Robust Value Function Approximation Using Bilinear Programming

Marek Petrik
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
petrik@cs.umass.edu

Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
shlomo@cs.umass.edu

Abstract

Existing value function approximation methods have been successfully used in many applications, but they often lack useful a priori error bounds. We propose *approximate bilinear programming*, a new formulation of value function approximation that provides strong a priori guarantees. In particular, this approach provably finds an approximate value function that minimizes the Bellman residual. Solving a bilinear program optimally is NP-hard, but this is unavoidable because the Bellman-residual minimization itself is NP-hard. We therefore employ and analyze a common approximate algorithm for bilinear programs. The analysis shows that this algorithm offers a *convergent* generalization of approximate policy iteration. Finally, we demonstrate that the proposed approach can consistently minimize the Bellman residual on a simple benchmark problem.

1 Motivation

Solving large Markov Decision Problems (MDPs) is a very useful, but computationally challenging problem addressed widely in the AI literature, particularly in the area of reinforcement learning. It is widely accepted that large MDPs can only be solved approximately. The commonly used approximation methods can be divided into three broad categories: 1) *policy search*, which explores a restricted space of all policies, 2) *approximate dynamic programming*, which searches a restricted space of value functions, and 3) *approximate linear programming*, which approximates the solution using a linear program. While all of these methods have achieved impressive results in many domains, they have significant limitations.

Policy search methods rely on local search in a restricted policy space. The policy may be represented, for example, as a finite-state controller [22] or as a greedy policy with respect to an approximate value function [24]. Policy search methods have achieved impressive results in such domains as Tetris [24] and helicopter control [1]. However, they are notoriously hard to analyze. We are not aware of any theoretical guarantees regarding the quality of the solution.

Approximate dynamic programming (ADP) methods iteratively approximate the value function [4, 20, 23]. They have been extensively analyzed and are the most commonly used methods. However, ADP methods typically do not converge and they only provide weak guarantees of approximation quality. The approximation error bounds are usually expressed in terms of the worst-case approximation of the value function over all policies [4]. In addition, most available bounds are with respect to the L_∞ norm, while the algorithms often minimize the L_2 norm. While there exist some L_2-based bounds [14], they require values that are difficult to obtain.

Approximate linear programming (ALP) uses a linear program to compute the approximate value function in a particular vector space [7]. ALP has been previously used in a wide variety of settings [2, 9, 10]. Although ALP often does not perform as well as ADP, there have been some recent

efforts to close the gap [18]. ALP has better theoretical properties than ADP and policy search. It is guaranteed to converge and return the closest L_1-norm approximation $\tilde{v}$ of the optimal value function v^* up to a multiplicative factor. However, the L_1 norm must be properly weighted to guarantee a small policy loss, and there is no *reliable* method for selecting appropriate weights [7].

To summarize, the existing reinforcement learning techniques often provide good solutions, but typically require significant domain knowledge [20]. The domain knowledge is needed partly because useful a priori error bounds are not available, as mentioned above. Our goal is to develop a more *robust* method that is guaranteed to minimize an actual bound on the policy loss.

We present a new formulation of value function approximation that provably minimizes a bound on the policy loss. Unlike in some other algorithms, the bound in this case does not rely on values that are hard to obtain. The new method unifies policy search and value-function search methods to minimize the L_∞ norm of the Bellman residual, which bounds the policy loss. We start with a description of the framework and notation in Section 2. Then, in Section 3, we describe the proposed Approximate Bilinear Programming (ABP) formulation. A drawback of this formulation is its computational complexity, which may be exponential. We show in Section 4 that this is unavoidable, because minimizing the approximation error bound is in fact NP-hard. Although our focus is on the formulation and its properties, we also discuss some simple algorithms for solving bilinear programs. Section 5 shows that ABP can be seen as an improvement of ALP and Approximate Policy Iteration (API). Section 6 demonstrates the applicability of ABP using a common reinforcement learning benchmark problem. A complete discussion of sampling strategies–an essential component for achieving robustness–is beyond the scope of this paper, but the issue is briefly discussed in Section 6. Complete proofs of the theorems can be found in [19].

2 Solving MDPs using ALP

In this section, we formally define MDPs, their ALP formulation, and the approximation errors involved. These notions serve as a basis for developing the ABP formulation.

