Calculating selection probabilities under lexicase selection is NP-Hard

Emily Dolson
dolsonem@msu.edu
Michigan State University
East Lansing, Michigan, USA

ABSTRACT

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Emily Dolson. 2023. Calculating selection probabilities under lexicase selection is NP-Hard. In *Proceedings of Genetic and Evolutionary Computation Conference Companion (GECCO 2023).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

2 PRELIMINARIES

2.1 Definition of the Lexicase Selection Probabilities Problem

Definition 2.1. Input: a vector, P, of N vectors of length M.

Output: a vector of length N indicating the probability of each vector in P being selected in a single round of lexicase selection.

2.2 Satisfiability (SAT)

v variables, c clauses (note, we are using v rather than the traditional n to avoid confusion with the n in lexicase)

2.3 Brief review of computational complexity theory

For readers unfamiliar with the $NP-Hard$ class of problems and how we prove that problems are in this class, we offer a capsule summary.

In order to talk about how hard problems are, we place them into classes. Many problems that we regularly encounter are in the class NP (non-deterministically polynomial), which is the set of problems for which we can verify in polynomial time that a given solution is correct. Some problems in NP are "easy" in the sense that polynomial-time solutions for them have been found. For the hardest problems in NP, no polynomial-time solution has been found thus far and it is thought to be unlikely that one exists. $NP-Hard$ problems are the class of problems that are at least as hard as the hardest problems in NP. Thus, if a problem is $NP-Hard$, it is unlikely to be solvable in polynomial time. If calculating lexicase selection probabilities is $NP-Hard$, that is useful to know because it tells us that efforts to find a polynomial time algorithm to calculate them are unlikely to pay off.

To prove that a problem is $NP-Hard$, we must take a problem that we already know is $NP-Hard$ show that it reduces (in polynomial time) to the problem we're interested in. Here, we will reduce SAT to LEX-PROB, i.e. we will convert instances of SAT to instances of LEX-PROB that, when solved, will also tell us the solution to the underlying SAT instance. By showing that any instance of SAT can be converted to an equivalent instance of LEX-PROB in polynomial time, we will prove that LEX-PROB is at least as hard as SAT. If we came up with a polynomial time algorithm that solved LEX-PROB, that would mean we also have a polynomial time algorithm for SAT (and, indeed, all of the $NP-Complete$ problems). As the suspicion of most experts is that SAT cannot be solved in polynomial time, this would mean that LEX-PROB likely cannot be solved in polynomial time either.

3 CALCULATING PROBABILITIES UNDER ϵ LEXICASE SELECTION IS NP-HARD

The lexicase selection probabilities problem is not in the set NP. There is no polynomial-sized certificate that could be returned that would allow someone to verify that the assigned probabilities are correct. Thus, the lexicase selection probabilities problem is not NP-Complete. However, we will show that it is NP-Hard.

We will show that . Or, more formally:

SAT $\leq_P$ ϵ-LEX-PROB-DECISION $\leq_P$ LEX-PROB-DECISION $\leq_P$ LEX-PROB

THEOREM 3.1. *The* ϵ-LEX-PROB *problem is NP-Hard.*

We will prove this theorem by reducing Satisfiability (SAT), a well-known NP-Complete problem [CITE Karp], to ϵ-LEX-PROB in polynomial time.

PROOF. Given an instance of SAT with c clauses and v variables, we can create an instance of ϵ-LEX-PROB with a population containing $N = 1 + c * v * 2 + v$ vectors of size $M = c * v * 2 + v * 2$. ϵ will be set to .1.

GECCO 2023, 2023, Portugal

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

The first $c * v * 2$ positions in each vector correspond to the clauses in the original SAT instance. Each clause has $v * 2$ positions, one for each variable and the negation of each variable. We will refer to these positions as C_{CiXj}, where i is the index of the clause the position is part of and j is the id of the variable the position refers to. For example, C_{C2X3} would be the position corresponding to the value of the variable $X3$ in clause 2.

