Competitive learning of monotone Boolean functions

Sascha Kurz[a]

[a] *University of Bayreuth, Department of Mathematics, D-95440 Bayreuth, Germany*

Key words: monotone Boolean functions, exact learning algorithms, competitive analysis
2000 MSC: 68W27, 06E30, 68Q32

1. Introduction

Complex systems with many components and resource restrictions can crash if certain combinations of the components are simultaneously active. As a demonstrative example one can imagine a computer system with different software packages. For n software packages there are 2^n combinations where the computer can either crash or work properly. The behavior of the computer system can be described by a Boolean function $g : \{0,1\}^n \to \{0,1\}$. Suppose we have the possibility to evaluate the underlying Boolean function at arbitrary points. In practice we might think of asking an expert or performing an experiment like simply running the respective set of software packages. In our abstract setting we speak of asking a question. For an arbitrary Boolean function in any case 2^n questions are necessary (and sufficient if all questions are pairwise different) to unveil the entire function. Fortunately in many applications we can assume some restrictions. In our example it is quite reasonable to assume some kind of monotonicity. If the computer crashes for a certain subset $S \subseteq N := \{1,\dots,n\}$ of the programs we can assume that it also crashes for every superset $N \supseteq T \supseteq S$ of the programs. Similarly, if the computer works properly for a set $S \subseteq N$ then it should also work properly for every subset $T \subseteq S$. So we restrict the underlying function to the class of monotone Boolean functions. For the ease of notation we write a Boolean function as $f : 2^N \to \{0,1\}$ in the following, where 2^N denotes the set of subsets of N.

Since asking questions or performing experiments can be quite expensive one naturally tries to minimize the number of necessary questions. Different concepts like worst case or average case analysis have been applied on this problem so far. As shown shown by Engel [3] up to $\binom{n}{\lfloor\frac{n}{2}\rfloor} + \binom{n}{\lfloor\frac{n}{2}\rfloor+1}$ questions are necessary to uniquely verify the worst case examples. There are algorithms, see e.g. [5], which achieve this unavoidable worst case bound. Since there are monotone Boolean functions which can be uniquely verified asking a single question, those, with respect to worst case analysis, optimal algorithms might not be adequate in all practical applications. Thus there are studies in the literature minimizing the average number of necessary questions while assuming an uniform distribution of the possible function, see e.g. [8]. In this paper we want to study exact learning of monotone Boolean functions using competitive analysis. This concept has the big advantage that no assumptions on the distribution of the occurring functions are necessary.

2. Preliminaries

We call a function $f : 2^N \to \{0,1\}$ a monotone Boolean function if $f(S) = 1$ implies $f(T) = 1$ for all $N \subseteq T \subseteq S$ and $f(S) = 0$ implies $f(T) = 0$ for all $T \subseteq S \subseteq N$. The set $\mathcal{L}$ of the maximal lower sets consists of the sets $S \subseteq N$ with $f(S) = 0$ where $f(T) = 1$ for all proper supersets of S. Similarly the set $\mathcal{U}$ of the minimal upper sets consists of the sets $S \subseteq N$ with $f(S) = 1$ where $f(T) = 0$ for all proper subsets of S. We would like to remark that either $\mathcal{L}$ or $\mathcal{U}$ suffice to uniquely characterize f within

Email address: sascha.kurz@uni-bayreuth.de (Sascha Kurz)

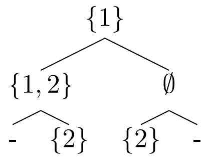
the class of Boolean functions, i.e. we have $f(S) = 1$ if and only if there exists an $U \in \mathcal{U}$ with $U \subseteq S$ or $f(T) = 0$ if and only if there exists an $L \in \mathcal{L}$ with $T \subseteq L$.

Via asking $f(S)$ we do not get the information whether S is an inclusion-maximal lower or an inclusion-minimal upper set but only $f(S) = 1$ or $f(S) = 0$. So in general asking the sets from $\mathcal{L}$ or $\mathcal{U}$ is not sufficient.

