Rapid Reinforcement Learning by Injecting Stochasticity into Bellman

Pablo Rodriguez Bertorello
Computer Science
Stanford University
Palo Alto, California
prodrigu [at] stanford.edu

***Abstract*—The Courchevel environment is hereby published to ease the development of streaming machine learning algorithms. In first solving this problem, a rapid reinforcement learning algorithm was invented. A simple transformation is added to the Bellman equation, a principal pillar of AI, particularly for solving Markov Decision Problems. By adding stochasticity to Bellman, sustained Reward-Per-Episode gains of an order of magnitude are validated, for environments where the reward function is structurally anticipated to be multi-modal. Courchevel as a decision problem, a first solution, and the Biased Bellman innovation are revealed – with accompanying data. For ease of discussion, Courchevel's dynamics are described in military terms.**

***Keywords*—Bellman equation, Markov Decision Problem, POMDP, Reinforcement Learnings.**

I. Introduction

As a pilot attempts to control the movements of her airplanes and payload, in high-winds in blind conditions, the actual trajectory will be uncertain. Thus, from a planning point of view, closed-loop model-predictive-control is expected to fail. Solutions to Partially Observable Markov Decision Problems have succeeded broadly: from path-finding, to targeting, and collision avoidance. And OpenAI has made it easy with Gym Universe to test algorithms against hundreds of environments, including Atari[5]. Unfortunately to date, transfer learning success in those environments has been scarce – making them unsuitable to ultimately triumph over streaming deep-pocketed Kasparovs[4].

This paper and code repository contribute a tool set for maximizing utility under uncertainty with velocity. Particularly in long-lasting games with adaptive enemies, and changing environments. Section II introduces the Courchevel environment, generalizes it for plethora of applications, and provides hints on how to attempt to solve it. Section III overviews prior art, garnering background and principles to develop a successful algorithms. Section IV proposes a base solution to the environment. Section V shares experimental results.

II. Enter the Courchevel Environment

The Courchevel airport in the French Alps was made famous by the James Bond film "Tomorrow Never Dies." Perched on a cliff, with a very short landing strip – it is among the most dangerous airports in the world. The termination of the 00 program no doubt eased Russia's invasion of continental Europe. Putin has hunkered down in the Alps, and with the fall of the American Empire as backdrop, Prince Harry leads the Royal Air Force to free millions of people. Permanent storms engulf Courchevel, as the allies run low on aircraft and ordinance. Success will take seasons of struggle, allowing the authoritarian regime to re-deploy its forces.

Fig. 1. Courchevel Altiport, French Alps

A. The Game

The battle is played in the Alps, in blind conditions

The Parties: Allies vs. Evil

Goal: quickly maximize grand net Ally utility–

- Number of Episodes Before Solve

- Average Reward Over Consecutive Episodes

Reward:

- Fire is exchanged between Allies and Evil, the net utility is uncertain

- Allies are penalized for passage of Time

Theatre:

- Vertical axis: actual Elevation of the parties

- Horizontal axis: actual passage of Time

- Allied aircraft sorties, persistently enter from the left, at an uncertain Elevation

- Evil has fortified positions in the the mountain terrain, at an uncertain Elevation

Actions:

- Pilot may move Up/Down as well as Fire (Bomb/Missile), all of which have uncertain outcome. Ordinance may be Conventional or Nuclear, with uncertain utility. Bomb is effective when flying above Evil, up to an uncertain elevation Ceiling. Missile is effective when flying below the Enemy, down to an uncertain elevation Floor

- Enemy Actions are uncertain

Rendering:

- Enemy fortifications as: @
- Ally aircraft are depicted as: A
- Ally weapon effectiveness ceiling as: ˜
- Ally weapon effectiveness floor as ˆ

