Counterexamples to RL approaches blending search and learning for problem-solving

Brieuc Pinon[0000−0001−5727−932X], Jean-Charles Delvenne[0000−0003−2356−7790], and Raphaël Jungers[0000−0002−7789−0940]

Department of Mathematical Engineering, UCLouvain
Louvain-la-Neuve - Belgium
{brieuc.pinon,jean-charles.delvenne,raphael.jungers}@uclouvain.be

Abstract. How can learning be leveraged to accelerate problem-solving, in Automated Theorem Proving for example? While various methods have been proposed in the literature, this paper examines the limitations of three prominent Reinforcement Learning approaches: the Bellman equation for learning value functions, Hindsight Experience Replay for learning universal value functions, and a divide-and-conquer strategy applied to tactic-based theorem proving. We construct counterexamples of seemingly simple problems that are provably intractable for these methods. Our findings reveal that these algorithms can fail to effectively utilize critical information about the problems, leading to significant inefficiencies.

Keywords: Reinforcement Learning Theory · Reasoning · Problem-Solving.

1 Introduction

One of the main goals of Artificial Intelligence is problem-solving: given a problem, the objective is to find a solution [10]. In the context of Automated Theorem Proving (ATP), for example, this involves finding a proof of a given theorem from a set of axioms within the constraints of a logic system. Since this must be done under constrained computational resources, algorithms are designed to find a solution quickly.

A promising idea to design these efficient algorithms is to blend search and learning, where learning is leveraged to accelerate the search. These algorithms repeatedly attempt to construct solutions, leveraging feedback from previous attempts to learn to guide the next constructions.

This paper examines the theoretical limitations of current Reinforcement Learning approaches blending search and learning to enhance our understanding and support the development of more effective algorithms. We focus on three specific approaches:

– **The Bellman equation** This Dynamic Programming approach optimizes objectives over sequences of decisions by associating a state with each decision-making point. It can be combined with Machine Learning to learn a *value*

function that predicts the value of the objective from a given state. The Bellman equation ensures the consistency of these estimates: a state is promising only if it leads to other promising states. Usually, algorithms using this approach iteratively follow decisions that maximize estimated values and enforce the Bellman equation across visited states. This method forms the foundation of many Reinforcement Learning (RL) algorithms, with various adaptations [15].

- **Hindsight Experience Replay (HER) [1]** HER builds upon the Bellman equation to learn *universal value functions* [16,11]. Assuming that the objective is to reach some goal-state, a standard value function only estimates the value of a state in terms of its ability to reach this goal. In contrast, a universal value function predicts the reachability of any state from any other state. HER utilizes the states encountered during the learning process to learn this universal value function, leveraging richer feedback than a simple binary outcome of whether the goal was reached or not.
- **Divide-and-conquer** We refer to several algorithms in the Automated Theorem Proving (ATP) literature built upon tactic-based theorem proving such as [9,5]. In tactic-based theorem proving, a theorem is broken down into subgoals according to a chosen tactic, these subgoals are then further decomposed by tactics until elementary true propositions are reached as in the Lean theorem prover [7]. The algorithms here learn a value function that estimates the provability of each subgoal. This value function is also learned using Dynamic Programming. This approach allows to leverage feedback from the constructed subgoals rather than only focusing on the main theorem.

We demonstrate that despite their strengths, implementations of these approaches can struggle with certain "easy" problems that are nevertheless provably hard to solve for them. Our study examines algorithms that apply the core idea of each approach, incorporating shortcuts for simpler theoretical analysis, such as using some optimal values directly in the feedback.

Each counterexample problem targets a different algorithm but is based on a common principle: combining a set of easy problems into one aggregated problem that should be nearly as easy to solve than the original problems. However, by carefully designing the aggregation, we can obscure the information from each individual problem, forcing the algorithm to tackle all original problems simultaneously rather than breaking them down into independent, simpler tasks.

This type of theoretical analysis was pioneered by [14], who constructed a family of problems that model-free RL methods struggle to solve efficiently compared to a model-based method with access to a predetermined goal. Subsequent research expanded these findings by demonstrating a similar limitation broadened to some model-based methods, moreover, it did not require a priori knowledge of a goal [8]. Both of these works highlighted limitations in Bellman equation-based methods by embedding critical information within unknown dynamics.

Our contributions are:

- Demonstrating a limitation for an algorithm based on the Bellman equation across a family of problems with a known common dynamics across the

problems. This extends the scenarios in which an inefficiency for a Bellman equation-based method can be expected, such as in ATP.
– Identifying limitations in two additional approaches: Hindsight Experience Replay and a divide-and-conquer strategy for tactic-based ATP.

Intuitively, the results of this paper formalize the incapacity of these algorithms to efficiently "learn from failure". A common thread in our counterexamples is that failed attempts provide minimal useful information to learn to guide the construction of a solution.

