Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

On Temporal Differences Algorithms For Continuous Systems

Alexandre Donzé

Report n^o TR-2005-8

May 13, 2005

On Temporal Differences Algorithms For Continuous Systems

Alexandre Donzé

May 13, 2005

Abstract

This report sets a general, intuitive and rigorous framework for designing temporal differences algorithms to solve optimal control problems in continuous time and space. Within this framework, we derive a version of the classical TD(λ) algorithm as well as a new TD algorithm which is similar, but designed to be more accurate and to converge as fast as TD(λ) for the best values of λ, without the burden of finding these values.

Keywords: Optimal Control of Continuous Systems, Dynamic Programming, Optimal Value function, Reinforcement Learning, Temporal Differences Algorithms

Reviewers:

Notes:

How to cite this report:

@techreport { ,
title = { On Temporal Differences Algorithms For Continuous Systems},
authors = { Alexandre Donzé},
institution = { Verimag Technical Report },
number = {TR-2005-8},
year = { },
note = { }
}

1 INTRODUCTION

Dynamic programming (DP) is a powerful technique to compute optimal feedback controllers which was developed at the early stages of the field of optimal control [Bel57]. It was used since then in a wide variety of domains ranging from control of finite state machines, Markov Decision Process (MDP), to optimal control of continuous and hybrid systems. For all these problems, its goal is to compute a so-called *value function* (VF) mapping the states of the system to numbers that reflect the optimal cost to reach a certain objective from these states. Once this function computed, the optimal feedback control can be easily deduced from its values. The main drawback of DP lies in its computational cost. Computing the VF over the whole state space of the system can be prohibitively expensive if this state space contains a large number of states, particularly if it is infinite. This lead some to develop *relaxed* DP techniques, e.g. [Lin03].

In the reinforcement learning (RL) community, the temporal differences (TD) family of algorithms gained a particular interest after several spectacular experimental success, such as TD-Gammon [Tes95], a player of backgammon that outperformed professional human players. Basically, these algorithms work by "learning" the VF while observing simulated trajectories. This family of algorithms is most typically designed for discrete MDPs, but various attempts were also made to adapt it for optimal control of continuous systems, see [SB98] for a survey. Recently, one of these tentatives gave very promising, not to say impressive, results [Cou02] in the control of a continuous mechanical system with a large number of degrees of freedom and input variables. This motivated our interest in studying the applicability of TD algorithms to continuous optimal control problem. The main problem of this study is the lack of theoretical results such as convergence proofs, uniqueness of solutions and so on. For TD algorithms, results are generally available in their original context of MDPs and are not trivial to adapt to the continuous case. On the optimal control side, the most advanced theoretical framework available is that of functional analysis applied to the numerical solutions of Hamilton Jacobi Bellman (HJB) equations To the best of our knowledge, the most complete and rigorous analysis of RL algorithm in this context is [Mun00], but it does not include TD algorithms. Algorithms under study are schemes usually used to solve partial differential equations, finite differences, finite elements, etc. One potential advantage of TD methods over other numerical schemes to compute the solution of HJB equations is that as it is based on simulation of real trajectories, it naturally takes into account the structure of the reach set of the system. By guiding the trajectories towards 'interesting' zones inside the reach set, it is possible to converge quickly to an approximation of the VF that is accurate in these zones and probably coarser in other less interesting parts of the state space.

This article attempts to provide a clarified framework for TD algorithms applied to continuous problems. Our hope is that by clearly identifying and isolating different problems arising in the design of TD algorithms, it will help to develop new and better TD algorithms, and to analyze of their behavior rigorously. Our first result in this direction is the derivation of a version of the classical TD(λ) algorithm along with a new TD algorithm designed to be more accurate, to converge as fast as TD(λ) for the best values of λ, without the burden of finding these values.

The paper is organized as follows. Section 2 recalls the principles of DP, first applied to the simple case of deterministic discrete transition systems, then adapted to continuous problems. Section 3 introduces our continuous framework for TD algorithms, the TD(λ) algorithm instantiated in this framework and our variant TD($\emptyset$). Section 4 present some experimental results obtained on two test problems.

2 DYNAMIC PROGRAMMING

2.1 DP For Discrete Systems

We first consider in this section a purely discrete deterministic system with a state set $\mathcal{X}$, an input or action set $\mathcal{U}$ and a transition map $\rightarrow$: $\mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$. At each pair (x, u) of state-action, a cost function $c(x, u)$ reflects the price of taking action u from state x. In the following, we assume that c is positive and bounded by a constant $\bar{c}$. Given a state x_0 and an infinite input sequence $u = (u_n)$, we can thus define the cost-to-go or *value function* of the trajectory $x_0 \overset{u_0}{\to} x_1 \overset{u_1}{\to} \dots$ as:

$$V^u(x_0) = \sum_{n=0}^{\infty} \gamma^n c(x_n, u_n) \tag{1}$$

where γ is a discount factor lying strictly between 0 and 1 which prevents the infinite sum V^u from diverging. Since $V^u(x_0)$ represents the cost of applying the input sequence u from x_0, the optimal control problem consists in finding a sequence u^* that minimizes this cost, i.e.