A *Markov Decision Process* is a tuple $(\mathcal{S}, \mathcal{A}, P, r, \alpha)$, where $\mathcal{S}$ is the *finite* set of states, $\mathcal{A}$ is the *finite* set of actions. $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ is the transition function, where $P(s', s, a)$ represents the probability of transiting to state s' from state s, given action a. The function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, and $\alpha : \mathcal{S} \mapsto [0, 1]$ is the initial state distribution. The objective is to maximize the infinite-horizon *discounted cumulative reward*. To shorten the notation, we assume an arbitrary ordering of the states: $s_1, s_2, \ldots, s_n$. Then, P_a and r_a are used to denote the probabilistic transition matrix and reward for action a.

The solution of an MDP is a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ from a set of possible policies Π, such that for all $s \in \mathcal{S}$, $\sum_{a \in \mathcal{A}} \pi(s, a) = 1$. We assume that the policies may be stochastic, but stationary [21]. A policy is deterministic when $\pi(s, a) \in \{0, 1\}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The transition and reward functions for a given policy are denoted by P_π and r_π. The value function update for a policy π is denoted by L_π, and the Bellman operator is denoted by L. That is:

$$L_\pi v = P_\pi v + r_\pi \qquad\qquad Lv = \max_{\pi \in \Pi} L_\pi v.$$

The optimal value function, denoted v^*, satisfies $v^* = Lv^*$. We focus on *linear value function approximation* for discounted infinite-horizon problems. In linear value function approximation, the value function is represented as a linear combination of *nonlinear basis functions (vectors)*. For each state s, we define a row-vector $\phi(s)$ of features. The rows of the basis matrix M correspond to $\phi(s)$, and the approximation space is generated by the columns of the matrix. That is, the basis matrix M, and the value function v are represented as:

$$M = \begin{pmatrix} - & \phi(s_1) & - \\ - & \phi(s_2) & - \\ & \vdots & \end{pmatrix} \qquad v = Mx.$$

Definition 1. A value function, v, is *representable* if $v \in \mathcal{M} \subseteq \mathbb{R}^{|\mathcal{S}|}$, where $\mathcal{M} = \text{colspan}\,(M)$, and is *transitive-feasible* when $v \geq Lv$. We denote the set of transitive-feasible value functions as: $\mathcal{K} = \{v \in \mathbb{R}^{|\mathcal{S}|} \,|\, v \geq Lv\}$.

Notice that the optimal value function v^* is transitive-feasible, and $\mathcal{M}$ is a linear space. Also, all the inequalities are element-wise.

Because the new formulation is related to ALP, we introduce it first. It is well known that an infinite horizon discounted MDP problem may be formulated in terms of solving the following linear program:

$$\begin{array}{ll} \underset{v}{\text{minimize}} & \sum_{s\in\mathcal{S}} c(s)v(s) \\ \text{s.t.} & v(s) - \gamma \sum_{s'\in\mathcal{S}} P(s',s,a)v(s') \geq r(s,a) \quad \forall (s,a) \in (\mathcal{S},\mathcal{A}) \end{array} \tag{1}$$

We use A as a shorthand notation for the constraint matrix and b for the right-hand side. The value c represents a distribution over the states, usually a uniform one. That is, $\sum_{s\in\mathcal{S}} c(s) = 1$. The linear program in Eq. (1) is often too large to be solved precisely, so it is approximated to get an *approximate linear program* by assuming that $v \in \mathcal{M}$ [8], as follows:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c^{\mathsf{T}} v \\ \text{s.t.} & Av \geq b \\ & v \in \mathcal{M} \end{array} \tag{2}$$

The constraint $v \in \mathcal{M}$ denotes the approximation. To actually solve this linear program, the value function is represented as $v = Mx$. In the remainder of the paper, we assume that $\mathbf{1} \in \mathcal{M}$ to guarantee the feasibility of the ALP, where $\mathbf{1}$ is a vector of all ones. The optimal solution of the ALP, $\tilde{v}$, satisfies that $\tilde{v} \geq v^*$. Then, the objective of Eq. (2) represents the minimization of $\|\tilde{v} - v^*\|_{1,c}$, where $\|\cdot\|_{1,c}$ is a c-weighted L_1 norm [7].

The ultimate goal of the optimization is not to obtain a good value function $\tilde{v}$, but a good policy. The quality of the policy, typically chosen to be greedy with respect to $\tilde{v}$, depends non-trivially on the approximate value function. The ABP formulation will minimize policy loss by minimizing $\|L\tilde{v} - \tilde{v}\|_\infty$, which bounds the policy loss as follows.