2 Preliminaries

We note $[n] = \{1, \ldots, n\}$ the set of the n first natural numbers. For a vector $\mathbf{x} \in S^n$, with some set S and $n \in \mathbb{N}$, we note $x_{\leq i}(/x_{<i})$ the vector restricted to the first $i(/i-1)$th coordinates. An index list I over $n \in \mathbb{N}$ is a sequence of numbers in $[n]$. For I an index list over n, x_I is the vector composed of the values of x at the coordinates in I. For x and y two vectors, we note $[x, y]$ their concatenation.

We use the name *variable* for the value at some coordinate in a vector.

3 The Bellman equation

We identify a limitation on a Bellman equation-based method for the family of SAT instances. We define Algorithm 1 implementing the Bellman equation to learn a value function and guide the search for solutions to a SAT instance. Then, we state in Theorem 1 that aggregating SAT instances can produce intractable problems for Algorithm 1, this holds even if the original SAT instances are relatively easy to solve individually.

We model the problem of finding a solution for some SAT instance with n binary variables as a sequence of n binary decisions. Specifically: given the SAT instance and the ith first fixed bits, fix the $i+1$th variable to 0 or 1.

The Bellman equation applied on some value function v over this sequence of decisions amounts to:

$$v(x_{\leq i}; p) = \max\{v([x_{\leq i}, 0]; p),\, v([x_{\leq i}, 1]; p)\}, \tag{1}$$

where $x_{\leq i}$ is a partial candidate solution of length i and p is the problem to solve.

Algorithm 1 implements this equation. The algorithm begins with an initial set of hypotheses for the value functions V. It iteratively samples sequences of decisions to construct candidate solutions guided by V. At each state of the construction, the Bellman equation is applied to eliminate inconsistent value functions from V, continuing this process until a solution is found.

The set of hypothesis V acts as a prior over possible value functions, and learning occurs during the search by enforcing the Bellman equation on this set. For example, the set can correspond to a set of decision trees or neural networks.

This prior can encode learned knowledge about solving SAT instances because any value function not in V reduces the set of possible solutions and accelerates the search.

Likewise, the set V can also encode uncertainties. Several value functions in V that provide different estimates represent incertitude and force the algorithm to explore possibilities before discarding them with the Bellman equation.

To facilitate our theoretical analysis, we introduce stronger feedback than the Bellman equation at some states. In the first steps of a construction, the algorithm applies the standard Bellman equation, but after a predetermined index, it replaces the Bellman equation with a condition that enforces any hypothesis function in V to match the optimal values.

We now define problems, solutions, (optimal) value functions, and conditions that are assumed to hold for our result.

Definition 1. *A* problem *p is a binary number in $\{0,1\}^*$ with an associated number $n \in \mathbb{N}$ of variables. The problems are all linked to a function* Check*: $\{0,1\}^n \times \{0,1\}^* \to \{False, True\}$ that correspond to a verifier for CNF-SAT instances [2].*

Definition 2. *A* solution *x to problem p over n variables is a binary vector $\{0,1\}^n$ for which* Check(x, p) *returns True.*

Definition 3. *An* aggregation of problems *is a problem p over n variables defined by K problems $p_1, \ldots, p_K$ and K non-intersecting index lists $I_1, \ldots, I_K$ over n.*

The number of variables n must be the sum of the number of variables of each problem $p_1, \ldots, p_K$. The problem p is the concatenation of all the problems with their variables mapped by the index lists such that Check(x, p) is True iff Check(x_{I_1}, p_1),..., Check(x_{I_K}, p_K) are all True.

Definition 4. *A* value function *is a function taking in input a binary vector in $\{0,1\}^i$ with $i \in [n]$ and a problem over n variables, and outputting in $\{0,1\}$.*

Definition 5. *A value function v is* optimal *for problem p over n variables if for all $x \in \{0,1\}^i$, $v(x; p)$ is 1 iff there exists $y \in \{0,1\}^{n-i}$ such that Check$([x, y]; p)$ returns True.*

Definition 6. *A* policy *is a randomized function that provided a problem, outputs a candidate solution.*

A policy is symmetric *with respect to a set of value functions V if it can be written as the following process: Starting from an empty vector, incrementally sample a binary variable x_i and concatenate it to construct $x_{\leq i}$ until a complete vector $x \in \{0,1\}^n$ is obtained. The binary variables x_i must be sampled according to the sets $V_0 = \{v([x_{<i}, 0], x_{<i}; p) | \, v \in V\}$ and $V_1 = \{v([x_{<i}, 1], x_{<i}; p) | \, v \in V\}$ such that the probabilities of $x_i = 0$ and $x_i = 1$ must be equivariant to those sets, $\Pr(x_i = 0 | \, (V_0, V_1), x_{<i}; p) = \Pr(x_i = 1 | \, (V_1, V_0), x_{<i}; p)$.*

Definition 7. *A value function v is* monotonic *if for any problem p with n variables, any $x \in \{0,1\}^n$, and any $i, j \in [n]$ with $i < j$, v satisfies $v(x_{\leq i}; p) \geq v(x_{\leq j}; p)$.*

Algorithm 1 Search with the Bellman equation.
Bellman_Search is the main function, it iteratively samples candidate solutions guided by a set of value functions V. In parallel, the Bellman equation is iteratively enforced on V along already sampled candidates. The subfunction Bellman_equation enforces these equations. The procedure is parametrized by i^* and v^* that forces the procedure to use the optimal values furnished by v^* from index i^*. The candidates are sampled according to $\pi^{V^t}(p)$ a policy that is symmetric w.r.t. V^t (Definition 6).

