Tree-based Variable Selection for Dimensionality Reduction of Large-scale Control Systems

Andrea Castelletti, Stefano Galelli, Marcello Restelli and Rodolfo Soncini–Sessa
Department of Electronics and Information
Politecnico di Milano, Milano, Italy
Email: {castelle,galelli,restelli,soncini}@elet.polimi.it

***Abstract*—This paper is about dimensionality reduction by variable selection in high–dimensional real–world control problems, where designing controllers by conventional means is either impractical or results in poor performance.**
In this paper we propose a novel model–free variable selection approach that, starting from a dataset of one–step state transitions and rewards, identifies which state, control, and disturbance variables are most relevant for control purposes, and reduces the problem dimensionality by removing the others. The core of the procedure is the Recursive Variable Selection (RVS) algorithm, which, starting from the subset of variables needed to explain the reward, recursively repeats the variable–selection procedure on the state variables that have been selected, but whose transition model still needs to be explained. The set of selected variables is incrementally built by adding the best variables provided by a ranking algorithm based on a statistical measure of significance that accounts for non-linear dependencies.
The effectiveness of the proposed methodology is tested on two real–world control problems: balancing of a two–wheeled inverted–pendulum robot and the operation of Tono Dam (JP) modeled with a coupled 1D hydrodynamic-ecological model. Preliminary results show that the proposed variable selection approach significantly simplifies the learning of good control policies and can highlight interesting properties of the system to be controlled.

I. Introduction

Large–scale, real–world systems are usually described by complex non–linear, stochastic models with many continuous state and control variables whose actual relevance to the system dynamics is often unknown. The control of such systems is often formalized as a Markov Decision Process (MDP), for which high–dimensional, continuous state and control spaces do not allow to use exact algorithms to find optimal solutions. To solve such problems, research in dynamic–programming and reinforcement–learning has produced a variety of approximated algorithms which can be classified into two (not disjoint) main categories: *solution space reduction* approaches, which search the best solution (i.e., the optimal policy or the optimal value function) to the original MDP within a predefined (usually parameterized) subset of candidate solutions, and *model reduction* ones, which compute the optimal solution to a simplified version of the original MDP. In this paper, we focus on the latter approach[1] where the control problem is simplified by relying on a simpler, computationally efficient, reduced (order) model that mimics the dynamic behavior of the underlying large-scale system.

In literature many reduction techniques based on the heuristic simplification of the model structure have been proposed [1]. Unfortunately, requiring a certain amount of user interaction, the usage of these techniques is often limited to domain experts. On the other hand, there is a great interest in automatic model reduction techniques, where the model dimensionality is reduced by projecting the original systems of state transition equations into a lower-dimension subspace (e.g., using Singular Value Decomposition or moment matching methods).

A common approach to model reduction is *state aggregation*, where "similar" states are grouped together and treated as a single state [2]. In particular, *model minimization* [3], [4], aims at finding the smallest aggregated state of a MDP that preserves the same properties of the original MDP. This can be obtained by aggregating states that are equivalent according to the stochastic bisimulation homogeneity notion [5]. However, the computational complexity of this approach and the fact that it can be applied only to finite MDPs prevent its use to reduce real–world MDP models. A particular form of state aggregation is the one realized by variable selection, where a reduced MDP model is built by considering only a subset of the original state variables (in this way, the states that differ only for the values of the removed variables are aggregated). Following this approach, it is possible to formalize the problem as a factored MDP [6], thus providing more information about the structure of the problem. Factored MDPs represent a complex state space using state variables and the transition model using a dynamic Bayesian network. This representation often allows an exponential reduction in the representation size of structured MDPs, but the complexity of exact solution algorithms for such MDPs can grow exponentially in the representation size [7]. In [3], the authors propose

[1]Nonetheless, we will experimentally show that even algorithms in the first category can benefit from model reduction.

978-1-4244-9888-8/11/$26.00 ©2011 IEEE

a model minimization algorithm for MDPs with factored space, which has been extended to factored control spaces in [8]. Most of these approaches assume to work with finite MDPs with perfect knowledge of the factored models of the state–transition and reward functions. Some approaches have proposed *online* methods to estimate the factored model both for finite MDPs [9], [10] and for continuous ones [11], [12].

In this paper, we propose a model–free offline variable selection approach for reducing the complexity of solving high–dimensional MDPs. The goal is to identify which state, control, and disturbance variables are actually required for solving the MDP through an accurate estimate of the optimal value function. As we will show, this goal can be restated as: 1) select the variables on which the reward function is defined, 2) recursively add the variables needed to estimate the transition model of the state variables previously selected. Differently from most of the works on variable selection in MDPs, we do not assume any prior knowledge about the MDP or its factored representation. The *recursive variable selection* (RVS) approach proposed in this paper works on a dataset of experience tuples (each tuple is a sample of the state transition model and the reward function) directly collected from the system to be controlled or sampled from a generative model. Variables selected by this algorithm derive from the union of the variable subsets needed to accurately approximate the reward function and those state–transition models that are useful for value function estimation. For each function that needs to be modeled, the variable subset is incrementally built following the *iterative variable selection* (IVS) procedure: at each iteration, the variable subset is enlarged with the best variable according to a *variable ranking* (VR) algorithm. Following [13], each variable is scored by estimating its contribution (in terms of variance reduction) to the building of an ensemble of regression trees that models the required function. The procedure embodies some very important properties: it is fully automated, independent on domain experts and system knowledge, and suitable for non-linear processes; it has high potential in terms of complexity reduction, thus allowing for the control of large-scale systems; finally, it provides interpretation of the reduced model structure.

The rest of the paper is structured as follows. Next section introduces basic notation and concepts about MDPs. In Section III, after defining and providing conditions for lossless dimensionality reduction in MDPs, we formally define the lossless variable selection problem. Section IV describes the sample–based recursive variable selection algorithm which represents the main contribution of this paper. The proposed approach is then demonstrated on two real–world case studies whose results are shown and discussed in Section V. Finally, Section VI draws conclusions and proposes future research directions.