Lemma 2.1. *A monotone Boolean function f is uniquely characterized if and only if all values of $\mathcal{L}$ and $\mathcal{U}$ are given.*

PROOF. Suppose there is a single $U \in \mathcal{U}$ whose value is not known. We construct a Boolean function f' by setting $f'(S) = f(S)$ for all subsets $S \neq U$ and $f'(U) = 0$. This function is also monotone since for all proper subsets $T \subsetneq U$ we have $f'(T) = f(T) = 0$. If there is a single $L \in \mathcal{L}$ whose value is not known we can consider the monotone Boolean function f'' with $f''(S) = f(S)$ for all $S \neq L$ and $f''(L) = 1$. $\square$

So let us denote by $m(f)$ the cardinality $|\mathcal{U} \cup \mathcal{L}|$, i.e. a lower bound for each deterministic algorithm to reconstruct f via asking questions. As notation for those algorithms we use binary decision trees, where we use the questions as nodes and where the answer “0” corresponds to the left successor:



In this example for $n = 2$ the first question is $\{1\}$. Once the answer is 0 the algorithm continues by asking $\{1, 2\}$. If the answer is again 0 then there is only one possible monotone Boolean function (the all-zero function f_0: $f_0(S) = 0$ for all $S \subseteq N$) left and we have reconstructed f. By $A(f)$ we denote the number of questions asked by algorithm A to reconstruct f. In our example we have $A(f) = 2$ while $m(f) = 1$ would be possible for the optimal algorithm. In competitive analysis the fraction $\frac{A(f)}{m(f)}$ is studied.

We call a deterministic algorithm A reconstructing an n-variable monotone Boolean function c-competitive for a real number $c \geq 1$ if $\frac{A(f)}{m(f)} \leq c$ for all $f \in \mathcal{M}_n$, where $\mathcal{M}_n$ denotes the set of monotone Boolean functions on n variables. The best possible competitivity is denoted by $c_n^\star$, i.e. the infimum of the possible c for c-competitive algorithms on n-variables.

Let us at first comment on the competitivity of some classical learning algorithms. The Hansel's algorithm [5] behaves very badly using this measure, e.g. for the all-one function f_1 ($f_1(S) = 1$ for all $S \subseteq N$) with $m(f_1) = 1$ at least $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ questions are asked (there is some freedom in the definition of Hansel's algorithm), see e.g. [8]. Thus Hansel's algorithm is not c-competitive for $c < \binom{n}{\lfloor \frac{n}{2} \rfloor}$ while being worst-case optimal.

Another algorithm for learning a monotone Boolean function is the so called FIND-BORDER algorithm of Gainanov [4] (which is used as a subroutine in several other learning algorithms). In each iteration an element of $\mathcal{U}$ is determined and verified using at most $n + 1$ questions. Thus the FIND-BORDER is $n + 1$-competitive for all $n \in \mathbb{N}$. We would like to remark that a refined analysis shows that at most $n \cdot |\mathcal{U}| + 1 + |\mathcal{L}|$ questions are asked and that it can be slightly adopted to yield an n-competitive algorithm for $n \geq 2$. (If the elements of $\mathcal{L}$ are iteratively determined then at most $n \cdot |\mathcal{L}| + 1 + |\mathcal{U}|$ questions are asked.)

The enumeration of the set $\mathcal{M}_n$ is a classical combinatorial problem known as Dedekind's problem [2]. So far the exact numbers could be determined only up to $n = 8$ and are given by 3, 6, 20, 168, 7 581, 7 828 354, 2 414 682 040 998, and 56 130 437 228 687 557 907 788, see e.g. [7]. To factor out symmetry we call two monotone Boolean functions f and g equivalent if there is a bijection σ on N such that $f(S) = g(\sigma(S))$ for all $S \subseteq N$. The number of inequivalent monotone Boolean functions (or orbits) are given by 3, 5, 10, 30, 210, 16353, see e.g. [6] , but grow nevertheless double exponentially.