$$V^{u^*}(x_0) \leq V^u(x_0), \forall u \in \mathcal{U}^{\mathbb{N}}.$$

Note that $V^{u^*}(x_0)$ does not depend on u^* and thus will be noted $V^*(x_0)$. Instead of searching an optimal sequence, DP aims at computing directly the optimal value function $V^*(x)$ for all $x \in \mathcal{X}$. It relies on *Bellman equation* satisfied by V^* which is easily deduced from (1):

$$V^*(x) = \min_{u,\ x \overset{u}{\to} x'} c(x, u) + \gamma V^*(x') \tag{2}$$

Once V^* has been computed, an optimal sequence u^* is obtained by solving in u the right hand side of Bellman equation which gives a state feedback controller:

$$u^*(x) = \arg\min_{u,\ x \overset{u}{\to} x'} c(x, u) + \gamma V^*(x') \tag{3}$$

Since (2) is a fix point equation, V^* can be computed using fix point iterations. This gives the *value iteration algorithm*. Convergence of this algorithm is guaranteed by the discount factor γ which makes the

Algorithm 1 Value Iteration

```
Init V^0, i ← 0
repeat
    for all x ∈ X do
        V^{i+1}(x) ← min_{u, x -u-> x'} c(x,u) + γ V^i(x')
    end for
    i ← i + 1
until V has converged
```

iteration a contraction. Indeed, it is easy to see that

$$\|V_{i+1} - V_i\|_\infty \leq \gamma \|V_i - V_{i-1}\|_\infty$$

2.2 DP in Continuous Time and Space

In this section, we adapt the previous algorithm to the continuous case. Let $\mathcal{X}$ and $\mathcal{U}$ now be bounded subsets of $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively and $f : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}^n$ the dynamics of the system, so that for all $t \geq 0$,

$$\dot{x}(t) = f(x(t), u(t)) \tag{4}$$

Cost and cost-to-go functions have their continuous counter parts: For each x and u, $c(x, u)$ is a positive scalar bounded by a constant $\overline{c} > 0$, and for any initial state x_0 and any input function $u(\cdot)$,

$$V^{u(\cdot)}(x_0) = \int_0^\infty e^{-s_\gamma t} c(x(t), u(t)) dt$$

Where $x(t)$ and $u(t)$ satisfy (4) with $x(0) = x_0$. The optimal value function, still not depending on $u(\cdot)$, is such that $\forall x \in \mathcal{X}$,

$$V^*(x) = \min_{u(\cdot) \in \mathcal{U}^{\mathbb{R}^+}} \int_0^\infty e^{-s_\gamma t} c(x(t), u(t)) dt$$

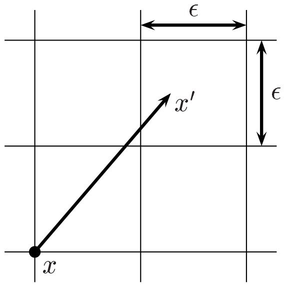

Figure 1: Simple grid of resolution ϵ. V^i is known in the grid points, e.g. in x, whereas $V^i(x')$ has to be interpolated

A straightforward way to apply the discrete value iteration algorithm described in the previous section is first to discretize time and space and then to find an equivalent of the Bellman equation (2). Let us fix a time step Δt and a regular grid $\mathcal{X}_\epsilon$ of resolution ϵ covering $\mathcal{X}$. We evaluate V^* on each point of the grid and interpolate for any x in $\mathcal{X}$ outside $\mathcal{X}_\epsilon$ (see figure 1).

In order to get a transition function, one can classically perform an Euler integration of (4):

$$\begin{aligned} x \xrightarrow{u} x' \quad \Leftrightarrow \quad & x' = x + \Delta x \\ & \text{where } \Delta x = f(x,u)\Delta t \end{aligned} \tag{5}$$

Moreover we can write :

$$\begin{aligned} V^u(x_0) &= \int_0^{\Delta t} e^{-s_\gamma t} c(x(t),u(t))dt \\ &\quad + \int_{\Delta t}^{\infty} e^{-s_\gamma t} c(x(t),u(t))dt \\ &\simeq c(x(0),u(0))\Delta t + e^{-s_\gamma \Delta t} V^u(x(0)+\Delta x) \\ &\simeq c(x_0,u_0)\Delta t + \gamma V^u(x_0+\Delta x) \end{aligned} \tag{6}$$

Where we note $e^{-s_\gamma \Delta t} = \gamma$ then V^* satisfies

$$V^*(x) \simeq \min_{u,\ x \xrightarrow{u} x'} c(x,u)\Delta t + \gamma V^*(x') \tag{7}$$

Which provides a decent equivalent of the discrete Bellman equation. From here, nothing prevents us from applying Algorithm 1 using (7) as a fix point iteration. The complete algorithm is given below.

Algorithm 2 Continuous Value Iteration

Init V_ϵ^0, $i \leftarrow 0$
repeat
  for all $x \in \mathcal{X}$ **do**
    $V_\epsilon^{i+1}(x) \leftarrow \min_{u,\ x \xrightarrow{u} x'} c(x,u)\Delta t + \gamma V_\epsilon^i(x')$
  end for
  $i \leftarrow i+1$
until V_ϵ has converged

Again, Algorithm 2 is guaranteed to converge for the same reasons as Algorithm 1 does, and, moreover, in [Mun00] it is proven that under certain regularity conditions, as ϵ and Δt tend toward zero, V_ϵ tends toward the exact optimal VF of the continuous problem.