II. Notation

This section introduces the Markov Decision Process (MDP) notation that will be used throughout the paper. We denote with $\mathbf{X} \subset \mathbb{R}^{n_x}$ the feasible state set spanned by n_x state variables $\mathcal{X} = \{X^1, \dots, X^{n_x}\}$ and with $\mathbf{U} \subset \mathbb{R}^{n_u}$ the feasible control set spanned by n_u control variables $\mathcal{U} = \{U^1, \dots, U^{n_u}\}$. Given the state vector $\mathbf{x}_t \in \mathbf{X}$ at time t, the next state vector at time $t+1$ is defined by the state–transition (vector) equation:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t), \tag{1}$$

where $\mathbf{u}_t \in \mathbf{U}$ is a control vector and $\mathbf{w}_t \in \mathbf{W} \subset \mathbb{R}^{n_w}$ is a vector of n_w random disturbances $\mathcal{W} = \{W^1, \dots, W^{n_w}\}$ sampled from some probability distribution $P(d\mathbf{w}_t|\mathbf{x}_t, \mathbf{u}_t)$, and $\mathbf{f} = [f^1, \dots, f^{n_x}]$ is a vector of n_x state transition functions, one for each state variable. At each time step, taking control $\mathbf{u}_t$ in state $\mathbf{x}_t$ produces a bounded reward value $r_{t+1} \in R \subset \mathbb{R}$ which may depend on the disturbance $\mathbf{w}_t$:

$$r_{t+1} = g(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t). \tag{2}$$

A *stationary policy* is a (time–independent) mapping $\pi : \mathbf{X} \mapsto \mathbf{U}$ from the state space to controls. In this paper, the quality of a policy π is measured by the infinite horizon discounted reward. In this context, we can defined a value function $V^\pi : \mathbf{X} \mapsto \mathbb{R}$ that maps each state to its value under policy π:

$$V^\pi(\mathbf{x}) = \mathop{\mathbb{E}}_{\mathbf{w}} \left[\sum_{t=0}^{\infty} \gamma^t g(\mathbf{x}_t, \pi(\mathbf{x}_t), \mathbf{w}_t) | \mathbf{x}_0 = \mathbf{x} \right], \forall \mathbf{x} \in \mathbf{X},$$

where $\gamma \in [0, 1)$ is a discount factor and the state sequence is generated according to $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \pi(\mathbf{x}_t), \mathbf{w}_t)$. The optimal value function V^* is defined as:

$$V^*(\mathbf{x}) = \max_\pi V^\pi(\mathbf{x}), \forall \mathbf{x} \in \mathbf{X}.$$

The optimal control problem can be formulated as a dynamic program yielding the Bellman equation [14]:

$$V^*(\mathbf{x}) = \max_{\mathbf{u} \in \mathbf{U}} \mathop{\mathbb{E}}_{\mathbf{w}} \left[g(\mathbf{x}, \mathbf{u}, \mathbf{w}) + \gamma V^*(\mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w})) \right], \forall \mathbf{x} \in \mathbf{X}. \tag{3}$$

Solving such recursive equation means to find the optimal value function for the entire state space. Once the optimal function has been computed, the optimal control policy can be derived as follows:

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{u} \in \mathbf{U}} \mathop{\mathbb{E}}_{\mathbf{w}} \left[g(\mathbf{x}, \mathbf{u}, \mathbf{w}) + \gamma V^*(\mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w})) \right], \forall \mathbf{x} \in \mathbf{X}.$$

For any MDP there exists at least one deterministic stationary policy π^* that attains the optimal value function V^*.

When the model of the MDP is known, several well–known methods can be used to solve the system (e.g., dynamic programming [14] and linear programming [15]). On the other hand, when the model is unknown, reinforcement–learning techniques [16] come into the picture. In any case, when the system to be controlled is too complex, exact algorithms become impractical and approximate algorithms or model reduction techniques are usually employed.

III. Lossless Dimensionality Reduction in MDPs

When the number of state and control variables is large, solving an (even finite) MDP gets intractable. This problem is known as the Bellman's curse of dimensionality [14]. Since there is no unique set of state variables that describe any given system, many different sets of variables may be selected to yield a complete system description. One approach to solve high–dimensional problems is to reduce the number of state and control variables by projecting the original model to a lower dimensional one. This solution is effective when the MDP model is highly redundant and correlation between variables can be considered to identify the most significant dimensions.

For sake of brevity, in the following we will denote with $\mathbf{v}_t^i \in \mathbf{V}^i = \mathbf{X} \times \mathbf{U} \times \mathbf{W} \subset \mathbb{R}^n$ the values of n input variables $\mathcal{V}^i = \{V_1^i, \ldots, V_n^i\} = \mathcal{X} \cup \mathcal{U} \cup \mathcal{W}$ (where $n = n_x + n_u + n_w$) of the state transition functions and of the reward function at time t: $\mathbf{v}_t^i = [\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t]$. Similarly, we denote with $\mathbf{v}_t^o = [\mathbf{x}_{t+1}, r_{t+1}] \in \mathbf{V}^o = \mathbf{X} \times R$ the values of output variables (next state and reward values) at time t ($\mathcal{V}^o = \mathcal{X} \cup R$).

We consider a vector projection (dimensionality reduction) function $\boldsymbol{\rho} = [\rho_x, \rho_u, \rho_w]$, that, given an input vector $\mathbf{v}_t^i$, returns a reduced input vector $\tilde{\mathbf{v}}_t^i \in \tilde{\mathbf{V}} = \tilde{\mathbf{X}} \times \tilde{\mathbf{U}} \times \tilde{\mathbf{W}} \subset \mathbb{R}^{\tilde{n}}$ (where $\tilde{n} = \tilde{n}_x + \tilde{n}_u + \tilde{n}_w$) with much less components (i.e., $\tilde{n}_x \ll n_x$, $\tilde{n}_u \ll n_u$, $\tilde{n}_w \ll n_w$):

$$\tilde{\mathbf{v}}_t^i = \boldsymbol{\rho}(\mathbf{v}_t^i) = [\rho_x(\mathbf{x}_t), \rho_u(\mathbf{u}_t), \rho_w(\mathbf{w}_t)] = [\tilde{\mathbf{x}}_t, \tilde{\mathbf{u}}_t, \tilde{\mathbf{w}}_t].$$

