Brief Announcement: Certified Multiplicative Weights Update

Verified Learning Without Regret

Alexander Bagnall
Ohio University
ab667712@ohio.edu

Samuel Merten
Ohio University
sm137907@ohio.edu

Gordon Stewart
Ohio University
gstewart@ohio.edu

ABSTRACT

The Multiplicative Weights Update method (MWU) is a simple yet powerful algorithm for learning linear classifiers, for ensemble learning à la boosting, for approximately solving linear and semidefinite systems, for computing approximate solutions to multicommodity flow problems, and for online convex optimization, among other applications.

In this brief announcement, we apply techniques from interactive theorem proving to define and prove correct the first formally verified implementation of MWU (specifically, we show that our MWU is no regret). Our primary application – and one justification of the relevance of our work to the PODC community – is to verified multi-agent systems, such as distributed multi-agent network flow and load balancing games, for which verified MWU provides a convenient method for distributed computation of approximate Coarse Correlated Equilibria.

CCS CONCEPTS

•**Theory of computation** → **Program verification;** *Algorithmic game theory; Convergence and learning in games;* Multi-agent learning; Network games; •**Software and its engineering** → *Distributed systems organizing principles;*

KEYWORDS

The Multiplicative Weights Update Method; Interactive Theorem Proving; Coq

1 INTRODUCTION

The Multiplicative Weights Update method (MWU, [1, 6]) solves the general problem of "combining expert advice", in which an agent repeatedly chooses which action, or "expert", to play against an adaptive environment. The agent, after playing an action, learns from the environment both the cost of that action and of other actions it could have played in that round. The environment, in turn, may adapt in order to minimize environment costs. MWU works by maintaining a weighted distribution over the action space, in which each action initially has equal weight, and by updating weights with a linear or exponential loss function to penalize poorly performing actions.

PODC'17, July 25-27, 2017, Washington, DC, USA

ACM ISBN 978-1-4503-4992-5/17/07.
DOI: http://dx.doi.org/10.1145/3087801.3087852

This simple algorithm performs remarkably well: In number of rounds logarithmic in the size of the action space, MWU's expected cost approaches to within a small bound ϵ that of the best fixed action the agent could have chosen in hindsight (MWU has bounded external regret). In [1], Arora, Hazan, and Kale showed that MWU has wide-ranging connections to numerous problems in computer science, including optimization, linear and semidefinite programming, and machine learning (cf. boosting [4]).

Our work targets another important application of MWU that is perhaps of greater interest to the PODC community: the approximate solution of multi-agent games, especially as such games relate to the construction of distributed systems. It is well known (cf. [7, Chapter 4]) that no-regret algorithms converge, in expectation when played by multiple independent agents, to a large equilibrium class known as Coarse Correlated Equilibria (CCEs). CCEs may not be socially optimal, but for some games (e.g., Roughgarden's smooth games [8]) the social, or objective, cost of such equilibrium states can be bounded with respect to the optimal cost (the Price of Anarchy, or POA, of the game). Our broader research program, the Cage project:

https://github.com/gstew5/cage

seeks to use such results and others from algorithmic game theory and distributed optimization to build distributed systems – e.g., distributed network routers and load balancers – that have verified convergence and correctness properties by design.

Contributions. In promotion of the first part of our broader research program, this brief announcement reports on the design, construction, and verification of the first formally certified implementation of the MWU algorithm, available open-source on the Cage project's website.

By *verified*, we mean our MWU implementation has mechanically checked convergence bounds and correctness proof within an interactive theorem prover (specifically, Ssreflect [5], an extension of the Coq [2] system). By *convergence* and *correctness*, we mean that we prove both that MWU produces the right answer (functional correctness wrt. a high-level functional specification), but also that it does so with external regret bounded by a function of the number of iterations of the protocol (convergence).