2.3 Discussion

Limitations As presented above, adaptation of discrete value iteration algorithm to the continuous case may seem simple. Unfortunately, in practice it will fail for several reasons: first, as it is a direct application

of DP, it has the limitation that Bellman himself called originally the *curse of dimensionality* [Bel57] which expresses the fact that complexity of these algorithms is exponential in the number of dimensions of the system. This renders them impractical for problems in high dimensions. Another limitation of Algorithm 2 more specific to the continuous case is the simplicity of the underlying numerical methods. Explicit Euler integration used in (5) is the simplest method to solve a differential equations and is known to have severe limitations. In particular, it is of order one and is sensitive to stiffness. Also, in (6), the rectangle method is used to approximate integral on interval $[0, \Delta t]$ which is also a method of order one.
A lot can be done to benefit form more clever numerical schemes to solve (4) or to better approximate integral in (6) using more advanced methods of numerical analysis. For instance, backward Euler integration is often preferable to explicit Euler integration [Doy00] and a cost-less trick to improve approximation (6) consists in, instead of considering that $e^{-s_\gamma t}c(x,u)$ is constant on $[0, \Delta t]$, only considering that $c(x,u)$ is constant and integrating formally the exponential, which gives, if $\gamma = e^{-s_\gamma \Delta t}$,

$$V^*(x) \quad \simeq \quad \min_{u,\ x \xrightarrow{u} x'} c(x,u)\frac{(1-\gamma)}{s_\gamma} + \gamma V^*(x') \tag{8}$$

which is a more precise continuous equivalent of the discrete Bellman equation than (7) that can be used in Algorithm 2 [Cou02].

Function approximators The major reason for the curse of dimensionality is the discretization of the state space into a simple grid, an object whose size grows exponentially with the number of dimensions. Among classical alternative solutions are variable resolution discretization [MM99], sparse coarse coding [Sut96], or neural networks [Cou02], [Tes95]. It is worth recalling that all these techniques, including simple grids, approximate a function defined over (a bounded subset of) $\mathbb{R}^n$ using a finite number of parameters $\{\omega_i,\ i \in \mathbb{N}\}$. In the case of simple grids, these parameters are associated to the exponentially many grid points. In the case of neural networks, the parameters are the weights of the "neurons" in the network, whose number is dimension independent. Although methods using sophisticated function approximators like neural networks proved to be potentially very powerful in some practical cases, such as for swimmers in [Cou02], they offer in general no guarantee of convergence or optimality. In fact, function approximators are double-edged swords. On one hand, they provide *generalization*,that is, while the VF is updated at a point x, it is also updated in some neighborhood of x, or more precisely in all points affected by the change of the parameters of the approximator;the bad side is *interference* which is in fact an excessive generalization, e.g. if an update in x affects the VF in y in such a way that it destroys the benefits of a previous update in y. Understanding and controlling the effects of generalization, in particular in the case of non linear function approximators, is a very deep and numerical analysis complicated problem.

Hamilton-Jacobi-Bellman Equation Another problem to be aware of is that of biased convergence. To better understand this point, let us first note that (7) and (8) can be seen in fact as finite differences approximation schemes of the Hamilton-Jacobi -Bellman (HJB) equation (basically obtained from those equations by dividing each term by Δt and making Δt going to zero):

$$\min_{u \in \mathcal{U}} \left(c(x,u) + \frac{\partial V^*}{\partial x} \cdot f(x,u) - s_\gamma V^*(x) \right) = 0 \tag{9}$$

from which we define the *Hamiltonian*:

$$\mathcal{H}(x) = \min_{u \in \mathcal{U}} \left(c(x,u) + \frac{\partial V^i}{\partial x} \cdot f(x,u) - s_\gamma V^i(x) \right) \tag{10}$$

Algorithms for computing V^* thus often try to minimize $\mathcal{H}$ in order to find a solution to (9). The problem is that there might be an infinite number of *generalized* solutions of (9) i.e. functions that satisfy $\mathcal{H}(x) = 0$ *almost everywhere*, while being very far from the true value function V^* [Mun00]. Thus, if no guarantee is given by the algorithm other than the minimization of $\mathcal{H}(x)$,then no one can tell whether the computed solution is optimal or even near to the optimal VF.
Munos uses the theory of viscosity solutions to prove that Algorithm 2 actually converges toward the true value function but this result is limited to the case of grids.

3 TEMPORAL DIFFERENCES ALGORITHMS

DP algorithms presented so far compute the next estimation $V^{i+1}(x)$ based on $V^i(x)$ and $V^i(x')$ where x' is in the spatial neighborhood of x. The slightly different point of view taken by temporal differences (TD) algorithms is that instead of considering that x' is the *neighbor in space* of x they consider it as its *neighbor in time* along a trajectory. This way, with the same algorithm, $V^{i+1}(x)$ would be updated from $V^i(x(0))$ and $V^i(x(\Delta t))$, where $x = x(0)$ and $x' = x(\Delta t)$ are then viewed in the context of a whole trajectory $(x(t))_{t>0}$. Then, pursuing in this spirit, it is clear that if $V^i(x(\Delta t))$ holds interesting information for the update of $V^i(x(0))$, then it is also the case for $V^i(x(t))$ for all $t > 0$. Henceforth, TD algorithms update their current approximation of the VF thanks to data extracted from complete trajectories.