The reduced model of the MDP is

$$\tilde{\mathbf{x}}_{t+1} = \tilde{\mathbf{f}}(\tilde{\mathbf{v}}_t^i) \tag{4a}$$

$$\tilde{r}_{t+1} = \tilde{g}(\tilde{\mathbf{v}}_t^i). \tag{4b}$$

The optimal value function for the reduced problem can be defined similarly to Eq. 3:

$$\tilde{V}^*(\tilde{\mathbf{x}}) = \max_{\tilde{\mathbf{u}} \in \tilde{\mathbf{U}}} \mathop{\mathbb{E}}_{\tilde{\mathbf{w}}} \left[\tilde{g}(\tilde{\mathbf{v}}^i) + \gamma \tilde{V}^*(\tilde{\mathbf{f}}(\tilde{\mathbf{v}}^i)) \right], \quad \forall \tilde{\mathbf{x}} \in \tilde{\mathbf{X}}.$$

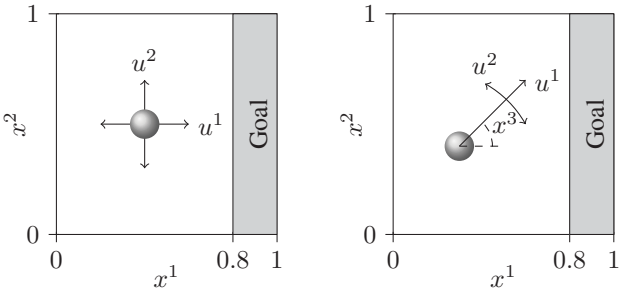
The dimensionality reduction is *lossless* when the solution to the reduced problem is a control policy which is optimal also when back–projected into the original system. It can be easily shown that sufficient condition for lossless reduction is

$$V^*(\mathbf{x}) = \tilde{V}^*(\rho_x(\mathbf{x})), \quad \forall \mathbf{x} \in \mathbf{X}. \tag{5}$$

In this paper, we restrict our attention to a special case of dimensionality reduction: the *variable selection* problem (VSP). In a VSP, the variables of the reduced problem are a subset of the original variables, obtained by eliminating the most irrelevant ones. In particular, a lossless variable selection for an MDP will remove all the variables that do not affect the computation of the optimal policy, or, according to the sufficient condition in Eq. 5, the estimation of the optimal value function. In order to better explain how the FSP can be defined for an MDP, some simple examples are presented.

Example 1: Consider the continuous gridworld depicted in Fig. 1(a). This problem can be modeled as an MDP with two state variables $x^1, x^2 \in [0, 1]$ that define the position of the agent and two action variables $u^1, u^2 \in [-0.1, 0.1]$ that allow the agent to move along the x^1–axis and x^2–axis, respectively. The (deterministic) state transition equations are

$$\begin{aligned} x_{t+1}^1 &= \max(0, \min(1, x_t^1 + u_t^1)) \\ x_{t+1}^2 &= \max(0, \min(1, x_t^2 + u_t^2)). \end{aligned}$$



(a) Example 1 (b) Example 2

Fig. 1. Continuous gridworlds

The task is episodic and it ends when the agent reaches the gray area on the right. The reward function is defined as

$$r_{t+1} = \begin{cases} 0 & x_t^1 + u_t^1 > 0.8 \\ -1 & \text{otherwise.} \end{cases}$$

It is worth noting that the reward function is independent from both x^2 and u^2. Actually, it is easy to show that such variables can be ignored and the problem can be reduced to a one-dimensional problem with only one control.

From this example, we might erroneously conclude that the FSP in an MDP can be restated as an FSP for the reward function. Actually, this is not always the case as it is shown by the following example.

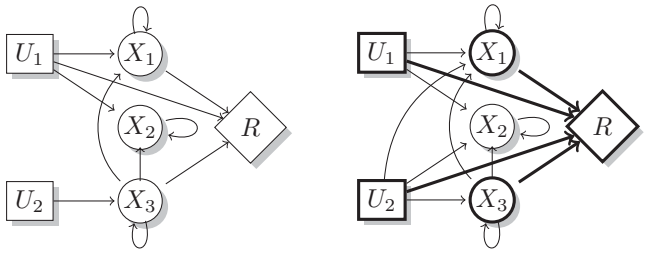
Example 2: Consider the continuous gridworld depicted in Fig. 1(b). This problem differs from the one in Fig. 1(a) in that a third state variable x^3 representing the agent's orientation is introduced. In this problem, control variable $u^1 \in [0, 0.1]$ determines how far the agent moves in the direction she is facing, while $u^2 \in [-\pi/4, \pi/4]$ modifies her orientation. The state transition equations are

$$\begin{aligned} x_{t+1}^1 &= \max(0, \min(1, x_t^1 + u_t^1 \cos(x_t^3))) \\ x_{t+1}^2 &= \max(0, \min(1, x_t^2 + u_t^1 \sin(x_t^3))) \\ x_{t+1}^3 &= x_t^3 + u^2. \end{aligned}$$

The reward function is similar to the one previously defined:

$$r_{t+1} = \begin{cases} 0 & x_t^1 + u_t^1 \cos(x_t^3) > 0.8 \\ -1 & \text{otherwise.} \end{cases}$$

Again, the reward does not depend on the values of variables x^2 and u^2, but, in this case, the problem cannot be reduced by eliminating these variables; in fact, even if control variable u^2 does not directly affect the reward, it determines the value of state variable x^3 on which the reward function is defined. So, only x^2 can be safely discarded.



(a) Graph representation of the adjacency matrix $\mathbf{\Sigma}$.

(b) Graph representation of the transitive closure $\mathbf{\Sigma}^+$.

Fig. 2. Dependency graphs for the continuous gridworld in Fig. 1(b): one-step and transitive closure.

From this second example, it emerges that we can eliminate a (state, control, or disturbance) variable only when it does not affect, either directly or indirectly through its effects on other required state variables, the estimation of the reward function. So, for each state variable that is directly needed to compute the reward function, we need to add to the currently selected variables the ones required to compute its transition model. Such process is recursively repeated on newly selected state variables until no further enlargement occurs. In the following, we formalize the FSP for MDPs and we discuss some properties.