3. Lower bounds on the optimal competitivity

Based on the fact that the all-zero function f_0 and the all-one function f_1 need only one question to be completely reconstructed, we can state $c_n^\star \geq 2$ for all $n \in \mathbb{N}$. By $b_i(n)$ we denote the number of monotone Boolean functions on n variables with $m(f) = i$. Since the answer to each question splits the set of monotone Boolean functions which are compatible with the answers so far into two subsets of remaining candidates, we have:

Lemma 3.1.

$$c_n^\star \geq \frac{\left\lceil \log_2\left(\sum\limits_{j=1}^{i} b_j(n)\right)\right\rceil}{i} \quad \forall i \geq 1.$$

Lemma 3.2.

$$b_{i+1}(n) \geq \binom{n}{i} \quad \forall 1 \leq i \leq n.$$

PROOF. Let S be an arbitrary i-element subset of n. For the monotone Boolean function with unique maximal lower set $N\backslash S$ the minimal upper sets correspond to the elements of S. □

Corollary 3.3. *For each $\varepsilon > 0$ there is a $n_0(\varepsilon)$ such that $c_n^\star \geq (1-\varepsilon)\log_2 n$ for all $n \geq n_0(\varepsilon)$.*

Lemma 3.4. *If $U \in \mathcal{U}$ then $|\mathcal{L}| \geq |U|$.*

PROOF. Each $L \in \mathcal{L}$ can at most contain one set $U\backslash\{i\}$ with $i \in U$ as a subset. □

Using this one can easily determine $b_1(n) = 2$, $b_2(n) = n$, $b_3(n) = 2\binom{n}{2}$, and $b_4(n) = 8\binom{n}{3}$.

Lemma 3.5.

$$c_{n+1}^\star \geq c_n^\star.$$

PROOF. Let A be a deterministic online algorithm for $n+1$ variables. We can obtain an online algorithm A' for n variables by adjusting each question S to $S\backslash\{n+1\}$ and slightly adapting the final output.

Let f be an arbitrary monotone Boolean function on n variables and g be a monotone Boolean function on $n+1$ variables defined via $f(S) = g(S) = g(S \cup \{n+1\})$ for all $S \subseteq \{1, \dots, n\}$. Due to $\mathcal{U}(g) = \{U \mid U \in \mathcal{U}(f)\}$ and $\mathcal{L}(g) = \{L \cup \{n+1\} \mid L \in \mathcal{L}(f)\}$ we have $m(g) = m(f)$. □

4. Optimal algorithms for small n

We call a deterministic online algorithm to reconstruct a monotone Boolean function or its corresponding binary decision tree reasonable if only sets are asked whose function value cannot be deduced from previous answers. For $n = 1$ variable there are only two reasonable binary decision trees, each having a competitivity of $c_1^\star = 2$. For $n = 2$ variables an example with competitivity $c_2^\star = 2$ is given in Section 2.

Our next aim is to prove that every deterministic online algorithm for $n = 3$ variables has a competitivity of at least $\frac{5}{2}$. To conclude a lower bound on the competitivity it suffices to give a sequence of answers to the questions of the algorithm that are compatible with a monotone Boolean function. By choosing a suitable sequence of answers for a given deterministic algorithm we can conclude the tight lower bound. Since we use only a path of the binary decision tree, consisting of the sequence on questions, the same sequence of answers results in the same lower bound for a large set of binary decision trees. We can further reduce the set of candidates of paths by utilizing symmetry. Therefore we call two such paths $P_1 = (S_1, \dots, S_l)$ and $P_2 = (T_1, \dots, T_l)$, where $S_1, \dots, S_l, T_1, \dots, T_l \subseteq N$, equivalent if there is a bijection σ of N fulfilling $\sigma(S_i) = T_i$ for all $1 \le i \le l$. It suffices to consider inequivalent paths only, e.g. we can assume that the first question is either $\emptyset$, $\{1\}$, $\{1,2\}$, or $\{1,2,3\}$.