Inputs: V^0 : an initial set of value functions; p : the problem to solve; i^*, v^* : parameters to provide the true value function (v^*) at index i^*

function BELLMAN_SEARCH(V^0, p; i^*, v^*)
  for $t \leftarrow 0, 1, \ldots$ **do**
    $x^t \sim \pi^{V^t}(p)$
    if CHECK(x^t, p) **then**
      return x^t
    $V^{t+1} \leftarrow V^t$
    for $i \leftarrow 1, \ldots, n$ **do**
      $V^{t+1} \leftarrow$ BELLMAN_EQUATION(V^{t+1}, x^t, i, p; i^*, v^*)
function BELLMAN_EQUATION(V, x, i, p; i^*, v^*)
  if $i \geq i^*$ **then**
    return $\{v \in V \mid v(x_{\leq i}; p) = v^*(x_{\leq i}; p)\}$
  else
    return $\{v \in V \mid v(x_{\leq i}; p) = \max\{v([x_{\leq i}, 0]; p), v([x_{\leq i}, 1]; p)\}\}$

We have defined everything needed to state our result. Theorem 1 aggregates several problems into one problem and states under assumptions that this aggregated problem is intractable for Algorithm 1. The proof relies on the fact that while the Bellman equation is sufficient to identify an optimal value function and produce a solution, it does so very slowly in this case. In particular, the Bellman equation relies on the outputs of the value function to learn, however, a failure to produce a solution can be attributed to many different decisions, such that the output 0 of the value function (indicating a failure) does not provide much feedback. This poor feedback does not inform the algorithm on why the attempt failed and how to learn to anticipate other failures.

Theorem 1 constructs a problem with this difficulty to attribute a failure by forcing Algorithm 1 to take several hard independent binary decisions from the start, and if one of these decisions is incorrect, the value is zero and the attempt failed.

Theorem 1. *Let p be a problem over $n = K^2$ variables defined by an aggregation of problems $p_1, \ldots, p_K$ over K variables with index lists $I_1, \ldots, I_k, \ldots, I_K$ where the first element of each I_k is k.*

Let V and $V_1, \ldots, V_K$ be sets of value functions.

Assume:

1. *The set V factories into $V_1, \ldots, V_K$, i.e. for all $v \in V$ there exists $v_1 \in V_1, \ldots, v_K \in V_K$ such that for all $x \in \{0,1\}^n$ and all $i \in [n]$, $v(x_{\leq i}; p) = \prod_{k \in [K]} v_k(x_{I_k \cap [i]}; p)$.*
2. *For all $k \in [K]$ and any $v \in V_k$ there exists $v' \in V_k$ such that $v(0;p) = v'(1;p)$ and $v(1;p) = v'(0;p)$.*
3. *For all $k \in [K]$, any $v_k \in V_k$ is monotonic.*
4. *For v^* an optimal value function for problems $p_1, \ldots, p_K$, for all $k \in [K]$ either $v^*(0;p_k) = 1$ or $v^*(1;p_k) = 1$.*

Then Algorithm 1 with arguments $V^0 = V$, $p = p$, $i^* = K$ and v^* an optimal value function for problem p, runs for an expected time of at least $2^{\sqrt{n}-1}$.

Proof. By condition (4) the first variable of all problems $p_1, \ldots, p_K$ must be either 0 or 1 to be a solution. Thus, any solution to problem p must have its first $K = \sqrt{n}$ variables equal to some unique binary vector $x^*_{\leq K} \in \{0,1\}^K$.

We prove that Algorithm 1 treats symmetrically the sampling of these first $K = i^* = \sqrt{n}$ binary variables until $x^*_{\leq K}$ is sampled for these variables. More formally, let $(V^0, x^0_{\leq K}), (V^1, x^1_{\leq K}), \ldots$ be the result of the stochastic process induced by Algorithm 1, then for any index $i \in [K]$ we can define $(V^{0\prime}, x^0_{\leq K}{}'), (V^{1\prime}, x^1_{\leq K}{}'), \ldots$ where x' is x with the ith bit flipped and $V^{t\prime}$ is constructed from V^t. For all $v \in V^t$ we construct a corresponding $v' \in V^{t\prime}$ such that $v(x;p) = v'(x';p)$ for all $x \in \{0,1\}^{[K]}$. We will prove that the two sequences have the same probability when truncated to the first appearance of $x^*_{\leq K}$ in the original sequence.

Since from this symmetry the expected first time of appearance of any $x_{\leq K} \in \{0,1\}^K$ is the same, we can derive the following lower bound on the expected running time: 2^{K-1}.