To specify a subset of input variables, we introduce the binary vector $\boldsymbol{\sigma} = [\boldsymbol{\sigma}_{\mathcal{X}}, \boldsymbol{\sigma}_{\mathcal{U}}, \boldsymbol{\sigma}_{\mathcal{W}}] = [\sigma^1, \dots, \sigma^n] \in \{0,1\}^n$: σ^j assumes value 0 if the j–th variable is excluded from the subset and 1 if it is present in the subset. Given a variable vector $\mathbf{v}$, the vector of the selected variables according to $\boldsymbol{\sigma}$ is computed as their point–wise vector product: $\boldsymbol{\sigma} \cdot \mathbf{v} = [\sigma^1 v^1, \dots, \sigma^n v^n]$.

Given the reward function $g : \mathbf{V}^i \mapsto R$, we define $\mathbb{F}_R$ as the set of variable subsets of $\mathcal{V}^i$ that allow a lossless estimation (in a function space $\mathcal{F}$) of g:

$$\mathbb{F}_R = \{\boldsymbol{\sigma} \in \{0,1\}^n | \exists \tilde{g} \in \mathcal{F}, g(\mathbf{v}) = \tilde{g}(\boldsymbol{\sigma} \cdot \mathbf{v}), \forall \mathbf{v} \in \mathbf{V}^i\}.$$

Similarly, we define $\mathbb{F}_{X^j}$ as the set of variable subsets for a lossless estimation of state transition function $f^j : \mathbf{V}^i \mapsto X^j$:

$$\mathbb{F}_{X^j} = \{\boldsymbol{\sigma} \in \{0,1\}^n | \exists \tilde{f}^j \in \mathcal{F}, f^j(\mathbf{v}) = \tilde{f}^j(\boldsymbol{\sigma} \cdot \mathbf{v}), \forall \mathbf{v} \in \mathbf{V}^i\}.$$

Finally, $\mathbb{F}$ is the set of $(n+1)$–by–$(n+1)$ binary matrices $\mathbf{\Sigma}$ whose last column is a vector of zeros, and the other columns are filled as follows: the first n_x rows are vectors representing lossless variable subsets for the state variables, the following $n_u + n_w$ rows are zero vectors and the last row is the vector of the lossless variable subset for the reward variable:

$$\mathbb{F} = \left\{ \begin{bmatrix} \boldsymbol{\sigma}_{X^1} & 0 \\ \vdots & \vdots \\ \boldsymbol{\sigma}_{X^{n_x}} & 0 \\ \mathbf{0} & 0 \\ \vdots & \vdots \\ \mathbf{0} & 0 \\ \boldsymbol{\sigma}_R & 0 \end{bmatrix} | \boldsymbol{\sigma}_R \in \mathbb{F}_R, \boldsymbol{\sigma}_{X^i} \in \mathbb{F}_{X^i}, i = 1, \dots, n_x \right\}.$$

Any matrix $\mathbf{\Sigma} \in \mathbb{F}$ can be viewed as an adjacency matrix of a directed graph whose nodes represent the $n+1$ variables $[\mathcal{X}, \mathcal{U}, \mathcal{W}, R]$, and an edge between node j and node i ($\mathbf{\Sigma}_{i,j}$) means that the j–th variable at time t is needed to compute the value of the i–th variable at time $t+1$. Since the value of controls and disturbances cannot be predicted, all the vectors corresponding to these variables are set to zero. Since rewards cannot be used to predict other variables, the last column is a zero vector.

In order to identify the connected components of such graph, we need to compute the transitive closure of the adjacency matrix $\mathbf{\Sigma}$:

$$\mathbf{\Sigma}^+ = \bigvee_{k=1,\dots,n+1} \mathbf{\Sigma}^k,$$

which is a Boolean matrix sum of the first n Boolean powers[2] of the adjacency matrix $\mathbf{\Sigma}$; the k–th power of the adjacency matrix $\mathbf{\Sigma}^k$ is recursively defined as follows:

$$\mathbf{\Sigma}^k_{i,j} = \bigvee_{l=1,\dots,n+1} \mathbf{\Sigma}^{k-1}_{i,l} \wedge \mathbf{\Sigma}_{l,j}$$

and represents all paths of length k: the (i,j)–element of $\mathbf{\Sigma}^k$ is equal to 1 if there is a k–step path from j to i in the original graph defined by adjacency matrix $\mathbf{\Sigma}$. The transitive closure $\mathbf{\Sigma}^+$ is again an adjacency matrix which directly connects each pair of nodes that is connected by at least one path in the original graph. We denote with $\mathbb{F}^+$ the set of transitive closures built from any adjacency matrix in the set $\mathbb{F}$: $\mathbb{F}^+ = \{\mathbf{\Sigma}^+ | \mathbf{\Sigma} \in \mathbb{F}\}$. In the context of variable selection for MDPs, the i–th row of any $\mathbf{\Sigma}^+$ in $\mathbb{F}^+$ ($\mathbf{\Sigma}^+_{i,\cdot}$) represents a variable subset that is sufficient for estimating the evolution of the i–th variable over time. In particular, the first n elements of the last row of the transitive closure matrix ($\mathbf{\Sigma}^+_{n+1,[1:n]}$) represents the set of variables whose values influence in one or more steps the value of the reward. We denote with $\mathbb{F}^* = \{\mathbf{\Sigma}^+_{n+1,[1:n]} | \mathbf{\Sigma}^+ \in \mathbb{F}^+\}$ the set of all these vectors.

To clarify these concepts, in Fig. 2 is shown the graph corresponding to the adjacency matrix $\mathbf{\Sigma}$ (Fig. 2(a)) and its transitive closure $\mathbf{\Sigma}^+$ (Fig. 2(b)) related to the gridworld described in *Example 2*. As we can notice from Fig. 2(a), only state variables X_1, X_3 and control variable U_1 are needed to predict the value of the reward function at each step. On the other hand, as we have previously discussed, since the value of control variable U_2 influences the state variable (X_3) that determines the reward value, U_2 has to be selected for a correct value function estimation. Figure 2(b) shows the transitive closure of the graph in Fig. 2(a), and in particular it puts in evidence (using thick lines) the last row of the matrix which selects state and control variables that (directly or through their influence on other state variables) can determine the rewards.

