the corresponding value $A(\omega)$. To be more precise, at each stage of our algorithm, we will have two sets:

- *a set Ω, with a function $A : \Omega \to 2^S$, and*
- *a set Ω_0 of sequences ω which we have already added to Ω and for which we still need to analyze "children" $\omega 0$ and $\omega 1$.*

We start with $\Omega = \Omega_0 = \emptyset$. Then, we add the empty sequence Λ to the set Ω, take $A(\Lambda) = S$, and add Λ to the set Ω_0. At each step of the algorithm, if $\Omega_0 \ne \emptyset$, we do the following for each sequence $\omega \in \Omega_0$:

- *if $|A(\omega)| = 1$, then no further questions are necessary, so we simply delete the sequence ω from the set Ω_0;*
- *if $N(\omega) = |A(\omega)| > 1$, then we:*
  - *add both children $\omega 1$ and $\omega 0$ to the set Ω;*
  - *select the corresponding question so that for $\lfloor \alpha \cdot N(\omega) \rfloor$ of the alternatives the answer is "yes", and for $N - \lfloor \alpha \cdot N(\omega) \rfloor$ of the alternatives the answer is "no", i.e., so that $|A(\omega 1)| = \lfloor \alpha \cdot N(\omega) \rfloor$ and $|A(\omega 0)| = N - \lfloor \alpha \cdot N(\omega) \rfloor$;*
  - *delete ω from the set Ω_0 of all sequences for which we are need to analyze children, and add the children $\omega 0$ and $\omega 1$ to this set.*

The construction ends when $\Omega_0 = \emptyset$.

Definition 3. *We say that an algorithm produces an asymptotically optimal search method if for every N, W_0, and W_1, the search procedures $P(N)$ produced by this algorithm satisfy the property $E(P(N)) \asymp T(N)$.*

Proposition 4. *For every N, W_0, and $W_1 < W_0$, the Algorithm* **B** *produces the asymptotically optimal search procedure.*

4 Statistical approach

4.1 Formulation of the problem

Instead of the *worst-case* effort $E(P)$, we can optimize the *average* effort $E^a(P)$:

Definition 4. *Let $W_0 > W_1$ be two positive real numbers, and S is a set with N elements.*

- *By an average effort of a procedure P over S, we mean*

$$E^a(P) = \frac{1}{N} \cdot \sum_{a \in S} W(\omega(a, P)).$$

- *We say that P_0 is a statistically optimal search procedure if*

$$E^a(P_0) = \min_{P:|P|=N} E^a(P),$$

where the minimum is taken over all search procedures over S. This minimum will be denoted by $T^a(N)$.

4.2 Statistically optimal search procedure

Proposition 5.

$$T^a(N) = \min_{0<N_+<N} \left\{ \frac{N_+}{N} \cdot (W_1 + T^a(N_+)) + \frac{N - N_+}{N} \cdot (W_0 + T^a(N - N_+)) \right\}.$$

This proposition leads to the following dynamic programming-type algorithm for computing $T^a(N)$ for a given N: In order to compute $T^a(N)$, we compute $T^a(1), T^a(2), \ldots, T^a(N)$ as follows:

- when $N = 1$, we take $T^a(1) := 0$ (if there is only one alternative, no questions need to be asked).
- if we already know $T^a(1)$ through $T^a(n-1)$, then we can use the formula from Proposition 5 to compute $T^a(n)$.

To compute $T^a(N)$, we need N iterations, on each of which we perform $\le c \cdot N$ arithmetic operations; thus, we can compute $T^a(N)$ in time $O(N^2)$.

To design a statistically optimal algorithm, we must, at each step, not only compute $T^a(n)$, but record the value $N_+^a(n)$ for which the expression from Proposition 5 attains its minimum. Then, in designing a procedure, we divide a set $A(\omega)$ of size n into sets $A(\omega 1)$ of sizes $N_+^a(n)$ and $A(\omega 0)$ of size $n - N_+^a(n)$. We arrive at the following algorithm for designing the optimal search procedure:

Algorithm C. *Suppose that we are given a set S with $|S| = N$ alternatives, and the weights $W_0 > W_1$. Then, to develop the search procedure, we first sequentially compute the values $T^a(1), N_+^a(1), \ldots, T^a(N), N_+^a(N)$. After that, we construct the search procedure as follows.*