Suppose that the answer to this first question S_1 is one if $|S_1| \ge 2$ and zero otherwise, then $S_2 \in \{\emptyset, N\}$ since otherwise we could not exclude the all-one function f_1 or the all-zero function f_0 and would end up with an algorithm having a competitivity of at least 3. We can abstain from further considering the initial path segments $(\emptyset, N)$ and $(N, \emptyset)$ since the initial path segments $(\{1\}, N)$ and $(\{1,2\}, \emptyset)$ yield more information about the unknown Boolean function. In Figure 1 we depict the remaining part of our argument graphically. Each vertex is labeled with question and answer. At the leafs we further specify a compatible monotone Boolean function with $m(f) = 2$ via $\mathcal{U}$ and $\mathcal{L}$. Since each leaf has height 3 and the corresponding elements of $\mathcal{U}$ and $\mathcal{L}$ have not been asked so far, $\frac{3+2}{2}$ is a lower bound for the competitivity in each case.

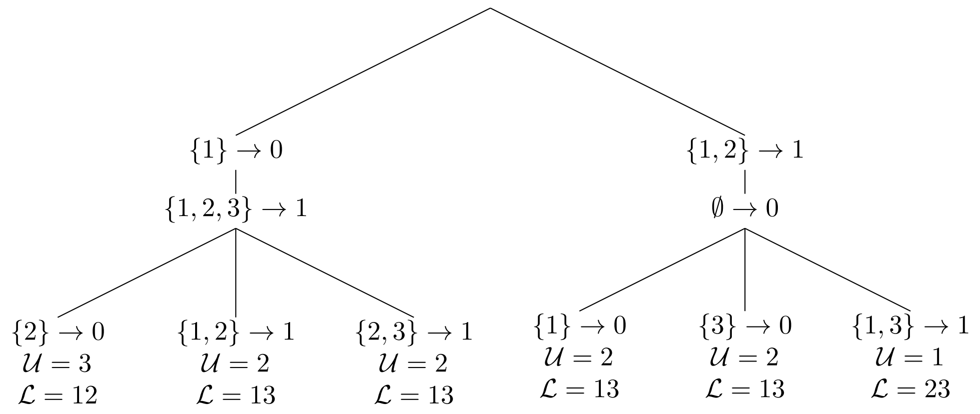

$\{1\} \to 0$
$\{1,2,3\} \to 1$
$\{1,2\} \to 1$
$\emptyset \to 0$
$\{2\} \to 0$ $\mathcal{U} = 3$ $\mathcal{L} = 12$
$\{1,2\} \to 1$ $\mathcal{U} = 2$ $\mathcal{L} = 13$
$\{2,3\} \to 1$ $\mathcal{U} = 2$ $\mathcal{L} = 13$
$\{1\} \to 0$ $\mathcal{U} = 2$ $\mathcal{L} = 13$
$\{3\} \to 0$ $\mathcal{U} = 2$ $\mathcal{L} = 13$
$\{1,3\} \to 1$ $\mathcal{U} = 1$ $\mathcal{L} = 23$

Figure 1: Lower bound for three variables.

For the other direction we describe a whole class of $\frac{5}{2}$-competitive online algorithms in Figure 2. As for a binary decision tree the internal vertices are labeled with the questions of the algorithm. The leafs are either labeled with $[c]$ or $[u, k, c]$. In the first case there is only a unique monotone Boolean function being compatible with the previous answers left, so that this path is c-competitive. In the later case there are only u unclassified sets, i.e. sets whose value cannot be deduced from the previous answers and the minimum $m(f)$ of the remaining compatible monotone Boolean functions is k so that every reasonable continuation of the online algorithm is c-competitive in the subtree starting at this leaf. Thus we have $c_3^\star = \frac{5}{2}$.

Using the same ideas and larger trees one can show $c_4^\star = \frac{8}{3}$ and $c_5^\star \ge 3$.