We now prove by induction that Algorithm 1 treats symmetrically (in the sense described above) V^t and the sampling of $x^t_{\leq K}$ until $x^*_{\leq K}$ is sampled.

By assumption (1) and (2) the image of V^0 on p is invariant for any binary vector of length less than K. Thus the distribution $\pi^{V^0}(p)$ is also invariant over $\{0,1\}^K$ by symmetry of π (Definition 6).

The set V^t evolves with three categories of operations: calling the Bellman equation procedure with (a) $i < i^*$; (b) $i = i^*$; (c) $i > i^*$.

For (a), by the inductive assumption the set V^t is symmetric and the policy to sample $x^t_{<K}$ is also symmetric. The Bellman equation is solely a symmetric function of $x^t_{<K}$ and V^t, so the complete process to compute V^{t+1} is symmetric.

For (b), by the inductive assumption the distribution of $x^t_{\leq K}$ is symmetric and $x^*_{\leq K}$ has not been sampled thus $v^*(x^t_{\leq K}; p)$ is always 0. The Bellman equation procedure process is symmetric with respect to $x^t_{\leq K}$ and V^t to produce V^{t+1}.

For (c), the Bellman equation procedure applied with $i > i^*$ does not affect the new set of value function V^{t+1}. By contradiction, assume $v \in V^t$ is removed by the Bellman procedure with $i > i^*$ and some x^t. Also by assumption we know $x^t_{\leq K} \neq x^*_{\leq K}$ and thus $v^*(x^t_{\leq K}; p) = 0$ and $v^*(x^t_{\leq i}; p) = 0$ since any correct value function is monotonic. By (1) we know that v can be factorized into

$v_1 \in V_1, \ldots, v_K \in V_K$, since this last equation removes v, we know $v_k(x^t_{\leq i}; p) = 1$ for all $k \in [K]$. Assumption (3) implies $v_k(x^t_{\leq K}; p) = 1$ for all $k \in [K]$, and thus $v(x^t_{\leq K}; p) = 1$. However, we know that at loop t the Bellman procedure has been applied earlier with the equation $v(x^t_{\leq K}; p) = 0$. Consequently, the value function v has been removed before and cannot be removed by the Bellman procedure with $i > i^*$. □

What does the assumptions of Theorem 1 mean? Assumption (1) requires that the prior over value functions V reflects the structure of the aggregated problem, meaning that the value functions must be factorizable into value functions corresponding to each individual sub-problem. This is a natural assumption since an optimal value function for the aggregated problem must also be factorizable into optimal value functions for the original independent sub-problems.

Assumption (2) implies a symmetry on V with respect to the first bit of each sub-problem. With this condition, the prior V does not allow to decide optimally the first bit of each sub-problem. This can be the case if deciding the first bit of the SAT instance is inherently difficult. Importantly, this assumption does not contradict the possibility that these sub-problems might be relatively easy; but it suggests that the prior induced by V does not enable an immediate solution, necessitating some degree of search.

Assumption (3) stipulates that the factorization of the value functions in V—as required in Assumption (1)—must result in monotonic value functions. This assumption is also natural since any correct value function for any SAT instance must be monotonic.

Assumption (4) posits that the first bit of each sub-problem is critical to solving that sub-problem. SAT instances exhibiting this property can be straightforwardly constructed.

Why is Theorem 1 relevant? In intuitive terms, Theorem 1 suggests that if given problems that are not directly solved by a (learned) prior over value functions, then, aggregating these problems can produce a problem intractable for a procedure relying on the Bellman equation with that prior. Interestingly, this holds even if the individual problems are quickly solvable with minimal search.

In other words, the Bellman equation can fail to leverage the decomposability of an aggregated problem, thereby hindering the learning of an optimal value function that could guide the search. We note that, in contrast, modern (conflict-driven clause learning) SAT solvers straightforwardly leverage the structure of an aggregated problem [4], and thus do not suffer from the identified issue.

Generality of the result. While our analysis focused on a specific algorithm that uses the Bellman equation to learn value functions, we believe our findings reveal a broader limitation on methods that rely on the Bellman equation. These include model-free Deep RL methods like Deep Q-learning [6] and Proximal Policy Optimization [12], as well as some model-based methods like AlphaZero [13], all of which depend on the Bellman equation to learn to guide the search.

Although we use the SAT problem, we believe that the results generalize to other hard problems such as Automated Theorem Proving.

4 Hindsight Experience Replay

Provided a goal to achieve, a common approach in RL is to sample trajectories using some initial exploration policy, then improve the policy by reinforcing behaviors that lead to the goal. However, there is a common pitfall with this approach: the goal can be hard to achieve with the initial policy, leading to a lack of feedback to improve the policy.

To address this issue, Hindsight Experience Replay was introduced [1]. HER deviates from traditional RL by learning a *universal value function* rather than focusing only on the task-specific value function. A universal value function estimates the reachability between states, predicting whether a particular state b can be reached from another state a. To reach a desired goal-state g, actions are taken that lead to states providing high estimates of reaching g.