The search procedure is defined as a pair consisting of the set Ω of binary sequence and a function $A : \Omega \to 2^S$. In our algorithm, we will start with an empty set Ω and then add sequences to this set. As we add a sequence ω to the set Ω, we will also define the corresponding value $A(\omega)$. To be more precise, at each stage of our algorithm, we will have two sets:

- *a set Ω, with a function $A : \Omega \to 2^S$, and*
- *a set Ω_0 of sequences ω which we have already added to Ω and for which we still need to analyze "children" $\omega 0$ and $\omega 1$.*

We start with $\Omega = \Omega_0 = \emptyset$. Then, we add the empty sequence Λ to the set Ω, take $A(\Lambda) = S$, and add Λ to the set Ω_0. At each step of the algorithm, if $\Omega_0 \neq \emptyset$, we do the following for each sequence $\omega \in \Omega_0$:

- *if $|A(\omega)| = 1$, then no further questions are necessary, so we simply delete the sequence ω from the set Ω_0;*
- *if $N(\omega) = |A(\omega)| > 1$, then we:*
  - *add both children $\omega 1$ and $\omega 0$ to the set Ω;*
  - *select the corresponding question so that for $N_+^a(N(\omega))$ of the alternatives the answer is "yes", and for $N - N_+^a(N(\omega))$ of the alternatives the answer is "no", i.e., so that $|A(\omega 1)| = N_+^a(N(\omega))$ and $|A(\omega 0)| = N - N_+^a(N(\omega))$;*
  - *delete ω from the set Ω_0 of all sequences for which we are need to analyze children, and add the children $\omega 0$ and $\omega 1$ to this set.*

The construction ends when $\Omega_0 = \emptyset$.

Proposition 6. *For every N, W_0, and $W_1 < W_0$, the Algorithm* **C** *designs the statistically optimal search procedure.*

4.3 Asymptotically statistically optimal search procedure

The above Algorithm **C** computes the optimal search procedure in $O(N^2)$ time. It is feasible, but for large N, similarly to the worst-case effort, N^2 computational steps may be still too many. So, to decrease the number of computation steps, we will describe an asymptotically optimal search algorithm which can be designed even faster. This algorithm is based on the following estimate:

Proposition 7. *For each W_0 and W_1, $T^a(N) \asymp K^a \cdot \log_2(N)$, where $K^a \geq 0$ and α^a are the solution of the following system of equations:*

$$\alpha^a \cdot W_1 + (1 - \alpha^a) \cdot W_0 + K^a \cdot (\alpha^a \cdot \log_2(\alpha^a) + (1 - \alpha^a) \cdot \log_2(1 - \alpha^a)) = 0;$$

$$W_0 - W_1 = K^a \cdot (\log_2(\alpha^a) - \log_2(1 - \alpha^a)).$$

Comments.

- Since α^a and $1 - \alpha^a$ are the probabilities of the "yes" and "no" answers, the first equation has a clear information meaning: it says that

  $$K^a = \frac{\alpha^a \cdot W_1 + (1 - \alpha^a) \cdot W_0}{-\alpha^a \cdot \log_2(\alpha^a) - (1 - \alpha^a) \cdot \log_2(1 - \alpha^a)},$$

  i.e., that K^a is equal to the average effort divided by the entropy of the corresponding probability distribution.
- Substituting this expression for K^a into the second equation, we get a single equation for a single unknown α^a.
- Alternatively, we can use the second equation to express α^a in terms of K^a: namely,

  $$\log_2\left(\frac{\alpha^a}{1 - \alpha^a}\right) = \frac{W_0 - W_1}{K^a},$$

hence

$$\frac{1-\alpha^a}{\alpha^a} = \frac{1}{\alpha^a} - 1 = 2^{-(W_0-W_1)/K^a},$$

and

$$\alpha^a = \frac{1}{1+2^{-(W_0-W_1)/K^a}}.$$

Substituting this expression for α^a into the first equation, we get a single equation for a single unknown K^a.