As we've mentioned, MWU has broad application across a number of subdisciplines of computer science, including linear programming, optimization, and machine learning. Our work uses MWU to implement no-regret dynamics, a general strategy for computing the CCEs of multi-agent games. By formally proving correctness and convergence results for our implementation of MWU, we demonstrate a new architecture for formal verification of (a subclass of) distributed systems: represent the system as a game, prove POA bounds for the game, then compose the POA results

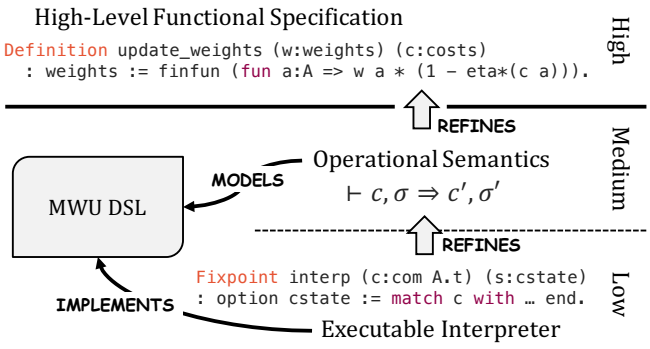

Figure 1: MWU Architecture

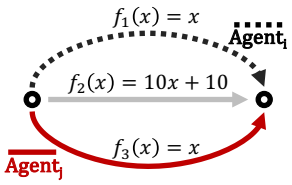

Figure 2: Routing game with source s, sink t, and affine cost functions $f_e(x) = a_e x + b_e$ where x is the amount of traffic on an edge e. Solutions of the game are assignments of players to source-sink paths. An optimal solution minimizes the total cost to all players.

with our general proof of correctness and convergence of MWU – which drives any such game to an approximate CCE, or ϵ-CCE – to yield correctness and performance results of the overall system. Section 3 sketches how these pieces fit together when applied to problem of distributed routing with affine latency functions.

2 MWU IMPLEMENTATION AND PROOF

Our implementation and proof of MWU (Figure 1) were designed to be extensible. At a high level, the structure of the proof follows the program refinement methodology, in which a high-level mathematical but inefficient specification of the MWU algorithm (High-Level Functional Specification) is gradually made more efficient by a series of refinements to various features of the program (for example, by replacing an inefficient implementation of a key-value map with a more efficient balanced binary tree).

For each such refinement, we prove that every behavior of the lower-level program is one of the acceptable behaviors of the higher-level program it refines. Thus specifications proved for all behaviors of the high-level program also apply to each behavior at the low level. By behavior here, we mean the trace of action distributions output by MWU as it interacts with, and receives cost vectors from, the environment.

In order to make our MWU implementation and proof extensible, we factor the lower implementation layers (Medium and Low) into an interpreter and semantics over a domain-specific language specialized to MWU-style algorithms. The DSL defines commands for updating the weights table as well as commands for interacting with the environment, in the style of process calculi or message-passing concurrency.

At the top level, the convergence theorem we prove of our high-level functional MWU is:

```
Theorem perstep_weights_boundedregret :
  (expCostsR − OPTR)/T ≤ etaR + (ln size_A) / (etaR*T).
```

Here expCostsR is the expected cost of MWU on a sequence of cost vectors, OPTR is the cumulative cost of the best fixed action, etaR is the algorithm's exploration parameter η (required to lie in the range $(0, 1/2]$), ln size_A is the natural log of the size of the action space A, and T is the number of time steps.

The exploration parameter η controls how quickly actions are penalized in MWU's update rule $w^{t+1}(a) = w^t(a) \cdot (1 - \eta \cdot c^t(a))$, and can therefore be tuned to balance exploration of the state space versus exploitation of cost information learned from the environment. Since our MWU interpreter performs exact rational arithmetic (in fact, dyadic rational arithmetic for performance), we require that η be representable as a dyadic rational number.