Theorem 1: For any $\boldsymbol{\sigma} \in \mathbb{F}^*$ there exists a reduced MDP defined by functions $\tilde{\mathbf{f}}$ and $\tilde{g}$ such that

$$V^*(\mathbf{x}) = \tilde{V}^*(\boldsymbol{\sigma}_{\mathcal{X}} \cdot \mathbf{x}), \quad \forall \mathbf{x} \in \mathbf{X}.$$

[2]Since it can be easily shown that $\mathbf{\Sigma}^{n+2} = \mathbf{\Sigma}^{n+1}$, this summation can be limited to the first $n+1$ powers.

Proof: (Sketch) the theorem can be proven by showing that the condition in Eq. 5 holds for such reduced models. The idea of the proof is to rewrite the two sides of Eq. 5 using their recursive definition and showing that the terms have the same value on both sides. ■

Finally, the model minimization problem can be formulated as finding the smallest set of variables that allow a lossless reduction of the original MDP:

$$\boldsymbol{\sigma}^* = \arg\min_{\boldsymbol{\sigma}\in\mathbb{F}^*} \|\boldsymbol{\sigma}\|,$$

where $\|\boldsymbol{\sigma}\|$ is the number of ones (selected variables) in $\boldsymbol{\sigma}$.

IV. Approximate Variable Selection in MDPs

The approach described in the previous section can be applied to MDPs for which the state–transition model is available. Unfortunately, in many large–scale real–world control problems, the analytic dynamics model is often unknown or roughly approximated. Furthermore, even when a model is available, a lossless variable selection is not expected to produce a significantly more compact model; in fact, all the variables that are somehow involved in the estimation of the optimal value function are selected independently from their actual contribution. In this section we describe the *recursive variable selection* (RVS) algorithm, that is an approximate version of the variable selection algorithm previously described.

A. Recursive Variable Selection

Assuming that a model of the system to be controlled is not available, the variable selection process is applied to a dataset of observed one–step system transitions $D = \{\langle \mathbf{v}^i_k, \mathbf{v}^o_k\rangle\}_{k=1}^N$. The RVS algorithm (summarized in Algorithm 1) takes as input such dataset, the output variable V^o that needs to be predicted, and the set of previously selected variables $\mathcal{V}^i_{sel}$. The algorithm starts from the identification of the most suitable subset of input variables to explain the reward variable ($\mathcal{V}^i_R$); this is achieved by invoking RVS with input parameters $V^o = R$ and $\mathcal{V}^i_{sel} = \emptyset$. Although any variable selection algorithm can be used in this step, here we consider the *iterative variable selection* (IVS) approach (described in Section IV-B) which is based on the ranking of variables according to their importance (in Section IV-C a method based on regression–tree ensembles is detailed). Given the variable subset for immediate reward, the algorithm is recursively repeated taking each state variable in such subset as the output variable to be explained $V^o_{\mathcal{X}} \in \mathcal{V}^i_R \cap \mathcal{X}$. The process selects new variables until no further state variable needing explanation is added and it terminates by returning the set of selected variables.

Algorithm 1 RVS(D, V^o, $\mathcal{V}^i_{sel}$): Recursive Variable Selection

Input: A dataset $D = \{\langle \mathbf{v}^i_k, \mathbf{v}^o_k\rangle\}_{k=1}^N$, the variable to be explained $V^o \in \mathcal{V}^o$, the set of previously selected variables $\mathcal{V}^i_{sel} \subset \mathcal{V}^i$
Output: $\mathcal{V}^i_{V^o} \subset \mathcal{V}^i$: the set of variables to estimate V^o

```
V^i_{V^o} ← FS(D, V^o)
V^{new}_X ← (V^i_{V^o} \ V^i_{sel}) ∩ X
for all V^o_X ∈ V^{new}_X do
    V^i_{V^o} ← V^i_{V^o} ∪ RVS(D, V^o_X, V^i_{sel} ∪ V^i_{V^o})
end for
return V^i_{V^o}
```

B. Iterative Variable Selection algorithm

As described in the previous section, each invocation of the RVS algorithm requires to select the most relevant variables to explain the specified output variable. Variable selection algorithms adopted for this task must account for both significance and redundancy: this means, in other words, that it is necessary to select only the most relevant variables, while trying to avoid the inclusion of redundant variables, which are cause of unnecessary model complexity. Literature shows a variety of different feature selection methods (for a review, see Guyon and Elisseeff, 2003), whose characteristics change with the feature selection problem being considered. This section provides a description of the iterative variable selection (IVS) algorithm, which, apart from providing good performances from a significance and redundancy point of view, shows good capabilities in dealing with high–dimensional datasets.

The ideal algorithm to be used in the selection of the most relevant variables should account for non-linear dependencies and redundancy between variables, as real–world control problems are usually characterized by non–linear dynamic models with multiple coupled variables. Moreover, it must be computationally efficient, since the number of candidate variables is generally large. To fulfill these requirements, we developed the IVS approach, a model-free, forward-selection algorithm which is summarized in Algorithm 2. Given the output variable to be explained V^o and the set of candidate variables $\mathcal{V}^i$, the IVS algorithm first globally ranks the variables according to a statistical measure of significance that accounts for non–linear dependencies (details are provided in the next section). To account for variable redundancy, only the most significant variable V^* is then added to the set of selected variables $\mathcal{V}_{sel}$, which is used for building a model $\hat{f}$ to explain V^o. The reason behind this choice is that, once a variable is selected, all the variables that are highly correlated with that variable may become useless and the ranking needs to be re–evaluated. So, the algorithm proceeds by repeating the ranking process using as new output variable the residuals of the model built at the previous iteration ($\hat{v}^o_k = v^o_k - \hat{f}(\boldsymbol{\sigma}_{\mathcal{V}_{sel}} \cdot \mathbf{v}^i_k), k = 1, \ldots, N$). The algorithm iterates these operations until the best variable returned by the ranking algorithm is already in the set $\mathcal{V}_{sel}$ or the accuracy of the model built upon the selected variables does not significantly improve. The accuracy is computed as the coefficient of determination R^2 between the value of the output variable V^o and the value $\hat{V}^o$ predicted by the model:

$$R^2(V^o, \hat{V}^o) = 1 - \frac{\sum_{k=1}^{N} (\hat{v}^o_k)^2}{\sum_{k=1}^{N} (v^o_k - \overline{v}^o)^2},$$