- The algorithm itself is similar to the above Algorithm **C**, with the only difference that instead of the exact value $N_+^a(n)$, we use an approximate value $\lfloor \alpha^a \cdot n \rfloor$:

Algorithm D. *Suppose that we are given a set S with $|S| = N$ alternatives, and the weights $W_0 > W_1$. Then, to develop the search procedure, we first find α^a. After that, we construct the search procedure as follows.*

The search procedure is defined as a pair consisting of the set Ω of binary sequence and a function $A : \Omega \to 2^S$. In our algorithm, we will start with an empty set Ω and then add sequences to this set. As we add a sequence ω to the set Ω, we will also define the corresponding value $A(\omega)$. To be more precise, at each stage of our algorithm, we will have two sets:

- *a set Ω, with a function $A : \Omega \to 2^S$, and*
- *a set Ω_0 of sequences ω which we have already added to Ω and for which we still need to analyze "children" $\omega 0$ and $\omega 1$.*

We start with $\Omega = \Omega_0 = \emptyset$. Then, we add the empty sequence Λ to the set Ω, take $A(\Lambda) = S$, and add Λ to the set Ω_0. At each step of the algorithm, if $\Omega_0 \neq \emptyset$, we do the following for each sequence $\omega \in \Omega_0$:

- *if $|A(\omega)| = 1$, then no further questions are necessary, so we simply delete the sequence ω from the set Ω_0;*
- *if $N(\omega) = |A(\omega)| > 1$, then we:*
  - *add both children $\omega 1$ and $\omega 0$ to the set Ω;*
  - *select the corresponding question so that for $\lfloor \alpha^a \cdot N(\omega) \rfloor$ of the alternatives the answer is "yes", and for $N - \lfloor \alpha^a \cdot N(\omega) \rfloor$ of the alternatives the answer is "no", i.e., so that $|A(\omega 1)| = \lfloor \alpha^a \cdot N(\omega) \rfloor$ and $|A(\omega 0)| = N - \lfloor \alpha^a \cdot N(\omega) \rfloor$;*
  - *delete ω from the set Ω_0 of all sequences for which we are need to analyze children, and add the children $\omega 0$ and $\omega 1$ to this set.*

The construction ends when $\Omega_0 = \emptyset$.

Proposition 8. *For every N, W_0, and $W_1 < W_0$, the Algorithm* **D** *produces the asymptotically statistically optimal search procedure.*

5 Proofs

5.1 Proof of Proposition 1

The proof of this proposition is similar to standard dynamic programming-type proofs (see, e.g., [2]). Namely, in any search procedure, we subdivide the original set S with N elements into two subsets $A(1)$ and $A(0)$. Let N_+ denote the number of elements in the set $A(1)$; then the number of elements in the set $A(0)$ is $N - N_+$.

- If the alternative is in the set $A(1)$, then we spend W_1 on asking the first question, and then we spend $T(N_+)$ to look for this alternative in the set $A(1)$ with N_+ elements. Thus, for such alternatives, the largest possible effort is equal to $W_1 + T(N_+)$.

- Similarly, the largest possible weight for alternatives from $A(0)$ is $W_0 + T(N - N_+)$.

Overall, for any choice of N_+, the largest possible effort is equal to $\max\{W_1 + T(N_+), W_0 + T(N - N_+)\}$. Thus, the optimal search method corresponds to the choice of N_+ for which this effort is the smallest possible. The proposition is thus proven.

5.2 Proof of Proposition 2

The proof of this proposition is also similar to standard dynamic programming-type proofs (see, e.g., [2]).

5.3 Proof of Propositions 3 and 4

5.3.1 General idea

Let us denote the search procedure generated by the Algorithm **B** for a given N by B_N. We will prove that there exist constants $C > 0$ and $C_1 > 0$ such that for every N, we have

$$K \cdot \log_2(N) \le T(N)$$

and

$$P(B_N) \le K \cdot \log_2(N) + C - \frac{C_1}{N}.$$

By definition, $T(N)$ is the smallest effort of all possible procedures, thus, $T(N) \le P(B_N)$. So, if we prove the above two inequalities, we will indeed complete the proof of Propositions 3 and 4.

5.3.2 Proof of the first inequality

Let us first prove the first inequality by induction over N. The value $N = 1$ represents the induction base. For this value, $K \cdot \log_2(1) = 0 = T(1)$, so the inequality holds.

Let us now describe the induction step. Suppose that we have already proved the inequality $K \cdot \log_2(n) \le T(n)$ for all $n < N$. Let us prove that $K \cdot \log_2(N) \le T(N)$.