This method has been applied in ATP, where random exploratory search often fails to construct a proof for the desired theorem [3]. However, despite HER's design to create feedback in challenging environments with sparse rewards, it can overlook critical information in problems that are relatively easy to solve.

To illustrate and prove this limitation, we construct a counterexample similar to the previous section. In this example, a search guided by classical value functions learned with the Bellman equation struggles to find a solution. Moreover, learning a universal value function in this scenario leverages no more feedback than learning a classical value function, leading to the same struggles in finding a solution.

As in the previous section, we work with the family of SAT instances. We also define a sequence of n decisions and corresponding states that incrementally build a binary solution to the instance. However, we introduce an additional final step with two possible states: True and False. Here, True indicates that the constructed candidate solution is valid for the problem, while False indicates otherwise. This final step provides a clear goal-state to HER: $g = \text{True}$.

We define the set of states S for a problem p over n variables as $\{0,1\}^{[n]} \cup \{\text{True}, \text{False}\}$, with the set of actions being binary: $\{0,1\}$.

Definition 8. *Given a problem p over n variables its* dynamics *is:*

Start in state $[]$, at each step up to length n, the transition appends the chosen action to the current state, resulting in a sequence of all actions taken. For a state of length n represented by the binary vector x, the dynamics leads to True *if Check$(x, p) =$* True*; otherwise, it leads to* False*.*

The operator $D^p : S \times \{0,1\} \to S$ implements that dynamics for problem p, taking a state and a binary action as input and outputting the next state.

We now define universal value functions, learned by the method.

Definition 9. *A* universal value function *takes as input a problem p and two states in S, and outputs a binary value.*

Similar to the previous section, the Bellman equation is used for learning. For a universal value function v, a problem p, a pair of states a and b, and the operator D^p, the Bellman equation reads:

$$v(a, b; p) = \max\{v(D^p(a, 0), b; p),\ v(D^p(a, 1), b; p)\}. \tag{2}$$

This equation is implemented in Algorithm 2. At initialization, a set V of universal value functions represents a (learned) prior for SAT-solving. This set is used iteratively to guide the constructions of candidate solutions by attempting to reach the goal-state g. In parallel, the equation is enforced on the set V to learn from sampled constructions.

There are numerous pairs of states a, b to which this equation can be applied. Algorithm 2 enforces the equation for pairs of states sampled in the same construction and with the goal-state g. This is a key feature of HER that we replicate.

Now, we define several concepts for universal value functions that parallel those defined for standard value functions in the previous section.

Definition 10. *A universal value function is* optimal *for problem p over n variables is a universal value function such that for any $s_1, s_2 \in S$, $v(s_1, s_2; p)$ is 1 iff from s_1 there exists a sequence of action leading to s_2 in the problem's dynamics.*

Definition 11. *A* policy *is a randomized function that provided a problem p outputs a sequence of states that follows the dynamics.*

A policy is symmetric *with respect to a set of universal value function V and a state $g \in S$ if it can be written as the following process: Starting from an empty vector, incrementally sample a binary variable x_i and concatenate it to construct $x_{\leq i}$ until a complete vector $x \in \{0, 1\}^n$ is obtained. The binary variable x_i must be sampled according to the sets $V_0 = \{v([x_{<i}, 0], x_{<i}, g; p) |\, v \in V\}$ and $V_1 = \{v([x_{<i}, 1], x_{<i}, g; p) |\, v \in V\}$ such that the probabilities of $x_i = 0$ and $x_i = 1$ must be equivariant to those sets, $\Pr(x_i = 0 |\, (V_0, V_1), x_{<i}, g; p) = \Pr(x_i = 1 |\, (V_1, V_0), x_{<i}, g; p)$.*

Definition 12. *A universal value function v is* monotonic *if for any problem p with n variables and state any $s \in S$, any $x \in \{0, 1\}^n$, and any $i, j \in [n]$ with $i < j$, v satisfies $v(x_{\leq i}, s; p) \geq v(x_{\leq j}, s; p)$.*

Theorem 2. *Let p be a problem over $n = K^2$ variables defined by an aggregation of problems $p_1, \ldots, p_K$ over K variables with index lists $I_1, \ldots, I_k, \ldots, I_K$ where the first element of each I_k is k.*

Let V and $V_1, \ldots, V_K$ be sets of universal value functions and $g =$ True be the goal-state.

Assume:

1. *The set V factories into $V_1, \ldots, V_K$, $v \in V$ iff there exists $v_1 \in V_1, \ldots, v_K \in V_K$ such that for all $x \in \{0, 1\}^n$ and all $i \in [n]$, $v(x_{\leq i}, g; p) = \prod_{k \in [K]} v_k(x_{I_k \cap [i]}, g; p)$.*

Algorithm 2 Hindsight Experience Replay.
Similarly to Algorithm 1, candidate solutions are iteratively sampled guided by a set V of universal value functions to reach a given goal-state g. This uses policy π^{V^t} symmetric w.r.t V^t and g (Definition 11). Simultaneously, the Bellman equation is enforced on V for pairs of sampled states and g.