5. Conclusion

We have considered the problem of minimizing the number of questions to an oracle to completely reconstruct an unknown monotone Boolean function from the perspective of competitive analysis. The classical algorithm of Hansen turns out to perform pretty bad using this measure. For given general monotone Boolean functions bounds on the best possible competitivity are far from being tight. As shown in [7] a *typical* monotone Boolean function, i.e. almost all of $\mathcal{M}_n$ functions, fulfills $m(f) \ge |\mathcal{L}| \ge \frac{1}{2}\binom{n}{\lceil \frac{n}{2} \rceil} - n2^{n/2}$. Thus most reasonable algorithms have a constant competitivity on almost all inputs. So the challenge is to deal with those monotone Boolean functions with atypically *small* $m(f)$. On the other hand the ratio between $|\mathcal{L}|$ and $|\mathcal{U}|$ can become exponential, see [1].

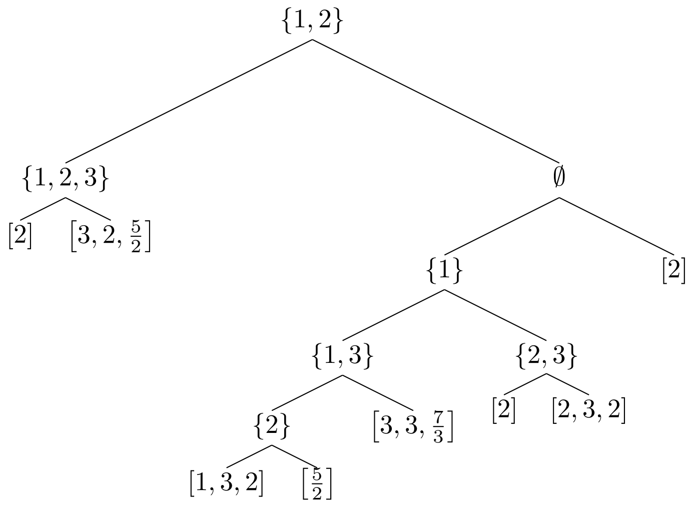


Figure 2: A class of $\frac{5}{2}$-competitive algorithms for three variables.

Here we have determined optimal algorithms for rather small problem instances only. On the other hand the described methods and shortcuts may be used in order to implement a non-trivial search to determine the next exact values of $c_n^\star$. (This is indeed what we plan to do next.) A direct exhaustive search on all reasonable binary decision trees seems impracticable even for rather small n.

References

[1] D. Angluin, *Queries and concept learning*, Machine Learning **2** (1988), 319–342.

[2] R. Dedekind, *On the decomposition of numbers by means of their greatest common divisors. (über Zerlegungen von Zahlen durch ihre größten gemeinsamen Teiler.)*, Festschrift Hoch. Braunschweig u. ges. Werke (1897), 103–148 (German).

[3] K. Engel, *Sperner theory*, Encyclopedia of Mathematics and Its Applications. 65. Cambridge: Cambridge University Press. ix, 417 p., 1997.

[4] D. N. Gainanov, *On one criterion of the optimality of an algorithm for evaluating monotonic Boolean functions*, U.S.S.R. Computational Mathematics and Mathematical Physics **24** (1984), 176–181.

[5] G. Hansel, *Sur le nombre des fonctions booléennes monotones de n variables*, C.R. Acad. Sci. Paris **262** (1966), no. 20, 1088–1090 (French).

[6] OEIS Foundation Inc., *The on-line encyclopedia of integer sequences*, http://oeis.org/A003182, 2011.

[7] A. D. Korshunov, *Monotone boolean functions*, Russ. Math. Surv. **58** (2003), no. 5, 929–1001 (English. Russian original).

[8] V. I. Torvik and E. Triantaphyllou, *Minimizing the average query complexity of learning monotone Boolean functions*, INFORMS Journal on Computing **14** (2003), no. 2, 144–174.