The next $v * 2$ positions in each vector are the ones that will be used to explore different variable assignments. The order in which lexicase selection picks from these positions will determine which variable assignments are made. We will refer to the position corresponding to choosing to set Xi to true as D_{Xi} (whereas the position that corresponds to setting it to false would be $D_{!Xi}$).

The first vector in this population represents the SAT instance. We will call this the focal vector. Its C_{CiXj} values will be set to indicate which variables are in each clause. If clause i contains variable j, then C_{CiXj} in the focal vector is set to 1. Otherwise, it is set to 0. All D_{Xi} values are set to .9.

The next $c * v * 2$ vectors in the population correspond to possible variable assignments for each clause. We will call these the variable vectors. The C_{CiXj} positions in these vectors are identical to the focal vector except in the clause corresponding to that variable vector. There, the position connected to the negation of the variable corresponding to the vector will be set to 2.

D_{Xi} values for the variable vectors are set to .9, except in the position corresponding to that variable and its negation. For variable vectors corresponding to Xi, $D_{!Xi}$ will be set to .8, and D_{Xi} will be set to 1. For variable vectors corresponding to $!Xi$, $D_{!Xi}$ will be set to 1, and D_{Xi} will be set to .8.

Lastly, there are v vectors that we will refer to as "timing" vectors. These vectors have all C_{CiXj} positions set to 3. Each timing vector corresponds to a single variable and its negation. D_{Xi} and $D_{!Xi}$ are set to 0 when i is that vector's corresponding variable. Otherwise, they are set to .9.

The timing vectors insure that, before any of the C_{CiXj} positions are selected, values have to be assigned to all variables by selecting either D_{Xi} or $D_{!Xi}$ for all i. Any ordering of positions in which a C_{CiXj} position is selected prematurely will result in one of the timing vectors winning.

When D_{Xi} is selected, it eliminates all variable vectors corresponding to $D_{!Xi}$, and visa versa, because .8 is less than 1 by more than ϵ. The other vectors, however, are not eliminated, as .9 is within ϵ of 1. Selecting D_{Xi} or $D_{!Xi}$ also eliminates the timing vector corresponding to variable i, which has 0s in both positions. Selecting a variable after selection its negation or the other way around will have no effect, as all remaining values for that position will be .8 or .9 and so will be within ϵ of each other and tie.

Once variable assignments have been made, C_{CiXj}s can now safely be selected. The focal vector will only have a chance of winning if, for each clause, there is at least one position where the focal vector has a 1 and no variable vectors with 2s remain in contention.

The different objective orderings in lexicase selection can be thought of as a decision tree with $N!$ leaf nodes, each representing a different ordering of objectives. Each level of the tree represents choosing the next objective, and so each node at level i has $N - i$ child nodes (since i objectives have already been selected, there are $N - i$ options remaining).

For the purposes of our reduction, we only care about the portions of this tree in which one of the D_{Xi} positions for each variable is ordered before any of the C_{CiXj} positions. These orderings represent trying all possible combinations of truth assignments to the variables. The focal vector will have a non-zero chance of selection if and only if one of those combinations of truth assignments corresponds to a truth assignment that satisfies the original SAT instance. Therefore, this transformation is a valid reduction from SAT to LEX-PROB.

This reduction can be done in polynomial time.

Therefore, SAT reduces in polynomial time to ϵ-LEX-PROB, meaning that ϵ-LEX-PROB must be at least as hard as SAT. Since SAT is NP-Complete, ϵ-LEX-PROB must be NP-Complete as well.

□

4 CALCULATING PROBABILITIES UNDER LEXICASE SELECTION IS NP-HARD

We have shown that ϵ-LEX-PROB is NP-Hard. However, that does not necessarily mean that LEX-PROB is NP-Hard. Perhaps the inclusion of the ϵ parameter makes ϵ-LEX-PROB substantially harder. Thus, to prove that LEX-PROB is also NP-Hard, we will reduce ϵ-LEX-PROB to LEX-PROB in polynomial time.