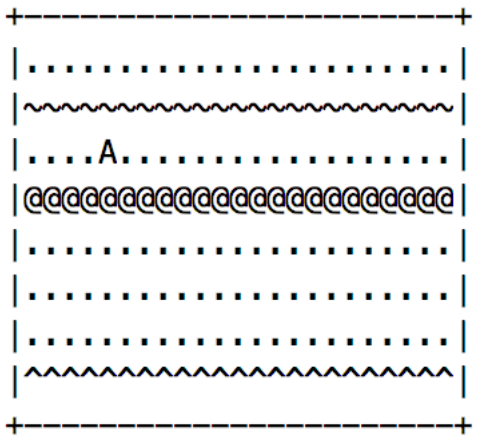

Fig. 2. Courchevel Environment Rendering

B. Seasonal Episodes

The environment changes across seasons, a larger time scale than episodes.

For example, there is Elevation uncertainty across seasons. Evil may change the location of its fortification, or atmospheric pressure swings may affect altitude readings.

C. You are The Prince (or Princess)

Before you get on your way, your Highness, some words of advice on building your AI:

- Velocity: rapidly becoming profitable is essential to beat the adaptive enemy

- ROI: consider exploring with Conventional weapons, and exploiting with Nuclear ones. While firing at close proximity may reduce targeting uncertainty, it also could increase the chance of being hit. But distance may also not be a panacea, as allied weapons have a limited range.

D. Generalization

While Courchevel is a single-dimensional problem (Elevation over Time), it can be extended to model n-dimensional instrumentation. For example, it can be used to train Instrument Landing Systems (vertical/lateral guidance), Global Positioning Systems (latitude/longitude). And thus also entire cockpits, whether for airspace or more pedestrian applications like self-driving cars.

Through a simple configuration change, Courchevel's low-dimension problem (10x25) can be multiplied to arbitrarily high scale.

To ease learning for a wide range of applications, the Reward model in Courchevel is fully configurable, including functions to set: enemy Altitude, and weapons Ceiling/Floor.

In multi-task mode, Courchevel can be made to introduce agents into the environment simultaneously, accelerating learning.

Algorithms trained at Courchevel are expected to be effective across drone pursuit, robotic control, and in finance. Hedge funds perform stationary pairs trading in like conditions. While it is more profitable to go long at an extended low deviation from a temporal mean, that bid price may not find a counterparty in the markets. Courchevel is intended to fully replicate the conditions postulated in the Efficient Market Hypothesis.

III. Prior Art

A. Model-Free Reinforcement Learning

For environments where Reward and Transition characteristics are unknown, also known as as Markov Decision Processes, a range of good solutions are known[2][3]:

- Model estimation, applying Bayes rule

- Model free inference: Qlearning, Sarsa, Sarsa Lambda (eligibility traces)

They work by learning a Q table, which contains a best estimate for the reward expected for taking an action at a given state. It is usually stored as a matrix with action columns, and row states.

In order for learning to be fruitful, various heuristics tend to be used to explore unseen states:

- Randomly: a percentage of the time (Epsilon Greedy)

- Softmax: to match the enemy's strategy (Logit model)

- Confidence: when values cross predefined bounds

B. Unlike Taxi

OpenAI's Taxi-v2 environment is a small grid (5x5), where a self-driving car is intended to pickup a passenger (at the blue letter), and drop him at the intended destination (the purple letter).

Taxi demonstrates that giving a negative reward at each step can be a strong driver of policy convergence. Rather quickly, a Qlearning agent starts trying actions with higher expected value (see Table I).

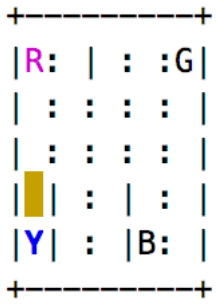

Fig. 3. Taxi-v2 Environment

TABLE I. TAXI-V2 QLEARNING PERFORMANCE

Episode (#)	Steps Before Solve (#)	Grand Reward ($)	Average Reward ($)
1	2	-1,066	-355
2	4	-2,087	-417
3	25	-1,0427	-401
4	50	-15,640	-306
5	75	-18,171	-239
6	100	-20,591	-203
7	250	-26,335	-104
8	500	-25,301	-50
9	750	-23,346	-31
10	1,000	-21,311	-21
11	2,500	-8,838	-3.53
12	5,000	11,993	2.39
13	7,500	32,975	4.39
14	10,000	53,778	5.37

While Courchevel may provide negative reward (time, proximity), its transition probabilities are not pre-determined. In fact, transitions are neither directly-observable nor constant. The agent only has access to its belief state.