Algorithm 2 IVS(D, V^o): Iterative Variable Selection

Input: A dataset $D = \{\langle \mathbf{v}_k^i, \mathbf{v}_k^o \rangle\}_{k=1}^N$, the variable to be explained $V^o \in \mathcal{V}^o$

Output: $\mathcal{V}_{sel} \subset \mathcal{V}^i$: set of variables selected to estimate V^o

Initialize: $\mathcal{V}_{sel} \leftarrow \emptyset$, $\hat{V}^o \leftarrow V^o$, $R^2_{old} \leftarrow 0$

repeat

$V^* \leftarrow \arg\max_{V \in \mathcal{V}^i} \text{VR}(D, \hat{V}^o, V)$

if $V^* \in \mathcal{V}_{sel}$ **then**

return $\mathcal{V}_{sel}$

end if

$\mathcal{V}_{sel} \leftarrow \mathcal{V}_{sel} \cup V^*$

$\hat{f} \leftarrow \text{MB}(D, V^o, \mathcal{V}_{sel})$

$\hat{V}^o \leftarrow V^o - \hat{f}(\mathcal{V}_{sel})$

$\Delta R^2 \leftarrow R^2(D, V^o, \hat{V}^o) - R^2_{old}$

$R^2_{old} \leftarrow R^2(D, V^o, \hat{V}^o)$

until $\Delta R^2 < \epsilon$

return $\mathcal{V}_{sel}$

where $\overline{V}^o$ is the mean of the output values: $\overline{V}^o = \frac{1}{N} \sum_{k=1}^N V_k^o$.

To complete the description of the algorithm, it remains to specify the variable ranking (VR) algorithm and the model building (MB) approach. Although the IVS approach could be coupled with any VR and MB algorithm, in this paper we propose to employ a class of tree–based regression methods, named extremely randomized trees (Extra-Trees) [17], and an Extra-Trees based ranking procedure [13]. Their characteristics and advantages are shown in the next two sections.

C. Model Building: Extremely Randomized Trees

Tree-based methods are non-parametric supervised learning methods that can provide several desirable variables in regression problems, as modeling flexibility, computational efficiency, good accuracy and interpretability. They are all based on the idea of decision tree models, which are tree-like structures representing a cascade of rules leading to numerical values [18]. These structures, composed of decision nodes, branches and leaves, are obtained by partitioning, according to a certain splitting criterion, the set of the input variables into a series of sub-sets, until either the numerical values belonging to each sub-set vary just slightly or only few elements remain. When the splitting process is over, the branches represent the hierarchical structure of the sub-set partitions, while the leaves are the finest sub-sets associated to the terminal branches. Each leaf is finally associated with a numerical value.

In this study we explore Extra-Trees [17], which randomize (totally or partially) both the input variable and the cut-point selection when splitting a node, and create an ensemble of trees to compensate for the randomization, via averaging of the constituent tree outcomes. The Extra-Trees building algorithm grows ensemble of M trees. Nodes are split using the following rule: K alternative cut-directions (input variables) are randomly selected and, for each one, a random cut-point is chosen; a score is then associated to each cut-direction and the one maximizing the score is adopted to split the node. The algorithm stops partitioning a node if its cardinality is smaller than n_{min}, and the node is therefore a leaf. To each leaf a value is assigned, obtained as the average of the outputs associated to the inputs that fall in that leaf. The estimates produced by the M trees are finally aggregated with arithmetic average. The rationale behind the approach is that the combined used of randomization and ensemble averaging provide more effective variance reduction than other randomization methods, while minimizing the bias of the final estimate. Extra-Trees are thus characterized by three parameters (i.e. K, n_{min} and M), whose value can be fixed on the basis of empirical evaluations.

The main advantage of Extra-Trees, apart from their computational efficiency and prediction accuracy, is that their building algorithm can be exploited to rank the importance of the n input variables in explaining the output behavior and then identify the most relevant variables among n candidate inputs (variable ranking).

D. Variable Ranking

The built-in variable–ranking approach implemented in Extra-Trees, as proposed in [13] and used in [19], is based on the idea of scoring each input variable by estimating the variance reduction produced anytime that such variable is chosen during the tree building procedure. The score is computed as the percentage of variance reduction achieved by each variable over the M different tree structures composing the ensemble. More precisely, let us consider a regression problem with an output variable V^o, n input variables $\mathcal{V}^i = \{V_1, \ldots, V_n\}$ and a training data-set D, composed of N input–output observations. The score of each input variable $V_i \in \mathcal{V}$ in explaining the output variable V^o given a dataset D can be evaluated as follows

$$VR(D, V^o, V_i) = \frac{\sum_{m=1}^M \sum_{j=1}^{\Omega_m} \delta(\nu_{j,m}, V_i) \cdot \Delta_{var}(\nu_{j,m})}{\sum_{m=1}^M \sum_{\nu_{j,m}=1}^{\Omega_m} \Delta_{var}(\nu_{j,m})} \tag{6}$$

where $\nu_{j,m}$ is the j-th non-terminal node in the m–th tree, Ω_m is the number of non–terminal nodes in the m–th tree, $\delta(\nu_{j,m}, V_i)$ is equal to 1 if V_i is used to split the node $\nu_{j,m}$ (and 0 otherwise), $\Delta_{var}(\nu_{j,m})$ is the variance reduction when splitting the node $\nu_{j,m}$, namely

$$\begin{aligned} \Delta_{var}(\nu_{j,m}) &= |D| \text{var}\{V^o|D\} - \\ &\quad |D_{i,l}| \text{var}\{V^o|D_{i,l}\} - |D_{i,r}| \text{var}\{V^o|D_{i,r}\}, \end{aligned}$$

where the terms $D_{i,l}$ and $D_{i,r}$ are the two subsets obtained by splitting D along the input dimension V_i. Input variables are finally sorted by decreasing values of their importance.