As a second consequence of our use of exact arithmetic, our verified MWU uses the linear scaling term $1 - \eta \cdot c(a)$ to update the weight of action a in each round, where $c(a)$ denotes the cost of a in the previous round. A variant of the algorithm uses the exponential scaling term $exp(-\eta \cdot c(a))$, which we cannot represent exactly in our executable version of MWU. One could use the Taylor-series approximation of e^x to approximate $exp(-\eta \cdot c(a))$ at each update step, but only at the expense of additional reasoning about approximation error bounds in the rest of the proof.

3 APPLICATIONS

No-regret algorithms such as MWU can be used to drive multi-agent systems toward the ϵ-CCEs of arbitrary games. Although the CCEs of general games may have high social cost, those of *smooth* games, as identified by Roughgarden [8], have robust Price of Anarchy (POA) bounds that extend even to ϵ-CCEs.

Our generic algorithm for proving bounds on the social cost of games like multi-agent affine routing has the following steps:

(1) Prove that the game is (λ, μ)-smooth, for smoothness parameters λ and μ. The smoothness parameters are specific to the game. In affine routing, $\lambda = \frac{5}{3}$ while $\mu = \frac{1}{3}$.
(2) Use smoothness from (1) to prove robust POA bounds for the game. For example, the POA of an ϵ-CCE of a (λ, μ)-smooth game is $\frac{(1+\epsilon)\lambda}{1-(1+\epsilon)\mu}$. The POA of affine routing wrt. ϵ-CCEs is therefore approximately 5/2.
(3) Prove that N agents each running verified MWU together drive the system to an ϵ-CCE. This proof follows directly from the per-agent regret bound that results from each agent running our verified MWU.
(4) Compose the results in (2) and (3) to prove overall bounds on the social cost of the resulting state.

We illustrate with an application – distributed routing – in which we let multiple agents independently run our verified MWU to

drive each other toward an ϵ-CCE.[1] In a simple version of the distributed routing game with affine latency functions (Figure 2), N routing agents each choose a path from a global source vertex s to a global sink vertex t (a generalization of this game allows s and t to differ across players). Latency over edge e, modeled by an affine cost function $f_e(x) = a_e x + b_e$, scales in the amount of traffic x over that edge. An optimal solution minimizes the total cost to all agents. Roughgarden [8] showed that such games are $(\frac{5}{3}, \frac{1}{3})$-smooth, implying a robust POA bound of 5/2; thus even ϵ-CCEs as produced by MWU are bounded with respect to the socially optimal solutions of the game.

In Coq, we represent games as pairs (A,C) of

- a finite type A, the strategy or action space; and
- a cost function $C(i,s) : \mathbb{Q}$, which for a given player $i \in [0,N)$ returns the cost to that player of state s, a strategy profile of type $[0,N) \to A$ mapping players to strategies.

As an example game, consider a simplified version of resource (congestion) games [3] in which agents may choose to use (or not use) just a single resource.[2] The cost to a player of this game is the total number of players using the resource, assuming the player chooses to use the resource, and 0 otherwise.

In Coq, we represent this game's strategy space A as the inductive data type Resource $\triangleq$ RYes | RNo in which the constructor RYes indicates that the agent chose to use the resource and RNo otherwise.[3] The cost function for Resource is:

$C_{\text{Resource}}(i,s)$ = **if** s_i **is** RYes **then** traffic s **else** 0

where traffic s equals the total number of agents who used the resource in state s.

From basic games such as (Resource, C_{Resource}) we build more complicated ones using a language of game combinators (functions mapping games to games). For example, the product combinator over games – which can be thought of informally as running two games in parallel – takes as input games (A,C_A) and (B,C_B) and produces as output the new game $(A \times B, C_{A\times B})$ over states mapping players to ordered pairs (a,b). The cost function $C_{A\times B}$ of the resulting game sums the costs with respect to C_A and C_B. To prove smoothness of both basic games such as (Resource, C_{Resource}) and of derived games, we developed a library of smoothness preservation proofs following the structure of our language of game combinators. In the product game, for example, if (A,C_A) is (λ_A,μ_A)-smooth and (B,C_B) is (λ_B,μ_B)-smooth, then $(A \times B, C_{A\times B})$ is $(\max(\lambda_A,\lambda_B), \max(\mu_A,\mu_B))$-smooth. We prove smoothness of basic games such as (Resource, C_{Resource}) by mechanizing the standard paper-and-pencil proofs (e.g., [8, 2.3.1] for congestion games).