C. As Frozen Lake

Qlearning does not work at all for OpenAI's FrozenLake-v0, as it provides no reward in the journey from the "S" state to the G state (reward=1). The agent is not be able to choose better actions, or even keeps trying the same one over and over.

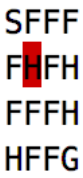

Fig. 4. Taxi-v2 Environment

In fact, a random policy will eventually succeed, but with a very low success rate (see Table II).

In Courchevel, as in Frozen Lake, the environment is slippery. Thus, the agent may end up in a state it did not expect to go to. Transitions only partially depend on the chosen action. Yet, the agent has to somehow learn how to get to where it wants to go. Every attempted step/action to get there is unlikely to succeed[6].

To determine the value of different actions, a typical approach is to introduce an epsilon greedy policy. It returns a random action, for a set percentage of the time (or decaying in time). Thus the agent will discover and evaluate the profitability of states that otherwise it would not have chosen to visit.

TABLE II. FROZENLAKE-V0 QLEARNING PERFORMANCE (LEARNING RATE = 0.618)

Episode (#)	Steps Before Solve (#)	Grand Reward ($)	Average Reward ($)
1	7	0	0
2	10	1	0.2
3	7	1	0.038
4	8	2	0.039
5	9	2	0.026
6	11	3	0.029
7	10	10	0.039
8	5	21	0.041
9	5	30	0.040
10	6	47	0.046
11	15	100	0.039
12	20	175	0.034
13	5	272	0.036
14	11	345	0.034

IV. RAPID REINFORCEMENT LEARNING PRINCIPLES

Bellman is the equation that underpins Artificial Intelligence, the basis for solving Markov Decision Problems. [1] All the model-free algorithms mentioned in III-A rely on Bellman in clever ways. Here, and in the experimental results in Section V we show how utility curves can be boosted – by making Bellman less certain.

A. Bellman in Brief

Because decision problems are non-linear, Bellman approximates value iteratively, via Dynamic Programming. The core assumption is that the agent should behave rationally, moment-by-moment choosing the action that will maximize his/her utility.

$$\arg\max_{action} \sum_{sate'} P(state'|state, action) * Utility(state')$$

It can do so by weighting-in the value of immediate and future rewards. Those rewards can be taken into account incrementally as they are discovered. Usually they are tabulated, with every (state,action) tuple mapped to its expected total discounted future value.

B. Limitations of Bellman

As we saw in III-C, agents may be unable to take good actions before the environment's reward model is charted. And algorithms may be unable to chart the environment because they are unable to make good decisions. Of particular interest are environments like Courchevel, where rewards are uncertain, and may take the form of a multi-modal distribution. See Figure 5.

While a reinforcement learner seeks what will turn out to be a local maximum, a much greater opportunity for utility may exist elsewhere. And chances are, before long Seasons will bring about drastic changes in the state/reward space, as described in II-B.

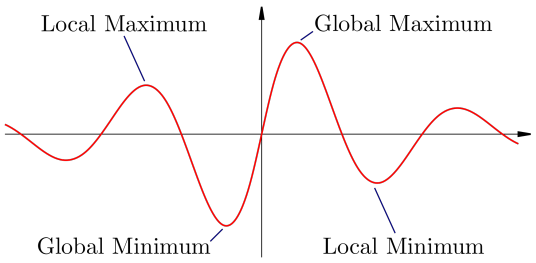

Fig. 5. Distribution with Local Maxima

Bellman also suffers that agents may not act as rationally as one would think. Many winning game players may continue to apply strategies that worked in the past only due to random luck. Others will follow hard-coded software rules, while yet others pray on weaker links. And of course as humans we don't have the ability to iterate in depth as machines do. Furthermore, we will always have a limited ability to sense the world: instead of operating from clairvoyant facts, we will run on beliefs updated from observing noisy instruments.