THEOREM 4.1. *The* LEX-PROB *problem is NP-Hard.*

PROOF. Given an instance of ϵ-LEX-PROB with N vectors of size M and some value ϵ, we will create an instance of LEX-PROB with $N + N^2M$ vectors of size NM.

The key property of ϵ-lexicase is that it makes multiple sets of ties possible within the same objective, depending on which vectors have yet to be eliminated from the population.

Thus, for each objective in our original ϵ-LEX-PROB instance, we will create $N - 1$ objectives in our LEX-PROB instance. Each of these objectives will contain only 0s and 1s. We will refer to these objectives as O_{ij}, where i is the objective it corresponds to in the ϵ-LEX-PROB instance, and j refers to the id of this objective among the other objectives that also correspond to i. Values for O_{ij} will be 1 if they are greater than or equal to the jth highest value that any vector in the ϵ-LEX-PROB instance had for objective i minus ϵ. Otherwise they will be 0. We will call this set of vectors the focal vectors.

$$O_{ijk} = 1 if V_{ki} >= S_j - \epsilon else 0 \quad (1)$$

By adding these objectives, we insure that every set of vectors that was able to tie on an objective in the original ϵ-LEX-PROB instance will be able to tie on an objective in the

new LEX-PROB instance. However, adding these objectives also creates the ability for ties to occur in orders that were impossible in the original ϵ-LEX-PROB instance.

To fix this problem, we add N^2M additional "timing" vectors, t_{fij}. Each of these vectors corresponds to a focal vector (f) and an O_ij. Each timing vector is identical to its corresponding focal vector, with the exception of the values of objectives O_{i0} through O_{iJ} (i.e. all of the values corresponding to objective i in the original ϵ-LEX-PROB instance). O_{ij} is set to -1. All objectives $O_{ij'<j}$ are set to 0, and all objectives are set to $ij' < j$ are set to 2.

This set up insures that in any ordering of objectives where the various objectives O_{ij} are chosen in the wrong order (i.e. an order other than strictly increasing j), one of the timing vectors would immediately be selected, as its value of 2 would be higher than any value a focal vector has for that objective. Only by selecting the O_{ij} objectives in increasing order of j can timing vectors for objective set O_i be eliminated.

What if other objectives are selected between two successive objectives O_{ij-1} and O_{ij}? This possibility does not actually present a problem. Only one objective in each set O_i will ever directly cause a focal vector to be eliminated from consideration. To understand why, it is helpful to consider some properties of the set of objectives O_{ij}. As j increases, so do the number of vectors with 1s for this objective. The vectors with 1s for O_{ij-1} are a subset of the vectors with 1s for O_{ij}. In other words, as j increases O_{ij} becomes more permissive in which vectors it will allow to persist. If O_{ij} is selected and a vector with a value of 1 for O_{ij} is still eligible for selection, all values with 0s for O_{ij} will be eliminated. Because the vectors that O_{ij+1} could eliminate are a subset of the ones that O_{ij} eliminated, choosing O_{ij+1} after O_{ij} has eliminated vectors from contention will have no effect.

The only circumstance in which choosing O_{ij+1} after choosing O_{ij} will have an effect is when choosing O_{ij} had no effect because there were no vectors with a value of 1 for O_{ij} that were still eligible to be selected (and thus all remaining vectors tied with a score of 0).

Therefore, only one objective in each set O_i will actually have an effect on selection. Consequently, we do not need to worry about the order that they are selected in beyond requiring that each objective O_{ij} be selected in increasing order of j.

Due to the combinatorics involved, the selection probabilities returned by the new instance of LEX-PROB will be different from the probabilities under ϵ-LEX-PROB. Fortunately, we are dealing with the decision versions of these problems, which simply ask whether a vector has a non-zero probability of selection.

□

5 SOFTWARE???

6 DISCUSSION

7 CONCLUSION

ACKNOWLEDGMENTS

We would like to thank the ECODE lab

REFERENCES