3.1 General Framework

In section 2, we discussed the fact that DP algorithms were bound to different issues and problematics: function approximation, numerical analysis etc. This was the case for value iteration algorithm and its adaptation to continuous time and space, and this is also the case for TD algorithms. In this subsection, we propose a general, high level framework common to most of TD algorithms that will allow us to separate those different issues, assuming that some are solved and focusing on the others. This high-level framework is represented by Algorithm 3.

Algorithm 3 Generic TD algorithm

1: **Init** V^0, $i \leftarrow 0$
2: **repeat**
3:    Choose $x_0 \in \mathcal{X}$
4:    Compute $(x(t))_{t\in[0,T]}$ using $u(\cdot)$, starting from x_0
5:    Compute update trace $e(\cdot)$ from $x(\cdot)$ and $u(\cdot)$
6:    **for all** $t \in \mathbb{R}^+$ **do**
7:       $V^{i+1}(x(t)) \leftarrow V^i(x(t)) + e(t)$
8:    **end for**
9:    $i \leftarrow i + 1$
10: **until** Stopping condition is **true**

To implement an instance of this generic algorithm, we need the non-trivial following elements:

- **A simulator** that can generate a trajectory of duration $T > 0$ $(x(t)))_{t\in[0,T]}$ given an initial state x_0 and an input function u.

- **A function approximator** to be used to represent any function V from $\mathcal{X}$ to $\mathbb{R}^+$ and allowing us to simply write $V(x)$ for any x in $\mathcal{X}$ e.g. lines 7. Questions here are, as already discussed in previous sections, whether to use simple grids, linear function approximators, or non linear function approximators such as neural networks etc.

- **An update function** which, given a function V, a state x and a quantity e, alters V^i into V^{i+1} so that $V^{i+1}(x)$ is equal to or near $V^i(x)+e$, which is written line 7 as an assignment statement. When parametrized function approximators are used, updating can be done e.g. by gradient descent on the parameters [Doy00].

- **A policy**, or controller, which returns a control input at each time t, needed by the simulator to compute trajectories. In the case of *optimistic policy iteration* [Tsi02], u is the feedback control obtained from V^i using the $\arg\min$ of left hand side of (10). It is said to be *optimistic* because if V^i is the optimal VF, then this is an optimal control input.

- **An update trace generator** used to extract information from the computed trajectory $x(\cdot)$ and to store it into the update trace $e(.)$ (line 5). In the field of RL, $e(.)$ is classically called the *eligibility trace* [SB98].

Variants of TD algorithms differ in the way this last point is implemented, as discussed in the following sections.

3.2 Continuous TD(λ)

3.2.1 Formal algorithm

The objective of TD algorithms is to build an improved estimation of the VF based on the data of computed trajectories. The main additional information provided by a trajectory $x(\cdot)$ is the values of the cost function along its course. If we combine these values with the current estimation V^i, we can then build a new estimation for a given horizon $\tau > 0$, which is:

$$\tilde{V}_\tau(x_0) = \int_0^\tau e^{-s_\gamma t} c(x,u)dt + e^{-s_\gamma \tau} V^i(x(\tau)) \tag{11}$$

In other terms, the estimation $\tilde{V}_\tau(x_0)$ relies on a portion of the trajectory of duration τ and on the old estimation $V^i(x(\tau))$ at the end of this portion. Let us remark that if V^i is already the optimal value function V^*, then $\tilde{V}^i_\tau = V^*$ for all $\tau > 0$, this equality being, in fact, a generalization of Bellman equation. Question is then, how to choose horizon τ ? In the case of TD(λ), the algorithm constructs a new estimation $\tilde{V}_\lambda$ which is a combination of the $\tilde{V}_\tau(x_0)$ for all $\tau > 0$. In our continuous framework, we define for $s_\lambda > 0$:

$$\tilde{V}^i_\lambda(x_0) \quad = \quad \int_0^\infty s_\lambda e^{-s_\lambda \tau} \tilde{V}^i_\tau(x_0) d\tau \tag{12}$$

Several remarks about this definition:

- Equation (12) means that $\tilde{V}^i_\lambda$ is constructed from all $\tilde{V}^i_\tau$, $\tau > 0$, each one of them contributing with an exponentially decreasing amplitude represented by the term $e^{-s_\lambda \tau}$. In other words, the further $\tilde{V}^i_\tau$ looks into the trajectory, the less it contributes to $\tilde{V}^i_\lambda$.
- This definition is sound in the sense that it can be seen as a convex combination of $\tilde{V}^i_\tau, \tau > 0$. In effect, it is true that:

  $$\int_0^\infty s_\lambda e^{-s_\lambda \tau} d\tau \quad = \quad 1$$

  Thus, if each $\tilde{V}^i_\tau$ is a sound estimation of V^*, then $\tilde{V}^i_\lambda$ is a sound estimation of V^*.
- This definition is consistent with the original definition of TD(λ) algorithm [SB98]. In fact, by choosing a fixed time step $\Delta\tau$, assuming that $\tilde{V}^i_\tau$ is constant on $[\tau, \tau + \Delta\tau]$ and summing $s_\lambda e^{-s_\lambda \tau}$ on this interval, we get:

  $$\begin{aligned} \tilde{V}^i_\lambda(x_0) \quad &\simeq \quad (1-\lambda)\sum_{k=0}^\infty \lambda^k \tilde{V}^i_{k\Delta\tau}(x_0) \\ &\text{where } \lambda = e^{-s_\lambda \Delta\tau} \end{aligned} \tag{13}$$

  which is the usual discrete TD(λ) estimation built upon the k-step TD estimation

  $$\begin{aligned} \tilde{V}^i_{k\Delta\tau}(x_0) \quad &= \quad \int_0^{k\Delta\tau} e^{-s_\gamma t} c(x,u)dt + e^{-s_\gamma \tau} V^i(x(k\Delta\tau)) \\ &\simeq \quad \sum_{j=0}^{k-1} \gamma^j c(x(j\Delta\tau), u(j\Delta\tau)) + e^{-s_\gamma k\Delta\tau} V^i(x(k\Delta\tau)) \\ &= \quad \sum_{j=0}^{k-1} \gamma^j c(x_j, u_j) + \gamma^k V^i(x_k) \\ &\text{where } \gamma = e^{-s_\gamma \Delta\tau} \end{aligned} \tag{14}$$

3.2.2 Implementation

In practice, we can only compute trajectories with finite duration T. As a consequence, $\tilde{V}_{\tau}^{i}$ can only be computed for $\tau \leq T$. In this context, (12) is replaced by

$$\begin{aligned} \tilde{V}_{\lambda}^{i}(x_0) &= \int_0^T s_\lambda e^{-s_\lambda \tau} \tilde{V}_{\tau}^{i}(x_0) d\tau + e^{-s_\lambda T} \tilde{V}_T^i(x_0) \\ &\simeq (1-\lambda) \sum_{k=0}^{N-1} \lambda^k \tilde{V}_{k\Delta\tau}^{i}(x_0) + \lambda^N \tilde{V}_T^i(x_0) \\ & \text{where } T = N\Delta\tau \end{aligned} \tag{15}$$

Combining (15) and (14), one can show that:

$$V^i(x_0) - \tilde{V}_{\lambda}^{i}(x_0) \simeq \sum_{j=0}^{N} \lambda^j \gamma^j \delta_j \tag{16}$$

where

$$\delta_j = c(x_j, u_j) + \gamma V^i(x_{j+1}) - V^i(x_j) \tag{17}$$

is what is usually referred as the *temporal difference error* [Tsi02]. This expression provides for an easy implementation given in Algorithm 4 which refines in fact line 5 of Algorithm 3. There, we assume that the trajectory has already been computed with a fixed time step $\Delta t = \Delta\tau$, thus at times $t_j = j\Delta t$, $0 \leq j \leq N+1$.

Algorithm 4 TD(λ): Update traces

```
for j = 0 to N do
    δ_j ← c(x_j, u_j) + γV^i(x_{j+1}) − V^i(x_j)
end for
e(t_N) ← δ_N
for k = 1 to N do
    e(t_{N−k}) ← e(t_k) + (λγ)e(t_{N−k+1})
end for
```

Note that the computation complexity of the update trace is in $O(N)$ where $N+2$ is the number of computed points in the trajectory x. Thus the overall complexity of the algorithms depend only on the number of trajectories to be computed to obtain a fair approximation of V^*.

3.2.3 Qualitative interpretation of TD(λ)

In the previous sections, we started by presenting TD(λ) as an algorithm that compute a new estimation of the VF using a trajectory and older estimations. This allowed us to provide a continuous formulation of this algorithm and an intuition of why it should converge towards the true VF. Then, using a fix time step and numerical approximations for implementation purposes, we came to equation (16) and Algorithm (4). These provide another intuition of how this algorithm behaves.

The TD error δ_j (17) can be seen as a local error in $x(t_j)$ (in fact, it is an order one approximation of the Hamiltonian $\mathcal{H}(x(t_j))$). Thus, (16) means that the *local* error in $x(t_j)$ affects the *global* error estimated by TD(λ) in $x(t_0)$ with a shortness factor equal to $(\gamma\lambda)^j$. λ ranges from 0 to 1 (in the continuous formulation, s_λ ranges from 0 to ∞). When $\lambda = 0$, then only local errors are considered, as in value iteration algorithm. When $\lambda = 1$ then errors along the trajectory are fully reported to x_0. Intermediate values of λ are known to provide better results than these extreme values. But how to choose the best value for λ remains an open question in the general case. Our intuition, together with experiments, tend to show that higher values of λ often produce larger updates, resulting in a faster convergence, at least at the beginning of the process, but also often return a coarser approximation of the VF, when it does not simply diverge. On the other hand, smaller values of λ result in a slower convergence but toward a more precise approximation. In the next section, we use this intuition to design a variant of TD(λ) that combines the qualities of high and low values of λ without the burden of finding its best value.