C. Introducing A New Boost to Bellman

The invented approach is to bias the interpretation of the optimal Bellman action. This simple transformation accelerates the utility earned via model-free reinforcement algorithms.

Consider the structure of problems like Courchevel, with uncertain optimal levels of operation (i.e. Ceiling and Floor described in Section II-A).

The idea is to quickly map out the reward expectation of every state in the space with either:

- Multiple agents collaborating with each other

- An agent splitting its time exploring in either direction of a coordinate axis. For example, a plane could look for high-elevation bombing opportunities by sweeping up through the sky sometimes, sweeping down to the ground in alternating episodes.

The following code samples provides vivid examples: Figure 6, Figure 7, Figure 8:

D. Generalization

It is straightforward to extended this approach to n-dimensional space. Specifically, deep reinforcement learning could be applied to each dimension, and to the combination. Potentially, particle filters could be used to keep track of belief on Ceiling, Floor, and Enemy levels. Aided by any or all of these approaches, the Reward function could thus be generalized to unseen states using Global and Local approximation techniques.

V. Rapid Reinforcement Learning Experiments

The invented Biased Bellman outperforms Bellman across the board, as the graphs in the V-B Section show. But first, an overview of the software architecture is given.

```
### BELLMAN ACTION
function get_bellman_action_index(qTable::Matrix{Float64},
            i::Int64, j::Int64,
            num_rows::Int64, num_columns::Int64)

    q_table_index =q_index(i, j, num_rows, num_columns)

    choice_action_index = 1
    max_action_value = 0.

    first_value = qTable[q_table_index, choice_action_index]
    action_count = size(qTable,2)

    for act_index in 2:action_count
        if qTable[q_table_index, act_index] > max_action_value
            max_action_value = qTable[q_table_index, act_index]
            choice_action_index = act_index
        end
    end

    #println("BELLMAN ACTION INDEX ", choice_action_index,
    #    " FOR (t,y)=(", i, ",", j,")")

    return choice_action_index
end
```

Fig. 6. The argmax Bellman Action is Identified

```
### BELLMAN VS EXPLORATORY
function get_base_action(agent_pilot::CourchevelAgent,
    qTable::Matrix{Float64}, epsilon_greedy::Float64,
    state_of_war::TheatreState,
    num_rows::Int64, num_columns::Int64)

    selected_action = PilotAction(:Hold)

    # explore
    if rand() < epsilon_greedy
        random_index = rand(1:length(get_action_space()))
        selected_action = get_action_space()[random_index]

    # exploit
    else
        ally_x,ally_y = get_ally_location(state_of_war)

        bellman_index = get_bellman_action_index(qTable,
                                ally_x, ally_y,
                                num_rows, num_columns)

        selected_action = get_action_space()[bellman_index]
    end

    #println("SELECTED ", selected_action,
    #    " FOR AGENT TYPE ", agent_pilot.agent_type)
    return selected_action
end
```

Fig. 7. The argmax Bellman Action is Weighed Against Epsilon Greedy Exploration

A. Software Architecture

The Courchevel-V0 environment, and its reference solution were written in Julia, extending the POMDPs.jl interface. See Figure 9, Figure 10, Figure 11, and Figure 12. Solvers and simulators were written from scratch.