Theorem 2 (e.g. [25])**.** *Let $\tilde{v}$ be an arbitrary value function, and let $\hat{v}$ be the value of the greedy policy with respect to $\tilde{v}$. Then:*

$$\|v^* - \hat{v}\|_\infty \leq \frac{2}{1-\gamma} \|L\tilde{v} - \tilde{v}\|_\infty,$$

In addition, if $\tilde{v} \geq L\tilde{v}$, the policy loss is smallest for the greedy policy.

Policies, like value functions, can be represented as vectors. Assume an arbitrary ordering of the state-action pairs, such that $o(s,a) \mapsto \mathbb{N}$ maps a state and an action to its position. The policies are represented as $\theta \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$, and we use the shorthand notation $\theta(s,a) = \theta(o(s,a))$.

Remark 3. The corresponding π and θ are denoted as π^θ and θ^π and satisfy:

$$\pi^\theta(s,a) = \theta^\pi(s,a).$$

We will also consider approximations of the policies in the policy-space, generated by columns of a matrix N. A policy is *representable* when $\pi \in \mathcal{N}$, where $\mathcal{N} = \operatorname{colspan}(N)$.

3 Approximate Bilinear Programs

This section shows how to formulate $\min_{v\in\mathcal{M}} \|Lv - v\|_\infty$ as a separable bilinear program. Bilinear programs are a generalization of linear programs with an additional bilinear term in the objective function. A separable bilinear program consists of two linear programs with independent constraints and are fairly easy to solve and analyze.

Definition 4 (Separable Bilinear Program)**.** A *separable* bilinear program in the normal form is defined as follows:

$$\begin{array}{ll} \underset{w,x\,|\,y,z}{\text{minimize}} & f(w,x,y,z) = s_1^{\mathsf{T}} w + r_1^{\mathsf{T}} x + x^{\mathsf{T}} C y + r_2^{\mathsf{T}} y + s_2^{\mathsf{T}} z \\ \text{s.t.} & A_1 x + B_1 w = b_1 \quad A_2 y + B_2 z = b_2 \\ & w, x \geq \mathbf{0} \qquad\qquad y, z \geq \mathbf{0} \end{array} \tag{3}$$

We separate the variables using a vertical line and the constraints using different columns to emphasize the separable nature of the bilinear program.

In this paper, we only use *separable* bilinear programs and refer to them simply as bilinear programs.

An approximate bilinear program can now be formulated as follows.

$$\begin{array}{lll} \underset{\theta \mid \lambda,\lambda',v}{\text{minimize}} & \theta^{\mathsf{T}}\lambda + \lambda' & \\ \text{s.t.} & B\theta = \mathbf{1} & z = Av - b \\ & \theta \geq \mathbf{0} & z \geq \mathbf{0} \\ & & \lambda + \lambda'\mathbf{1} \geq z \\ & & \lambda \geq \mathbf{0} \\ & \theta \in \mathcal{N} & v \in \mathcal{M} \end{array} \tag{4}$$

All variables are vectors except λ', which is a scalar. The symbol z is only used to simplify the notation and does not need to represent an optimization variable. The variable v is defined for each state and represents the value function. Matrix A represents constraints that are identical to the constraints in Eq. (2). The variables λ correspond to all state-action pairs. These variables represent the Bellman residuals that are being minimized. The variables θ are defined for all state-action pairs and represent policies in Remark 3. The matrix B represents the following constraints:

$$\sum_{a \in \mathcal{A}} \theta(s,a) = 1 \quad \forall s \in \mathcal{S}.$$

As with approximate linear programs, we initially assume that all the constraints on z are used. In realistic settings, however, the constraints would be sampled or somehow reduced. We defer the discussion of this issue until Section 6. Note that the constraints in our formulation correspond to elements of z and θ. Thus when constraints are omitted, also the corresponding elements of z and θ are omitted.

To simplify the notation, the value function approximation in this problem is denoted only implicitly by $v \in \mathcal{M}$, and the policy approximation is denoted by $\theta \in \mathcal{N}$. In an actual implementation, the optimization variables would be x, y using the relationships $v = Mx$ and $\theta = Ny$. We do not assume any approximation of the policy space, unless mentioned otherwise. We also use v or θ to refer to partial solutions of Eq. (4) with the other variables chosen appropriately to achieve feasibility.