Inputs: V^0 : an initial set of value functions; p : the problem to solve; g : a final goal-state to reach; i^*, v^* : parameters to provide the true value function (v^*) at index i^*

function HINDSIGHT_EXPERIENCE_REPLAY(V^0, p, g; i^*, v^*)
  for $t \leftarrow 0, 1, \ldots$ **do**
    $s_0^t, s_1^t, \ldots, s_n^t, s_{n+1}^t \sim \pi^{V^t}(p, g)$
    if $s_{n+1} = g$ **then**
      return s_n^t ▷ s_n^t is a solution to problem p
    $V^{t+1} \leftarrow V^t$
    for $i \leftarrow 1, \ldots, n$ **do**
      $V^{t+1} \leftarrow$ BELLMAN_EQUATION(V^{t+1}, s_i^t, g, p; i^*, v^*)
      for $j \leftarrow i+1, \ldots, n+1$ **do**
        $V^{t+1} \leftarrow$ BELLMAN_EQUATION(V^{t+1}, s_i^t, s_j^t, p; i^*, v^*)
function BELLMAN_EQUATION(V, s_i, s_j, p; i^*, v^*)
  if $i \geq i^*$ or $j \leq n$ **then**
    return $\{v \in V \mid v(s_i, s_j; p) = v^*(s_i, s_j; p)\}$
  else
    return $\{v \in V \mid v(s_i, s_j; p) = \max\{v(D^p(s_i, 0), s_j; p),\, v(D^p(s_i, 1), s_j; p)\}\}$

2. *For all $k \in [K]$ and any $v \in V_k$ there exists $v' \in V_k$ such that $v(0, g; p) = v'(1, g; p)$ and $v(1, g; p) = v'(0, g; p)$.*
3. *For all $k \in [K]$, any $v_k \in V_k$ is monotonic.*
4. *For v^* an optimal value function for problems $p_1, \ldots, p_K$, for all $k \in [K]$ either $v^*(0; p_k) = 1$ or $v^*(1; p_k) = 1$.*
5. *For any $v \in V$, $i, j \in [n]$ with $i \leq j \leq n$, $x_1 \in \{0,1\}^i$, $x_2 \in \{0,1\}^j$, $v(x_1, x_2; p) = v^*(x_1, x_2; p)$ for v^* an optimal universal value function.*
6. *For v^* an optimal universal value function for problem p, and any $s \in S$, if $v^*(s, \text{False}; p) = 1$ then for any $v \in V$, $v(s, \text{False}; p) = 1$.*

Then Algorithm 2 with arguments $V^0 = V$, $p = p$, $g = g$, $i^* = K$ and v^* an optimal universal value function for problem p, runs for an expected time of at least $2^{\sqrt{n}-1}$.

Proof. We reduce our claim to that of Theorem 1. The distribution used to sample candidate solutions follows the same constraints as in Theorem 1, with the value functions that guide the search, $v(.; p)$, being replaced by universal value functions evaluated with the goal g: $v(., g; p)$ for $v \in V$. The set V, when applied with g, follows the same constraints as in the previous theorem, with the exception that the Bellman equation is not only enforced with the final state g, but also with the state False and states corresponding to partial solutions. For the state False, by Assumption (6) either v is correct or the state must lead to a

solution. Since the state did not lead to a solution when enforcing the Bellman equation, the set V^t is not impacted.

The Bellman equation procedure is also called with the second state corresponding to partial solutions. By assumption (5), in that case, any universal value function in V^0 (and thus V^t) already matches the output of an optimal universal value function.

Consequently, the set V^t, when evaluated with $g = \text{True}$ in Algorithm 2, is updated in the same manner as in Algorithm 1, under the assumptions.

□

Theorem 2 uses similar assumptions to Theorem 1. Assumptions (1,2) have been adapted for universal value functions, while Assumptions (3,4) remain unchanged. Assumptions (5) and (6) are newly introduced.

Assumption (5) requires that any universal value function in the initial set V^0 is optimal at predicting which partial candidate solutions can be reached from a given state. This condition is satisfied by any optimal value function for the problem and is straightforward to compute.

Assumption (6) requires that any value function in V^0 can determine whether a partial candidate must lead to a full solution, regardless of subsequent actions. This condition is also satisfied by any optimal universal value function.

This section demonstrates that a naïve application of HER and universal value functions does not circumvent the issue identified in the previous section regarding the Bellman equation with classic value functions. We made specific choices in our implementation, and alternative choices could lead to different algorithms with potentially distinct properties. For example, our final states are limited to True or False to indicate whether the problem has been solved. HER could potentially be applied with a different choice of dynamics or final states, which might provide additional feedback to the algorithm. However, to our knowledge, there are no proposals from the literature in that specific direction.

5 Divide-and-conquer

Recent works have explored the application of a divide-and-conquer strategy to tactic-based Automated Theorem Proving (ATP) using Deep Learning [9,5]. A tactic-based theorem prover decomposes a given theorem into subgoals, recursively breaking down these subgoals until elementary true propositions are reached. These decomposition rules, known as *tactics*, are predefined by the theorem prover.