Due to Proposition 1, $T(N)$ is the smallest of the values $\max\{W_1 + T(N_+), W_0 + T(N - N_+)\}$ over $N_+ = 1, 2, \ldots, N-1$. So, to prove that $K \cdot \log_2(N)$ is indeed the lower bound for $T(N)$, we must prove that $K \cdot \log_2(N)$ cannot exceed each of these values, i.e., that $K \cdot \log_2(N) \le \max\{W_1 + T(N_+), W_0 + T(N - N_+)\}$ for every $N_+ = 1, 2, \ldots, N-1$. For these N_+, we have $N_+ < N$ and $N - N_+ < N$, so for all these values, we already know that $K \cdot \log_2(N_+) \le T(N_+)$ and $K \cdot \log_2(N - N_+) \le T(N - N_+)$. Therefore, $W_1 + K \cdot \log_2(N_+) \le W_1 + T(N_+)$, $W_0 + K \cdot \log_2(N - N_+) \le W_0 + T(N - N_+)$, and

$$\max\{W_1 + K \cdot \log_2(N_+), W_0 + K \cdot \log_2(N - N_+)\} \le \max\{W_1 + T(N_+), W_0 + T(N - N_+)\}.$$

So, to prove the desired inequality, it is sufficient to prove that

$$K \cdot \log_2(N) \le \max\{W_1 + K \cdot \log_2(N_+), W_0 + K \cdot \log_2(N - N_+)\}.$$

We will prove this inequality by considering two possible cases: $N_+ \le \alpha \cdot N$ and $N_+ \ge \alpha \cdot N$:

- When $N_+ \le \alpha \cdot N$, we have $N - N_+ \ge (1 - \alpha) \cdot N$ and therefore, $W_0 + K \cdot \log_2(N - N_+) \ge z$, where

  $$z = W_0 + K \cdot \log_2((1 - \alpha) \cdot N) = W_0 + K \cdot \log_2(N) + K \cdot \log_2(1 - \alpha).$$

  Here, by definition of α, we have $\alpha = 2^{-W_1/K}$, and by definition of K, we have $2^{-W_1/K} + 2^{-W_0/K} = 1$. Thus, $1 - \alpha = 2^{-W_0/K}$, hence $\log_2(1 - \alpha) = -W_0/K$, hence $W_0 + K \cdot \log_2(1 - \alpha) = 0$, and so $z = K \cdot \log_2(N)$. In this case,

  $$K \cdot \log_2(N) \le z = W_0 + K \cdot \log_2(N - N_+) \le \max\{W_1 + K \cdot \log_2(N_+), W_0 + K \cdot \log_2(N - N_+)\}.$$

- When $N_+ \geq \alpha \cdot N$, we have $W_1 + K \cdot \log_2(N_+) \geq z$, where

$$z = W_1 + K \cdot \log_2(\alpha \cdot N) = W_1 + K \cdot \log_2(N) + K \cdot \log_2(\alpha).$$

  By definition of α, we have $\alpha = 2^{-W_1/K}$, hence $\log_2(\alpha) = -W_1/K$, and $z = K \cdot \log_2(N)$. So, in this case,

$$K \cdot \log_2(N) \leq z = W_1 + K \cdot \log_2(N_+) \leq \max\{W_1 + K \cdot \log_2(N_+), W_0 + K \cdot \log_2(N - N_+)\}.$$

In both cases, we have the desired inequality. The induction step is proven, and so, indeed, for every N, we have $K \cdot \log_2(N) \leq T(N)$.

5.3.3 Proof of the second inequality

Let us now prove that there exist real numbers $C > 0$ and $C_1 > 0$ for which, for all N,

$$E(B_N) \leq K \cdot \log_2(N) + C - \frac{C_1}{N}.$$

To prove this inequality, we will pick a value N_0, prove that this inequality holds for all $N \leq N_0$, and then use mathematical induction to show that it holds for all $N > N_0$ as well.