The ABP formulation is closely related to approximate linear programs, and we discuss the connection in Section 5. We first analyze the properties of the optimal solutions of the bilinear program and then show and discuss the solution methods in Section 4. The following theorem states the main property of the bilinear formulation.

Theorem 5. *b Let $(\tilde{\theta}, \tilde{v}, \tilde{\lambda}, \tilde{\lambda}')$ be an optimal solution of Eq. (4) and assume that $\mathbf{1} \in \mathcal{M}$. Then:*

$$\tilde{\theta}^{\mathsf{T}}\tilde{\lambda} + \tilde{\lambda}' = \|L\tilde{v} - \tilde{v}\|_\infty \leq \min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty \leq 2 \min_{v \in \mathcal{M}} \|Lv - v\|_\infty \leq 2(1+\gamma) \min_{v \in \mathcal{M}} \|v - v^*\|_\infty.$$

In addition, $\pi^{\tilde{\theta}}$ minimizes the Bellman residual with regard to $\tilde{v}$, and its value function $\hat{v}$ satisfies:

$$\|\hat{v} - v^*\|_\infty \leq \frac{2}{1-\gamma} \min_{v \in \mathcal{M}} \|Lv - v\|_\infty.$$

The proof of the theorem can be found in [19]. It is important to note that, as Theorem 5 states, the ABP approach is equivalent to a minimization over *all* representable value functions, not only the transitive-feasible ones. Notice also the missing coefficient 2 (2 instead of 4) in the last equation of Theorem 5. This follows by subtracting a constant vector $\mathbf{1}$ from $\tilde{v}$ to balance the lower bounds on the Bellman residual error with the upper ones. This modified approximate value function will have $1/2$ of the original Bellman residual but an identical greedy policy. Finally, note that whenever $v^* \in \mathcal{M}$, both ABP and ALP will return the optimal value function.

The ABP solution minimizes the L_∞ norm of the Bellman residual due to: 1) the correspondence between θ and the policies, and 2) the dual representation with respect to variables λ and λ'. The theorem then follows using techniques similar to those used for approximate linear programs [7].

Algorithm 1: Iterative algorithm for solving Eq. (3)

$(x_0, w_0) \leftarrow$ random ;
$(y_0, z_0) \leftarrow \arg\min_{y,z} f(w_0, x_0, y, z)$;
$i \leftarrow 1$;
while $y_{i-1} \neq y_i$ *or* $x_{i-1} \neq x_i$ **do**
  $(y_i, z_i) \leftarrow \arg\min_{\{y,z \mid A_2 y + B_2 z = b_2 \; y,z \geq \mathbf{0}\}} f(w_{i-1}, x_{i-1}, y, z)$;
  $(x_i, w_i) \leftarrow \arg\min_{\{x,w \mid A_1 x + B_1 w = b_1 \; x,w \geq \mathbf{0}\}} f(w, x, y_i, z_i)$;
  $i \leftarrow i + 1$
return $f(w_i, x_i, y_i, z_i)$

4 Solving Bilinear Programs

In this section we describe simple methods for solving ABPs. We first describe optimal methods, which have exponential complexity, and then discuss some approximation strategies.

Solving a bilinear program is an NP-complete problem [3]. The membership in NP follows from the finite number of basic feasible solutions of the individual linear programs, each of which can be checked in polynomial time. The NP-hardness is shown by a reduction from the SAT problem [3]. The NP-completeness of ABP compares unfavorably with the polynomial complexity of ALP. However, most other ADP algorithms are not guaranteed to converge to a solution in finite time.

The following theorem shows that the computational complexity of the ABP formulation is asymptotically the same as the complexity of the problem it solves.

Theorem 6. *b Determining* $\min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty < \epsilon$ *is NP-complete for the full constraint representation,* $0 < \gamma < 1$*, and a given* $\epsilon > 0$*. In addition, the problem remains NP-complete when* $\mathbf{1} \in \mathcal{M}$*, and therefore* $\min_{v \in \mathcal{M}} \|Lv - v\|_\infty < \epsilon$ *is also NP-complete.*

As the theorem states, the value function approximation does not become computationally simpler even when $\mathbf{1} \in \mathcal{M}$ – a universal assumption in the paper. Notice that ALP can determine whether $\min_{v \in \mathcal{K} \cap \mathcal{M}} \|Lv - v\|_\infty = 0$ in polynomial time.