3.3 TD($\emptyset$)

3.3.1 Idea

The new TD algorithm that we propose is based on the intuition about local and global updates presented in the previous section. Global updates are those performed by $TD(1)$ whereas local updates are those used by $TD(0)$. The idea is that global updates should only be used if they are "relevant". In other cases, local updates should be performed. To decide whether the global update is "relevant" or not, we use a monotonicity argument: from a trajectory $x(\cdot)$, we compute an over-approximation $\bar{V}(x(t))$ of $V^*(x(t))$, along with the TD error $\delta(x(t))$. Then, if $\bar{V}(x(t))$ is less than, and hence necessary better than, the current estimation of the value function $V^i(x(t))$, it is chosen as a new estimation to be used for the next update. In the other case,$V^i(x(t)) + \delta(x(t))$ is used instead.
Let us first remark that since c is bounded and $s_\lambda > 0$, then $V^*(x) \leq V_{\max}$, $\forall x \in \mathcal{X}$, where

$$V_{\max} \quad = \quad \int_0^\infty e^{-s_\gamma t}\, \bar{c}\, dt = \frac{\bar{c}}{s_\lambda} \tag{18}$$

This upper-bound of V^* represents the cost-to-go of an hypothetic forever worse trajectory, that is, a trajectory for which at every moment, the pair state input (x, u) has the worse cost $\bar{c}$. Thus, $V_{\max}$ could be chosen as a trivial over-approximation of $V^*(x(t))$. In this case, our algorithm would be equivalent to $TD(0)$. But if we assume that we computed a trajectory $x(\cdot)$ on the interval $[0, T]$, then a better over-approximation can be obtained:

$$\bar{V}(x_0) = \int_0^T e^{-s_\lambda t} c(x, u) dt + e^{-s_\gamma T} V_{\max} \tag{19}$$

It is easy to see that (19) is indeed an over-approximation of $V^*(x(0))$: it represents the cost of a trajectory that would begin as the computed trajectory $x(\cdot)$ on $[0, T]$, which is at best optimal on this finite interval, and then from T to ∞ it behaves as the ever worse trajectory. Thus, $\bar{V}(x_0) \geq V^*(x_0)$.

3.3.2 Continuous Implementation of TD($\emptyset$)

In section 3.2.2 we fixed a time step Δt and gave a numerical scheme to compute estimation $\tilde{V}_\lambda$. This was useful in particular to make the link with discrete TD(λ). In this section, we give a continuous implementation of TD($\emptyset$) by showing that the computation of $\bar{V}$ can be coupled with that of the trajectory $x(\cdot)$ in the solving of a unique dynamical system.
Let $x_V(t) = \int_0^t e^{-s_\gamma r} c(x, u) dr$. Then,

$$\bar{V}(x_0) = x_V(T) + e^{-s_\gamma (T)} V_{\max}$$

and more generally, for all $t \in [0, T]$,

$$\bar{V}(x(t)) = e^{s_\gamma t}(x_V(T) - x_V(t)) + e^{-s_\gamma (T-t)} V_{\max}$$

x_V can be computed together with x by solving the problem:

$$\left\{ \begin{array}{ccc} \dot{x}(t) & = & f(x(t), u(t)) \\ \dot{x}_V(t) & = & e^{-s_\gamma t} c(x(t), u(t)) \\ x(0) = x_0 & , & x_V(0) = 0 \end{array} \right. \tag{20}$$

From there, we can give Algorithm 5.
Several remarks:

Algorithm 5 TD($\emptyset$)

Init V^0 with $V_{\max}$, $i \leftarrow 0$
repeat
  Choose $x_0 \in \mathcal{X}$
  Compute $(x(t))_{t\in[0,T]}$ and $(x_V(t))_{t\in[0,T]}$ by solving (20)
  Compute $\bar{V}(x(\cdot))$ and $\delta(x(\cdot))$
  for all $t \in [0, T]$ **do**
    $V^{i+1}(x(t)) \leftarrow \min(\bar{V}(x(t)), V^i(x(t)) + \delta(x(t)))$
  end for
until Stopping condition is **true**

- Initializing V^0 to $V_{\max}$ forces a certain monotonicity with respect to i. This monotonicity is not strict since when local updates are made, nothing prevents $\delta(x(t))$ from being positive, but during the first trajectories at least, as long as they pass through unexplored states, $\bar{V}_T$ will be automatically better than the pessimistic initial value. Note also that if it happens that V^* is known in some special states (e.g. on stationary points), V^0 is initialized to these values.

- Computing x and x_V (and hence $\bar{V}$) together in the same ordinary differential equation (ODE) facilitates the use of variable time step size integration, the choice of which can be left to a specialized efficient ODE solver, and thus permits a better control of the numerical error at this level.