V. Experimental Results

To evaluate the effectiveness of the proposed approach, we describe its application to two real–world control problems from robotics and environmental engineering: balancing of TiltOne, a two-wheeled inverted–pendulum robot, and the management of the Tono Dam. The variables selected by the RVS algorithm are then considered to generate a low–dimensional training dataset which feeds the fitted Q-iteration

(FQI) algorithm [20], a batch–mode model–free RL algorithm. FQI translates the RL problem in a sequence of H supervised learning problems, where H is the length of the optimization horizon. Since the supervised learning problems can be solved using any regression algorithm, we have resorted once again to Extra-Trees (see Section IV-C).

A. TiltOne case study

TiltOne is a two-wheeled balancing robot that can stand in vertical position by actively controlling the speed of its wheels. The robot is $90cm$ tall and weights $20kg$ with batteries with a maximum payload of about $50kg$ when moving at speeds up to about $1.5m/s$. The wheels are $50cm$ in diameter and are actuated by two $150W$ DC motors with a $26:1$ gear reduction. A synchronous belt transmission adds a $4:1$ reduction, resulting in a total reduction of $104:1$ and a maximum continuous torque at the wheels of $18N/m$. The robot is equipped with sensors used to measure a set of variables that can describe the state of the system. The state space of inverted–pendulum balancing tasks is usually made up of three state variables (angle, angular rate, and linear speed) [21]. In these experiments, we have taken into considerations seven state variables: the angle θ of the frame with respect to the vertical position, the angular rate $\dot{\theta}$, the speed of the robot on the horizontal plane $\dot{x}$, the raw accelerometer and gyroscope data, the integral and the derivative of the error (last two variables are the same used by PID controllers). The robot is controlled by setting a single control variable u that determines the wheel speed through the PWM (pulse width modulation) output signal. In the balancing task, the goal of the robot is to reach and keep the vertical position as soon as possible. To define such task we have chosen the following reward function:

$$r_{t+1} = -|\theta_{t+1}|.$$

The dataset used to select the variables is built on the basis of real data collected by the robot at $50Hz$. In order to provide a good coverage of the domain of the measured variables, acquisitions have been performed with the robot starting from the vertical position and controlled by controls taken uniformly at random. The value of the control is updated at a frequency of $10Hz$, so each control is maintained for 5 cycles to better evaluate the effect of a specific control on the robot behavior. When the robot falls (the absolute value of angle θ exceeds 8 degrees) the current acquisition is terminated and a new one is started. Depending on the random controls performed, a single acquisition can last from a fraction of second to about $2s$.

Since delays in sensing and actuation can make the state non–Markovian, we enlarge the set of state variables to include also a history of five past control decisions ($\mathbf{x}_t$ contains $u_{t-1}, \ldots, u_{t-5}$), thus resulting in a total of thirteen input variables (twelve state variables and one control variable). A dataset with a thousand tuples has been given to the RVS algorithm to evaluate which input variables are most relevant for this control problem. The result is the following: the reward r_{t+1} can be accurately predicted by knowing the angle θ_t and the control control u_t. Then, the algorithm evaluates which variables are required to approximate the state–transition model for variable θ, finding that θ_{t+1} depends on its value at the previous time step (θ_t), on the control control u_t, and, interestingly, on the control decision taken at time $t-1$ (u_{t-1}). Since the next state value of u_{t-1} is equal to u_t, no further variables need to be selected and the algorithm terminates.

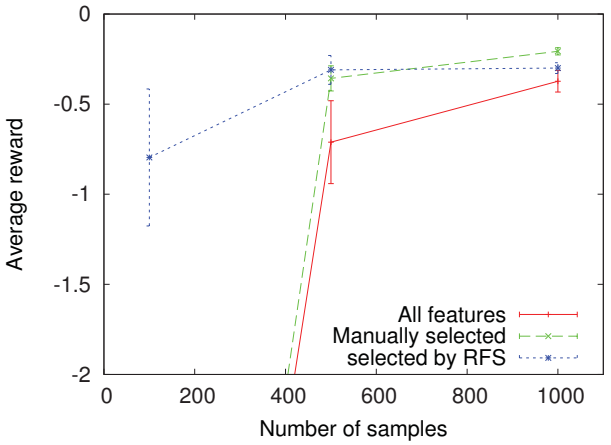


Fig. 3. Comparison of average rewards (along with error bars for the 95% confidence interval) earned from control policies learned by FQI using three different sets of state variables. The performances are plotted as a function of the number of samples in the training dataset.

To evaluate the effectiveness of this reduced model, we study the performances of the policies learned by the FQI algorithm (where $H = 10$ and Extra-Trees parameters are: $M = 100$ trees, K equal to the number of input variables and $n_{min} = 10$) using three different sets of input variables: all the 13 variables, the variables usually used to solve inverted pendulum tasks (θ_t, $\dot{\theta}_t$, $\dot{x}_t$, and u_t), and the variables selected by the RVS approach (θ_t, u_{t-1}, and u_t). For each policy, 5 testing episodes are performed and their results are averaged; each episode starts with the robot leaning on a support which determines an inclination $\omega = 2^o$ and stops after $5s$ (250 cycles). If the robot falls ($|\omega| > 8^o$) before the end of the run its reward is set to -8 for all the remaining cycles. The results are compared in Fig. 3, where the performance is evaluated as the average reward per step. As we can notice, when enough samples are provided, FQI is able to learn pretty good policies with any of the three sets of variables. Nonetheless, when the number of samples is quite small the advantage of using the reduced variable space identified by the RVS algorithm becomes very apparent. In particular, when only 100 tuples are used, FQI is able to balance the robot only by reducing the problem to the few variables selected by the RVS algorithm, while in the other two cases the robot falls after a few steps (so that average reward is close to -8). Considered the robustness of Extra-Trees in presence of many irrelevant variables, we may expect even stronger benefits when other regression algorithms are employed.

B. Tono Dam

Tono Dam is a small reservoir (12.4 x 10^6 m^3 of gross capacity) being constructed these years at the confluence of Kango and Fukuro rivers (JP). The management of the reservoir, apart from satisfying some water quantity targets (e.g., supply some downstream irrigation districts, feed a small hydropower station and supply industrial and drinking water), must concentrate on water quality, to protect the downstream environment and the irrigated crops during the germination phase. To this purpose, the Japanese legislation suggests indeed to reduce the effect of artificially induced temperature variations by keeping the outflow temperature as closest as possible to the natural inflow temperature. For this reason Tono Dam is being equipped with a Selective Withdrawal Structure (SWS) that allows to exploit the natural temperature stratification of the water body by releasing active storage water at different levels.