The main challenge for a prover algorithm is to find the appropriate tactics to construct a proof. In the approach we study, policy or value functions are learned via Dynamic Programming to guide the search.

Definition 13. *We define a* problem *by:*

- *A set of possible propositions P;*
- *A goal to prove $g \in P$;*

– *A set of tactics $T \ni t : P \to P^*$, each tactic takes a proposition as input and produces a set of new propositions entailing the input-proposition.*

For an input-proposition, a tactic proves it if it outputs an empty set. In practice, not all tactics apply to every proposition. We say that a tactic is *admissible* for a proposition if it can be applied to it. To represent non-admissibility, we include a dummy proposition False in the set P. If a tactic t is not admissible for a proposition x, then $t(x) = \{\text{False}\}$ and any tactic applied to False will also produce {False}.

A proof can be represented as a graph. Here, the goal is associated with a node, and the application of a tactic generates a set of child nodes. The successive application of tactics forms a tree of propositions, where the leaves entail the goal. In this representation, a problem is solved when this tree has the goal as its root, with the leaves followed by an empty set of child nodes.

Algorithm 3 implements a Dynamic Programming approach for this problem following existing work [9,5]. The specific learning method used is not crucial to our result; thus, we treat it as a parameter in our algorithm (procedure *Learn*). Instead, our algorithm and argument in this section focus on the reliance on value functions in this Dynamic Programming approach.

Although this divide-and-conquer strategy benefits from decomposing the problem into independently solvable sub-problems, its effectiveness is contingent upon the allowed decompositions. Our counterexample forces the algorithm to make decisions before any meaningful decomposition occurs, breaking this advantage.

We define a family of problems that are challenging for Algorithm 3. These problems are parametrized by: a natural number n, which takes values of the form 2^L for some natural number L; and by a hidden binary word $b \in \{0,1\}^n$. We now define the goal, the set of propositions, and the associated set of tactics.

The propositions in P are of the form (l, I, X) where:

– l is a natural number;
– I is an index list;
– X is a sequence of binary values the same size as I.

To these propositions, we add the dummy proposition False to the set P. For a set S of propositions with identical first element l, we note $l(S)$ this element.

The goal g is the proposition with $(l = 0, I = (), X = ())$.

The set of tactics T is composed of 4 tactics:

– *Add 0:* Taking in input a proposition (l, I, X) with $l < n$, it outputs $\{(l + 1, [I, l + 1], [X, 0])\}$.
– *Add 1:* Taking in input a proposition (l, I, X) with $l < n$, it outputs $\{(l + 1, [I, l + 1], [X, 1])\}$.
– *Decompose:* Taking in input a proposition (l, I, X) with $n \leq l < n + L$ and $|I| = 2^a$ for some natural number $a > 1$, it outputs $\{(l{+}1, I_{\leq 2^{a-1}}, X_{\leq 2^{a-1}}), (l{+}1, I_{>2^{a-1}}, X_{>2^{a-1}})\}$.
– *Check:* Taking in input proposition (l, I, X) with $l = n + L$, $I = (i)$, $X = (x)$ and $x = b_i$, then it outputs the empty set $\{\}$.

Algorithm 3 Divide-and-conquer.
The algorithm recursively samples tactics according to its policy to solve the goal and any generated sub-goals. When the algorithm encounters a dead end —indicated by a value function equal to 0— it halts and initiates a new attempt. Simultaneously, a dataset is constructed with: propositions, tactics used, and resulting estimated values. This dataset is then used to learn a policy and a value function, where the learning method is an unspecified procedure *Learn*. The value function v^* returns 1 if all the propositions in a given set are provable; otherwise it returns 0.

Inputs: g : the goal to prove; N : maximum number of tactics to apply by attempt; v^* : optimal value function; L^* : value l at which the optimal value function must be used.

function DIVIDE_AND_CONQUER(g, N; v^*, L^*)
  $D \leftarrow \{\}$ ▷ An empty dataset
  while True **do**
    $\pi \leftarrow$ LEARN(D)
    $v \leftarrow$ LEARN(D)
    $G \leftarrow \{g\}$
    for $i \leftarrow 0, 1, \ldots, N$ **do**
      $p \leftarrow \text{pop}(G)$
      $t \sim \pi(p)$
      Break if t is not admissible for p
      if $l(t(p)) = L^*$ **then** ▷ use a ground truth value function.
        $v' \leftarrow v^*(t(p))$
      else
        $v' \leftarrow v(t(p))$
      $D \leftarrow D \cup \{(p, t, v')\}$
      if $v' = 0$ **then** ▷ If unprovable, stop the proof attempt.
        Break
      $G \leftarrow G \cup t(p)$
      if G is empty **then**
        return Success

When these tactics are applied to a proposition that does not have the required form they output {False}.

The family of problems is easy to solve. Starting from the goal, only the two tactics (*Add 0/1*) are admissible and must be applied n times. Next, the *Decompose* tactic is used until propositions containing only one index i and a value x are obtained. Finally, the *Check* tactic is the only admissible one when $x = b_i$. Thus, the only decisions to be made are between tactics *Add 0* and *Add 1* during the initial n steps.