4 EXPERIMENTAL RESULTS

To compare the performances of TD($\emptyset$) with TD(λ) for various values of λ, we implemented and applied Algorithms 4 and 5 to deterministic, continuous versions of two classical problems in the field of RL:

The continuous walker: a one dimensional problem in which a robot must exit the zone $[-1, 1]$ as quickly as possible. Equations are given by

$$\dot{x} \quad = \quad \begin{cases} u & \text{if} \quad -1 < x < 1 \\ \min(0, u) & \text{if} \quad x = 1 \\ \max(0, u) & \text{if} \quad x = -1 \end{cases} \tag{21}$$

Input u represents the speed of the walker in either direction. It is bounded in absolute value by 1. Thus, in the context of optimal control, we can restrict to a binary decision problem where $u = 1$ or $u = -1$. The cost function is:

$$c(x) \quad = \quad \begin{cases} 1 & \text{if} \quad -1 < x < 1 \\ 0 & \text{if} \quad x = -1 \text{ or } x = -1 \end{cases} \tag{22}$$

The swing-up of a pendulum with limited torque. The goal is to drive a pendulum to the vertical position. Dynamics is described by a second order non linear differential equation:

$$\begin{aligned} \ddot{\theta} \quad &= \quad -\mu\dot{\theta} + g\sin\theta + u \\ & \quad\quad -10 < \dot{\theta} < 10, \; -3 < u < 3 \end{aligned} \tag{23}$$

The cost function is $c(\theta) = (1 - \cos(\theta))$. It is thus minimal, equal to 0 when the pendulum is up ($\theta = 0$) and maximal, equal to 2, when the pendulum is down. μ is a friction parameter and g the gravitational constant.

For both of these problems, we used regular grids and linear interpolation to represent values of V^i. We call a *sweep* a set of trajectories for which the set of initial states cover all points of a grid. For both problems, we then applied TD(λ) and TD($\emptyset$) to such set of trajectories and after each sweep, we measured the 2-norm

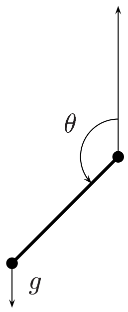

Figure 2: Pendulum

of the Hamiltonian error (noted $||\delta||_2$) over the state space. This allowed to observe the evolution of this error depending on the number of sweeps of the state space performed.
The results are given on figure 3 and 4. The interesting thing to observe is that for the simple problem of the walker, the value of λ that worked the best in our experiments was 0, whereas for the pendulum, this value was more around 0.7. Not surprisingly, in the first case, we see that the error curve of TD($\emptyset$) closely follows that of $TD(0)$, even performing slightly better. For the second problem, TD($\emptyset$) seems to perform better than TD(λ) for any value of λ. This leads to the very encouraging observation that TD($\emptyset$) seems to behave at least as well as TD(λ) for the best value of λ, independently of this value and of the problem. Of course, more experiments are needed to draw a firm conclusion.

5 CONCLUSIONS

We have presented a framework for the design of temporal differences algorithms for optimal control problems in continuous time and space. This framework is attractive as it clearly separates different issues related to the use of such algorithms, namely:

1. How to choose and compute trajectories - which initial state, and which choice of u - in order to ensure that the VF will be accurately and quickly computed in the interesting part of the state space ?

2. How to represent the VF efficiently in order to break the curse of dimensionality ?

3. How to update th VF based on simulated trajectories ?

After laying down the context of DP and our framework, we focused on question 3 and presented a continuous version of TD(λ) which was shown to be consistent after appropriate discretization with the original discrete version of the algorithm Process. We then discussed its efficiency with respect to the parameter λ, and presented a variant, TD($\emptyset$), that we found to be experimentally as efficient as TD(λ) for the best values of λ. Future work will include improvement of this algorithm and its application to higher dimensional problems, and consequently an investigation of Questions 1 and 2 above,a necessary step toward scaling up.