Induction basis. Let us first find the conditions on C, C_1, and N_0 under which for all $N \leq N_0$,

$$E(B_N) \leq K \cdot \log_2(N) + C - \frac{C_1}{N}.$$

Subtracting $K \cdot \log_2(N)$ and adding $\frac{C_1}{N}$ to both sides of the this inequality, we get

$$C \geq \frac{C_1}{N} + E(B_N) - K \cdot \log_2(N)$$

for all N from 1 to N_0. So, to guarantee that this inequality holds, if we have already chosen N_0 and C_1, we can choose

$$C = \max_{1 \leq N \leq N_0} \left(\frac{C_1}{N} + E(B_N) - K \cdot \log_2(N) \right).$$

Induction step. Let us assume that for all $n < N$ (where $N > N_0$), we have proven that

$$E(B_n) \leq K \cdot \log_2(n) + C - \frac{C_1}{n}.$$

We would like to conclude that

$$E(B_N) \leq K \cdot \log_2(N) + C - \frac{C_1}{N}.$$

According to the definition of B_N, we have $E(B_N) = \max\{W_1 + E(B_{N_+}), W_0 + E(B_{N-N_+})\}$, where $N_+ = \lfloor \alpha \cdot N \rfloor$. Due to induction hypothesis, we have

$$E(B_{N_+}) \leq K \cdot \log_2(N_+) + C - \frac{C_1}{N_+}$$

and

$$E(B_{N-N_+}) \leq K \cdot \log_2(N - N_+) + C - \frac{C_1}{N - N_+}.$$

Therefore,

$$E(B_N) \leq \max\left\{ W_1 + K \cdot \log_2(N_+) + C - \frac{C_1}{N_+}, W_0 + K \cdot \log_2(N - N_+) + C - \frac{C_1}{N - N_+} \right\}.$$

Thus, to complete the proof, it is sufficient to conclude that this maximum does not exceed

$$K \cdot \log_2(N) + C - \frac{C_1}{N}.$$

In other words, we must prove that

$$W_1 + K \cdot \log_2(N_+) + C - \frac{C_1}{N_+} \le K \cdot \log_2(N) + C - \frac{C_1}{N} \tag{1}$$

and that

$$W_1 + K \cdot \log_2(N - N_+) + C - \frac{C_1}{N - N_+} \le K \cdot \log_2(N) + C - \frac{C_1}{N}.$$

Without losing generality, let us show how we can prove the first of these two inequalities. Since $N_+ = \lfloor \alpha \cdot N \rfloor$, the left-hand side of the inequality (1) can be rewritten as

$$W_1 + K \cdot \log_2(\alpha \cdot N) + K \cdot (\log_2(N_+) - \log_2(\alpha \cdot N)) + C - \frac{C_1}{N_+}.$$

We already know that $W_1 + K \cdot \log_2(\alpha \cdot N) = K \cdot \log_2(N)$. Thus, the left-hand side of (1) takes the simpler form

$$K \cdot \log_2(N) + K \cdot (\log_2(N_+) - \log_2(\alpha \cdot N)) + C - \frac{C_1}{N_+}.$$

Substituting this expression into (1) and canceling the terms $K \cdot \log_2(N)$ and C in both sides, we get an equivalent inequality

$$K \cdot (\log_2(N_+) - \log_2(\alpha \cdot N)) - \frac{C_1}{N_+} \le -\frac{C_1}{N}. \tag{2}$$

Let us further simplify this inequality. We will start by estimating the difference $\log_2(N_+) - \log_2(\alpha \cdot N)$. To estimate this difference, we will use the intermediate value theorem, according to which, for every smooth function $f(x)$, and for arbitrary two values a and b, we have $f(a) - f(b) = (a - b) \cdot f'(\xi)$ for some $\xi \in [a, b]$. In our case,

$$f(x) = \log_2(x) = \frac{\ln(x)}{\ln(2)},$$

$a = N_+$, and $b = \alpha \cdot N$. Here,

$$f'(\xi) = \frac{1}{\xi \cdot \ln(2)},$$

so

$$f'(\xi) \le \frac{1}{N_+ \cdot \ln(2)};$$

also, $|a - b| \le 1$, so, the difference $\log_2(N_+) - \log_2(\alpha \cdot N)$ can be estimated from above by:

$$\log_2(N_+) - \log_2(\alpha \cdot N) \le \frac{1}{N_+ \cdot \ln(2)}.$$

Hence, the above inequality holds if the following stronger inequality holds:

$$\frac{K}{N_+ \cdot \ln(2)} - \frac{C_1}{N_+} \le -\frac{C_1}{N},$$

or, equivalently,

$$\frac{C_1}{N} \le \frac{C_1 - K/\ln(2)}{N_+}. \tag{3}$$

Here, $N_+ \ge \alpha \cdot N - 1$, i.e.,

$$\frac{N_+}{N} \ge \alpha - \frac{1}{N_+}.$$

When $N \to \infty$, we have $N_+ \to \infty$ and $\frac{1}{N_+} \to 0$. Thus, for every $\varepsilon > 0$, there exists an N_0 starting from which $\frac{1}{N_+} \le \varepsilon$ and hence, $N_+ \ge (\alpha - \varepsilon) \cdot N$. For such sufficiently large N, the inequality (3) can be proven if we have