In our implementation of the affine routing games of Figure 2, we model each edge e in the network as a Resource under application of a second combinator, Affine$(a_e, b_e, C_{\text{Resource}})$, which maps the C_{Resource} cost function to the more general

$C_e(i,s)$ = **if** s_i **is** RYes **then** a_e*(traffic s) + b_e **else** 0.

The resulting game over edge e has cost 0 if an agent does not use edge e in state s, and cost a_e*(traffic s) + b_e otherwise.

We generalize from games over a single edge to those over all source-sink paths in a network in two steps:

First, we construct an m-edge game, in which the agents are free to use any subset of the Resources modeling the graph's edges, by building the product of m affine resources:

$$T \triangleq \begin{cases} & \text{Affine}(a_{e_1}, b_{e_1}, C_{\text{Resource}}) \\ \times & \text{Affine}(a_{e_2}, b_{e_2}, C_{\text{Resource}}) \\ & \vdots \\ \times & \text{Affine}(a_{e_m}, b_{e_m}, C_{\text{Resource}}) \end{cases}$$

The associated cost function of the game over type T is the sum of the individual Affine$(a, b, C_{\text{Resource}})$ cost functions.

Second, to transform the unrestricted game over type T – in which players are free to choose any subset of the edges – to one in which the players may choose only valid paths, we apply a final combinator to T that limits the strategy space to those actions satisfying a predicate, isValidPath(G,s,t), specifying the set of valid paths from source s to sink t under a particular topology G.

All the combinators we apply in the affine routing game preserve smoothness. To prove that the game thus implemented by N agents each running MWU converges to an optimal routing configuration, it's therefore sufficient to compose our MWU proof with the robust POA guarantee we have proved generically for all smooth games.

Conclusion. This brief announcement reports on the first formally verified implementation of Multiplicative Weights Update, a simple yet powerful algorithm for approximately solving Coarse Correlated Equilibria (CCE), among many other applications. We prove our MWU implementation correct via a series of program refinements with respect to a high-level implementation of the algorithm. As part of the larger Cage project, we use our certified MWU to prototype a new architecture for verified distributed systems in which MWU generically drives game-based distributed systems with robust Price of Anarchy bounds to socially optimal CCEs.

ACKNOWLEDGMENTS

We thank the PODC reviewers for their comments on an earlier draft. The research was supported in part by NSF award #1657358.

REFERENCES

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. 2012. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing* 8, 1 (2012), 121–164.
[2] Yves Bertot and Pierre Castéran. 2013. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions.* Springer Science & Business Media.
[3] George Christodoulou and Elias Koutsoupias. 2005. The price of anarchy of finite congestion games. In *Proceedings of the 37th annual ACM Symposium on Theory of Computing.* ACM, 67–73.
[4] Yoav Freund and Robert E Schapire. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory.* Springer, 23–37.
[5] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. 2015. *A small scale reflection extension for the Coq system.* Technical Report. INRIA.
[6] Nick Littlestone and Manfred K Warmuth. 1989. The weighted majority algorithm. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science.* IEEE, 256–261.
[7] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic Game Theory.* Vol. 1. Cambridge University Press.
[8] Tim Roughgarden. 2009. Intrinsic robustness of the price of anarchy. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing.* ACM, 513–522.

[1] The project website includes a second application, to distributed load balancing.
[2] We generalize below to multiple-resource games with arbitrary affine cost functions.
[3] We define a new inductive type RYes | RNo as opposed to letting Resource equal the isomorphic bool for technical reasons related to Coq's typeclass resolution.