References

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957. 1, 2.3

[Cou02] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002. 1, 2.3, 2.3

[Doy00] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000. 2.3, 3.1

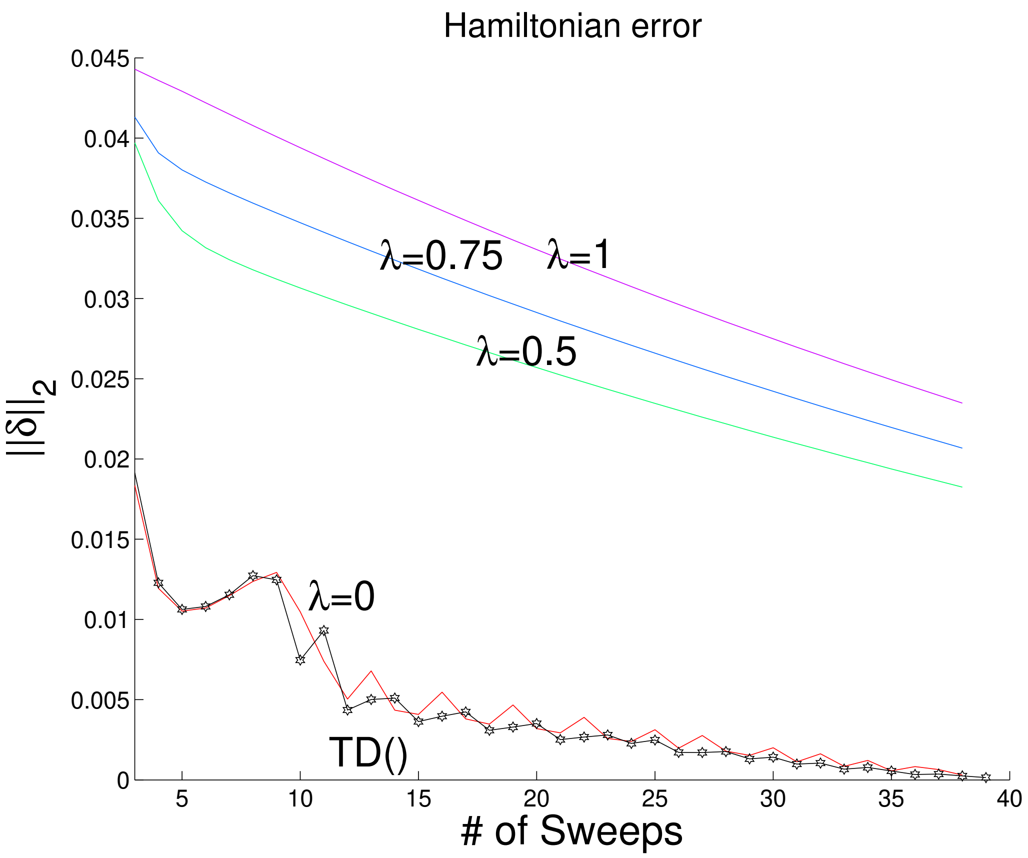

Figure 3: Hamiltonian error for walker

[Lin03] Bo Lincoln. *Dynamic Programming and Time-Varying Delay Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, May 2003. 1

[MM99] Rémi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *International Joint Conference on Artificial Intelligence*, 1999. 2.3

[Mun00] Rémi Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, 40(3):265–299, 2000. 1, 2.2, 2.3

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. 1, 3.1, 3.2.1

[Sut96] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996. 2.3

[Tes95] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995. 1, 2.3

[Tsi02] John N. Tsitsiklis. On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3:59–72, 2002. 3.1, 3.2.2

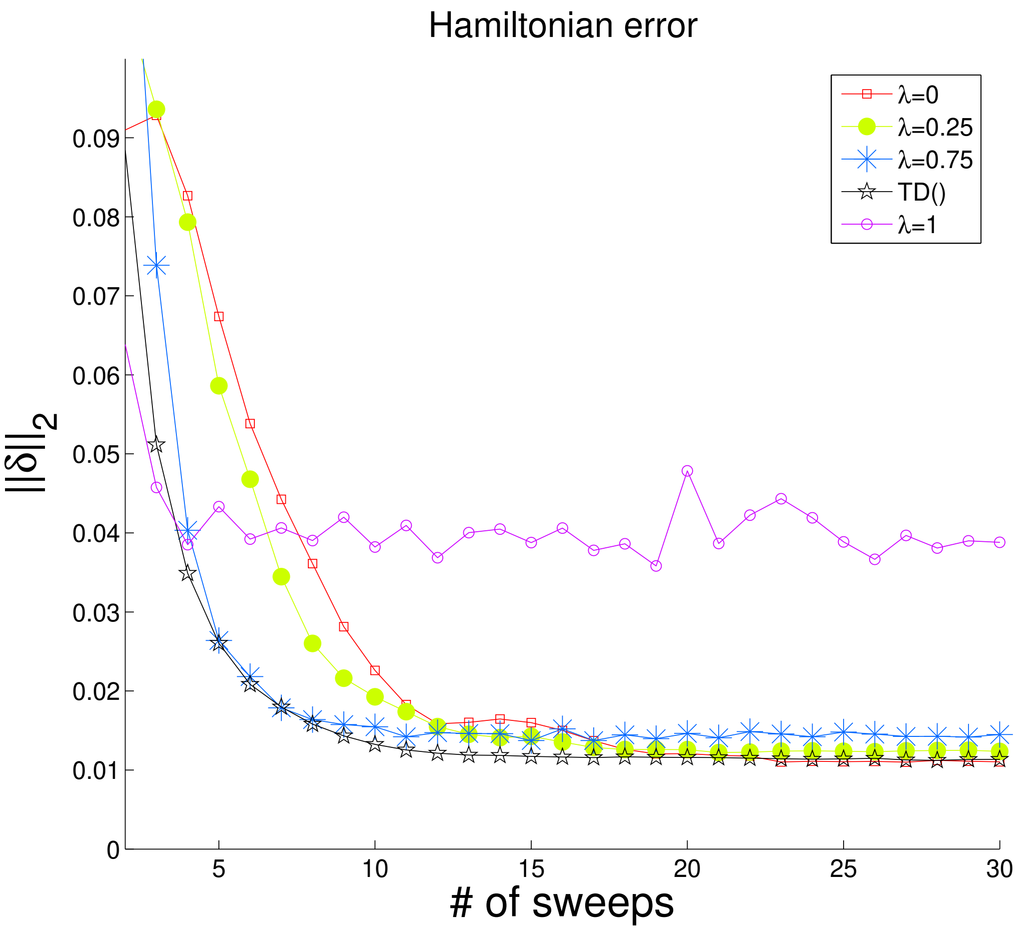

Figure 4: Hamiltonian error for pendul