As the management problem requires to account for the spatial dynamics of the water quality variables, the system is modeled with the 1D hydrodynamic-ecological model (DYRESM-CAEDYM [22]), whose output is the reward r_{t+1} accounting for the difference between the outflow and inflow water temperature. Since the large model dimensionality does not allow for the design of the reservoir release policy, whose controls u_t^{-3} and u_t^{-13} are the daily release decisions at $-3m$ and $-13m$ of the SWS, the variable selection process is applied to a dataset of observed one-step system transitions D. By combining 100 pseudo–randomly generated scenarios of the control and disturbance variables $\mathbf{u}_t$ and $\mathbf{w}_t$, the model provides the dataset D of 71 variables: 19 state variables (n_x), 50 disturbances (n_w) and 2 controls (n_u).

The RVS algorithm, which relies on the IVS algorithm, employs such dataset to identify the most suitable subset of input variables to explain the reward variable r_{t+1}. It results that the reward depends on 7 variables: the 2 controls u_t^{-3} and u_t^{-13}, 3 state variables (T_t^{sed}, T_t^{-7} and t) and 2 disturbances ($NH4_t^F$ and T_t^K). Given this variable subset for immediate reward, the algorithm is recursively repeated taking each state variable in such subset as the output variable to be explained, thus leading to a final subset of 6 state variables (h_t, t,T_t^{-3}, T_t^{-7}, T_t^{-13}, T_t^{sed}), 2 controls (u_t^{-3} and u_t^{-13}), and 7 disturbances.

As in the previous experiment, we use FQI (with $H = 40$ and the same Extra-Trees parameters as in the previous experiment) to evaluate the effectiveness of the selected variables to solve the specified control problem. The training dataset has been built with 100 simulations using 12 years of meteorological data (from 1995 to 2006), while the validation is performed over the years from 1991 to 1994. As a term of comparison, we have asked to a domain expert to choose the most relevant state variables for this task. He selected 6 state variables, and only two of these differ from those selected by RVS. Nonetheless, the performance of the policy learned by FQI using the variables from RVS (average reward -1.34) is significantly better than the one obtained with the variable suggested by the expert (average reward -1.75).

VI. Conclusion

In this paper we have proposed an offline model-free variable selection procedure for dimensionality reduction in Markov decision processes. The approach has been demonstrated on two real–world control problems: in both cases, the variables selected by the proposed algorithm allow to learn good control policies even when the training dataset is quite small.

Future research will be devoted to apply this approach for building small, computationally efficient emulation models.

References

[1] E. Van Nes and M. Scheffer, "A strategy to improve the contribution of complex simulation models to ecological theory," *Ecological modelling*, vol. 185, no. 2-4, pp. 153–164, 2005.

[2] L. Li, T. Walsh, and M. Littman, "Towards a unified theory of state abstraction for MDPs," in *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006, pp. 531–539.

[3] T. Dean and R. Givan, "Model minimization in Markov decision processes," in *Proceedings of AAAI*, 1997, pp. 106–111.

[4] B. Ravindran and A. Barto, "Model minimization in hierarchical reinforcement learning," *Abstraction, Reformulation, and Approximation*, pp. 196–211, 2002.

[5] R. Givan, T. Dean, and M. Greig, "Equivalence notions and model minimization in Markov decision processes," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 163–223, 2003.

[6] C. Boutilier, R. Dearden, and M. Goldszmidt, "Exploiting structure in policy construction," in *Proceedings of IJCAI*, vol. 14, 1995, pp. 1104–1113.

[7] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, "Efficient solution algorithms for factored MDPs," *Journal of Artificial Intelligence Research*, vol. 19, no. 1, pp. 399–468, 2003.

[8] T. Dean, R. Givan, and K. Kim, "Solving stochastic planning problems with large state and action spaces," in *Proc. Fourth International Conference on Artificial Intelligence Planning Systems*, 1998.

[9] T. Degris, O. Sigaud, and P. Wuillemin, "Learning the structure of factored markov decision processes in reinforcement learning problems," in *Proceedings of ICML*. ACM, 2006, pp. 257–264.

[10] A. Strehl, C. Diuk, and M. Littman, "Efficient structure learning in factored-state MDPs," in *Proceedings of AAAI*, vol. 22, no. 1, 2007, p. 645.

[11] C. Vigorito and A. Barto, "Incremental structure learning in factored mdps with continuous states and actions," *University of Massachusetts Amherst-Department of Computer Science, Tech. Rep*, 2009.

[12] B. Kveton, M. Hauskrecht, and C. Guestrin, "Solving factored MDPs with hybrid state and action variables," *Journal of Artificial Intelligence Research*, vol. 27, no. 1, pp. 153–201, 2006.

[13] L. Wehenkel, *Automatic learning techniques in power systems*. Kluwer Academic Publishers, 1998.

[14] R. E. Bellman, *Dynamic Programming*. Princeton, New Jersey, USA: Princeton University Press, 1957.

[15] M. Puterman, *Markov Decision Problems*. New York: Wiley, 1994.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, B. Book, Ed. Cambridge, MA: MIT Press, 1998.

[17] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[18] L. Breiman, *Classification and regression trees*. Chapman & Hall/CRC, 1984.

[19] R. Fonteneau, L. Wehenkel, and D. Ernst, "Variable selection for dynamic treatment regimes: a reinforcement learning approach," in *Proceedings of EWRL*, 2008.

[20] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, no. 1, pp. 503–556, Apr. 2005.

[21] A. Bonarini, C. Caccia, A. Lazaric, and M. Restelli, "Batch reinforcement learning for controlling a mobile wheeled pendulum robot," in *Proceedings of IFIP AI*, vol. 276. Springer Verlag, 2008, pp. 151–160.

[22] A. Imerito, "Dynamic REservoir Simulation Model: Dyresm Science Manual," Centre for water Research - University of Western Australia, Crawley - Western Australia, CWR Tech. Report, 2007.