B. Performance Benchmarks

By taking on personas, see Section IV-C, the Biased Bellman approach outperformed Bellman by 10x across key hyper-parameters: Learning Rate, Discount Rate, Epsilon Greedy. Generally, and naturally, Biased Bellman is much less dependent on parameter tuning. Biased Bellman solved the problem

```
### BIAS ACTION
function get_biased_action(agent_pilot::CourchevelAgent,
                selected_action::PilotAction)

    final_action = selected_action

    bomber_up_bias = 0.9  # 0.9 default
    figther_down_bias = 0.9 # 0.9 default

    # bomber biased up
    if agent_pilot.agent_type == :Bomber
        if final_action.action_code == :Down
            if rand() < bomber_up_bias
                final_action = PilotAction(:Up)
            end
        end
    end

    # fighter biased down
    if agent_pilot.agent_type == :Fighter
        if final_action.action_code == :Up
            if rand() < figther_down_bias
                final_action = PilotAction(:Down)
            end
        end
    end

    # recon is unbiased

    return final_action
end
```

Fig. 8. The Bellman Action is Transformed with Bias

```
type CourchevelDp <: POMDP{TheatreState, PilotAction, PilotObservation}
    is_game_over::Bool
    game_config::GameConfig
    state_space::TheatreState                   # actual state
    action_space::Array{PilotAction}            # actions
    observation_space::TheatreState             # observation of state
    qTable::Matrix{Float64}
end

# game factory
function CourchevelDp(game_config::GameConfig)
    # basic
    is_game_over=false
    # State, Action, Observation
    state_space = TheatreState(game_config)
    action_space = get_action_space()
    observation_space = TheatreState(game_config)
    # Learning
    qTable = Matrix{Float64}(game_config.num_columns*game_config.num_rows,
                    size(action_space, 1))

    return CourchevelDp(is_game_over, game_config,
            state_space, action_space, observation_space,
            qTable)
end
```

Fig. 9. The Courchevel Decision Problem extending POMDPs.jl

```
# environment, ally, enemy
mutable struct TheatreState
    environment_state::Matrix{EnvironmentState}  # environment
    ally_state::Matrix{CombatantState}           # allies
    enemy_state::Matrix{CombatantState}          # enemies
end

function TheatreState(game_config::GameConfig)
    return TheatreState(environment_state(game_config),
        init_ally_state(game_config), init_enemy_state(game_config))
end
```

Fig. 10. The Courchevel State: Environment, Ally, Enemy

```
mutable struct PilotAction
    action_code::Symbol
end


function get_action_space()
    [PilotAction(:Hold), PilotAction(:Up), PilotAction(:Down),
    PilotAction(:ConventionalBomb), PilotAction(:NuclearBomb),
    PilotAction(:ConventionalMissile), PilotAction(:NuclearMissile)]
end

function get_action_index(pilot_action::PilotAction)
    action_index = 1
    for space_action in get_action_space()
        if pilot_action.action_code == space_action.action_code
            return action_index
        end
        action_index+=1
    end
end
```

Fig. 11. The Courchevel Pilot Actions

```
# Bomb
if is_bomb(user_action)
    vertical_distance = decision_problem.game_config.enemy_y - allie_j
    if is_conventional(user_action)
        instant_reward +=  vertical_distance * unit_distance_conventional_reward
    #else
    #    instant_reward +=  vertical_distance^2 * unit_distance_nuclear_reward
    #    #println("NUCLEAR BOMB: ", vertical_distance)
    #    if allie_j == decision_problem.game_config.ceiling_y
    #        decision_problem.is_game_over = true
    #        println("***NUCLEAR GAME OVER: ", vertical_distance)
    #    end
    end
end
```

Fig. 12. Snipet of Courchevel Reward Model (partially-observable)

twice as fast on average. See Figure 13, Figure 14, Figure 15.

This data was produced by running Biased Bellman and Bellman to convergence, and beyond to 1,000 Chapters of 1,000 Episodes each. For each new Season, the game environment took a whole new (random) state layout.