An algorithm tailored to this family can easily determine the optimal sequence of n tactics (*Add 0* or *Add 1*) by leveraging the feedback from a single attempt, as the binary word b is completely revealed by the possible applications of the *Check* tactic at the end (does $x = b_i$?).

Theorem 3. *For any $n = 2^L, L \in \mathbb{N}$, there exists a problem in the family defined by n and $b \in \{0,1\}^n$ with associated T and g, such that Algorithm 3, given the goal g, any $N \in \mathbb{N}$, and L, takes at least 2^{n-1} steps in expectation to terminate.*

Proof. There is a unique sequence of n tactics *Add 0* and *Add 1* that can generate p (with $l(p) = L$) such that $v^*(t(p)) \neq 0$. This sequence corresponds to the binary word b and is necessary to complete the task.

Dataset D contains the only information about b to build π and v to guide the search. If the optimal sequence determined by b is not applied, Algorithm 3 will terminate its attempt because $v' = v^*(t(p)) = 0$. In D the only information coming from b will be given by $v' = v^*(t(p))$ for p a constructed proposition after n application of tactics *Add*. The value of v' is either 1 for the unique solution in 2^n possibilities, or 0.

This scenario is equivalent to the black-box problem of finding a solution among 2^n possibilities, with no feedback other than "correct" or "incorrect" for each attempt. As a result, the running time is at least 2^{n-1} for one of the problem in the family at any n.

□

Theorem 3 and its proof suggest that even when the decomposition of the initial goal creates rich feedback that could lead to a solution, a Dynamic Programming approach may still fail to efficiently assign credit to early decisions due to merging feedback of independent sub-problems.

While we derived our result using a small set of specific tactics and propositions, we believe our findings generalize to ATP since problems with similar structures could be created in the general framework of theorem proving.

6 Conclusion

In this paper, we defined algorithms for problem-solving with learning-based guidance in their search processes. Each algorithm implements an approach from the literature: learning a value function with the Bellman equation, learning a universal value function with Hindsight Experience Replay, and a divide-and-conquer strategy for tactic-based Automated Theorem Proving.

We constructed counterexample problems that, by design, are easy to solve but provably challenging for the respective algorithms. Our proofs demonstrate that these algorithms struggle because they do not leverage rich feedback from their failed attempts to improve their guidance.

One limitation of our theoretical analysis is its reliance on specific algorithmic implementations. While our algorithms are straightforward implementations of each approach, they may inadvertently introduce specific implementation choices not part of the original approach and limit the scope of our findings. We have sought to make these assumptions explicit in our presentation of each result.

Nonetheless, our work formalizes inefficiencies inherent to classical algorithmic ideas found in the literature. Moreover, our methods offer a framework for analyzing and guiding the development of new algorithms better leveraging learning to accelerate search, leading to more effective problem-solving techniques.

References

1. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., Zaremba, W.: Hindsight experience replay. Advances in neural information processing systems **30** (2017)
2. Arora, S., Barak, B.: Computational complexity: a modern approach. Cambridge University Press (2009)
3. Aygün, E., Anand, A., Orseau, L., Glorot, X., Mcaleer, S.M., Firoiu, V., Zhang, L.M., Precup, D., Mourad, S.: Proving theorems using incremental learning and hindsight experience replay. In: International Conference on Machine Learning. pp. 1198–1210. PMLR (2022)
4. Knuth, D.E.: The art of computer programming, Volume 4, Fascicle 6: Satisfiability. Addison-Wesley Professional (2015)
5. Lample, G., Lacroix, T., Lachaux, M.A., Rodriguez, A., Hayat, A., Lavril, T., Ebner, G., Martinet, X.: Hypertree proof search for neural theorem proving. Advances in Neural Information Processing Systems **35**, 26337–26349 (2022)
6. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
7. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The lean theorem prover (system description). In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25. pp. 378–388. Springer International Publishing, Cham (2015)
8. Pinon, B., Jungers, R., Delvenne, J.C.: Efficiency separation between rl methods: Model-free, model-based and goal-conditioned. arXiv preprint arXiv:2309.16291 (2023)
9. Polu, S., Sutskever, I.: Generative language modeling for automated theorem proving. arXiv preprint arXiv:2009.03393 (2020)
10. Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach. Pearson (2016)
11. Schaul, T., Horgan, D., Gregor, K., Silver, D.: Universal value function approximators. In: International conference on machine learning. pp. 1312–1320. PMLR (2015)
12. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
13. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
14. Sun, W., Jiang, N., Krishnamurthy, A., Agarwal, A., Langford, J.: Model-based rl in contextual decision processes: Pac bounds and exponential improvements over model-free approaches. In: Conference on learning theory. pp. 2898–2933. PMLR (2019)
15. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
16. Sutton, R.S., Modayil, J., Delp, M., Degris, T., Pilarski, P.M., White, A., Precup, D.: Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In: The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2. pp. 761–768 (2011)