$$\frac{C_1}{N} \le \frac{C_1 - K/\ln(2)}{(\alpha - \varepsilon) \cdot N},$$

i.e., if we have

$$C_1 \le \frac{C_1 - K/\ln(2)}{\alpha - \varepsilon}. \tag{4}$$

Since $\alpha \le 1$, for sufficiently large C_1, this inequality is true. For such C_1, therefore, the induction can be proven and thus, Propositions 3 and 4 are true.

5.4 Proof of Propositions 5 and 6

The proof of these propositions is similar to standard dynamic programming-type proofs (see, e.g., [2]).

5.5 Proof of Propositions 7 and 8

This proof is similar to the above proof of Propositions 3 and 4. The only difference is in the equation which describes K^a and α^a. Let us show how these equations can be obtained.

In the worst-case description, we were using the fact that for the desired K, the function $f(N) = K \cdot \log_2(N)$ satisfies the continuous-variable analog of the equation from Proposition 1, i.e.,

$$f(N) = \min_{0 < N_+ < N} F(N, N_+),$$

where

$$F(N, N_+) = \max\{W_1 + f(N_+), W_0 + f(N - N_+)\}$$

and min is taken over all *real* numbers from the interval $(0, N)$. In this equation, this minimum is attained when $N_+ = \alpha \cdot N$.

Similarly, we are now looking for a function $f^a(N) = K^a \cdot \log_2(N)$ which satisfies the continuous-variable analog of the equation from Proposition 5, i.e., for which

$$f^a(N) = \min_{0 < N_+ < N} F^a(N, N_+),$$

where

$$F^a(N, N_+) = \left\{ \frac{N_+}{N} \cdot (W_1 + f^a(N_+)) + \frac{N - N_+}{N} \cdot (W_0 + f^a(N - N_+)) \right\},$$

min is taken over all real numbers from the interval $(0, N)$, and minimum is attained when $N_+ = \alpha^a \cdot N$.

If we substitute $f^a(N) = K^a \cdot \log_2(N)$ and $N_+ = \alpha^a \cdot N$ into the minimized expression $F^a(N, N_+)$, we conclude that

$$F^a(N, N_+) = \alpha^a \cdot (W_1 + K^a \cdot \log_2(\alpha^a \cdot N)) + (1 - \alpha^a) \cdot (W_0 + K^a \cdot \log_2((1 - \alpha^a) \cdot N)).$$

If we use the fact that $\log_2(a \cdot b) = \log_2(a) + \log_2(b)$, we can conclude that

$$F^a(N, N_+) = K^a \cdot \log_2(N) + \alpha^a \cdot W_1 + (1 - \alpha^a) \cdot W_0 + K^a \cdot (\alpha^a \cdot \log_2(\alpha^a) + (1 - \alpha^a) \cdot \log_2(1 - \alpha^a)).$$

The fact that this value is supposed to be equal to $f^a(N) = K^a \cdot \log_2(N)$ leads to the first equation from Proposition 5. the fact that the expression $F(N, N_+)$ attains its minimum when $N_+ = \alpha^a \cdot N$, we differentiate the expression $F(N, N_+)$ with respect to α^a and equate its derivative to 0. As a result, we get the second equation.

Acknowledgments

This work was supported in part by NASA under cooperative agreement NCC5-209, by NSF grants No. DUE-9750858 and CDA-9522207, by United Space Alliance, grant No. NAS 9-20000 (PWO C0C67713A6), by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518, and by the National Security Agency under Grant No. MDA904-98-1-0561.

The authors are thankful to Paul Wang and to Chitta Baral for helpful discussions.

References

[1] D. Carnegie, *How To Win Friends and Influence People*, Simon and Schuster, N.Y., 1981.

[2] Th. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, MIT Press, Cambridge, MA, and Mc-Graw Hill Co., N.Y., 1994.

[3] E. N. Kamoroff, *How to Extract Knowledge From an Expert so That His Effort is Minimal*, Master Project, Computer Science Department, University of Texas at El Paso, 1993.