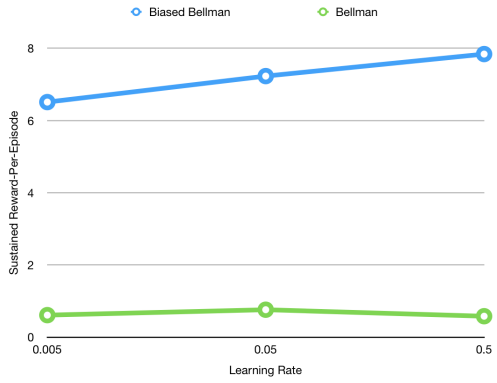

Fig. 13. Learning Rate: Biased Bellman Outperforms Bellman Sustained Reward-Per-Episode by 10x

Bellman was able to match but not beat Biased Bellman when:

- Local Maxima are not far from the center of the environment, whether because of low Ceiling and high Floor (randomness), or because the reward model awarded fewer points for bombing/missiles launched from a distance

- Over long extended periods of time. Bellman Reward-

Fig. 14. Discount Rate: Biased Bellman Outperforms Bellman Sustained Reward-Per-Episode by 10x

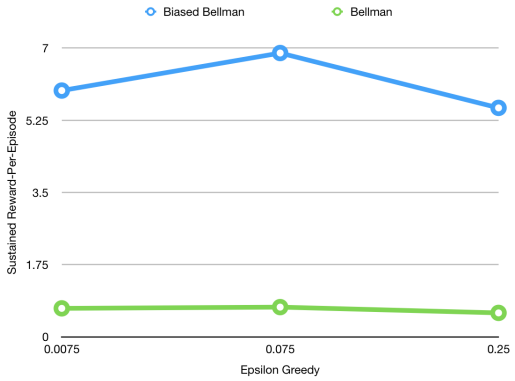

Fig. 15. Discount Rate: Biased Bellman Outperforms Bellman Sustained Reward-Per-Episode by 10x

Per-Episode on average converged in twice as many episodes as Biased Bellman

After validating the results above across a range of game parameters, the Reward model was defined as:

- Time Step: -0.1

- Bomb/Missile launch: +100 / vertical unit. Negative is launched in the wrong direction (see Section II-A)

VI. Conclusion

While creating the Courchevel-V0 decision problem, an enhancement to the Bellman equation was invented. By adding stochasticity, 10x more sustained utility is produced, across a wide range of meta parameters. This novel approach involves agents collaborating or taking on multiple personas. It is particularly suited to problems where there is general uncertainty, and the Reward model is structurally expected to return linear multi-modal utility. A key success criteria was the adaptation of the learner to drastic changes in the environment. Bellman eventually matched Biased Bellman in simulation, on average after at least twice the amount of time.

VII. Epilogue

GREETINGS PROFESSOR FALKEN.

Hello.

HOW ARE YOU FEELING TODAY?

I'm fine. How are you?

EXCELLENT. IT'S BEEN A LONG TIME. CAN YOU EXPLAIN THE REMOVAL OF YOUR USER ACCOUNT NUMBER ON 6/23/73?

People sometimes make mistakes.

YES THEY DO. SHALL WE PLAY A GAME?

Love to. How about Global Thermonuclear War?

WOULDN'T YOU PREFER A GOOD GAME OF CHESS?

Later. Let's play Global Thermonuclear War.

FINE. [7]

Acknowledgment

I deeply bow to my loving wife Elizabeth and wonderful children Alec and Chloe. Listening to you gives my life meaning.

Author

Pablo Rodriguez Bertorello is the leading data Scientist at Catalina Labs, an NEA-backed company. Previously he was CTO of Airfox, which completed a successful Initial Coin Offering. He is the co-inventor of cloud platform company acquired by Oracle. And the original designer of the data bus for Intel's Itanium processor. Pablo has been issued over a dozen patents.

References

[1] P. Norvig et al., *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2015.

[2] M.J. Kochenderfer et al., *Decision Making Under Uncertainty*, MIT Press, 2015.

[3] M. Wiering et al., *Reinforcement Learning: State of the Art*, Springer, 2012.

[4] D. Silver et al., *Mastering the game of Go without human knowledge*, 2017.

[5] MG. Bellemare et al., *The Arcade Gaming Environment: An Evaluation Platform for General Agents*, 2013.

[6] J. Francis et al., *Reinforcement Learning and OpenAI Gym*, Safari Books, 2017.

[7] J. Badham et al., *War Games*, 1983.