The proof of Theorem 6 is based on a reduction from SAT and can be found in [19]. The policy in the reduction determines the true literal in each clause, and the approximate value function corresponds to the truth value of the literals. The approximation basis forces literals that share the same variable to have consistent values.

Bilinear programs are non-convex and are typically solved using global optimization techniques. The common solution methods are based on concave cuts [11] or branch-and-bound [6]. In ABP settings with a small number of features, the successive approximation algorithm [17] may be applied efficiently. We are, however, not aware of commercial solvers available for solving bilinear programs. Bilinear programs can be formulated as concave quadratic minimization problems [11], or mixed integer linear programs [11, 16], for which there are numerous commercial solvers available. Because we are interested in solving very large bilinear programs, we describe simple approximate algorithms next. Optimal scalable methods are beyond the scope of this paper.

The most common approximate method for solving bilinear programs is shown in Algorithm 1. It is designed for the general formulation shown in Eq. (3), where $f(w, x, y, z)$ represents the objective function. The minimizations in the algorithm are linear programs which can be easily solved. Interestingly, as we will show in Section 5, Algorithm 1 applied to ABP generalizes a version of API.

While Algorithm 1 is not guaranteed to find an optimal solution, its empirical performance is often remarkably good [13]. Its basic properties are summarized by the following proposition.

Proposition 7 (e.g. [3])**.** *Algorithm 1 is guaranteed to converge, assuming that the linear program solutions are in a vertex of the optimality simplex. In addition, the global optimum is a fixed point of the algorithm, and the objective value monotonically improves during execution.*

The proof is based on the finite count of the basic feasible solutions of the individual linear programs. Because the objective function does not increase in any iteration, the algorithm will eventually converge.

In the context of MDPs, Algorithm 1 can be further refined. For example, the constraint $v \in \mathcal{M}$ in Eq. (4) serves mostly to simplify the bilinear program and a value function that violates it may still be acceptable. The following proposition motivates the construction of a new value function from two transitive-feasible value functions.

Proposition 8. *Let $\tilde{v}_1$ and $\tilde{v}_2$ be feasible value functions in Eq. (4). Then the value function $\tilde{v}(s) = \min\{\tilde{v}_1(s), \tilde{v}_2(s)\}$ is also feasible in Eq. (4). Therefore $\tilde{v} \geq v^*$ and $\|v^* - \tilde{v}\|_\infty \leq \min\{\|v^* - \tilde{v}_1\|_\infty, \|v^* - \tilde{v}_2\|_\infty\}$.*

The proof of the proposition is based on Jensen's inequality and can be found in [19].

Proposition 8 can be used to extend Algorithm 1 when solving ABPs. One option is to take the state-wise minimum of values from multiple random executions of Algorithm 1, which preserves the transitive feasibility of the value function. However, the increasing number of value functions used to obtain $\tilde{v}$ also increases the potential sampling error.

5 Relationship to ALP and API

In this section, we describe the important connections between ABP and the two closely related ADP methods: ALP, and API with L_∞ minimization. Both of these methods are commonly used, for example to solve factored MDPs [10]. Our analysis sheds light on some of their observed properties and leads to a new *convergent* form of API.

ABP addresses some important issues with ALP: 1) ALP provides value function bounds with respect to L_1 norm, which does not guarantee small policy loss, 2) ALP's solution quality depends significantly on the heuristically-chosen objective function c in Eq. (2) [7], and 3) incomplete constraint samples in ALP easily lead to unbounded linear programs. The drawback of using ABP, however, is the higher computational complexity.

Both the first and the second issues in ALP can be addressed by choosing the right objective function [7]. Because this objective function depends on the optimal ALP solution, it cannot be practically computed. Instead, various heuristics are usually used. The heuristic objective functions may lead to significant improvements in specific domains, but they do not provide any guarantees. ABP, on the other hand, has no such parameters that require adjustments.

The third issue arises when the constraints of an ALP need to be sampled in some large domains. The ALP may become unbounded with incomplete samples because its objective value is defined using the L_1 norm on the states, and the constraints are defined using the L_∞ norm of the Bellman residual. In ABP, the Bellman residual is used in both the constraints and objective function. The objective function of ABP is then bounded below by 0 for an arbitrarily small number of samples.

ABP can also improve on API with L_∞ minimization (L_∞-API for short), which is a leading method for solving factored MDPs [10]. Minimizing the L_∞ approximation error is theoretically preferable, since it is compatible with the existing bounds on policy loss [10]. In contrast, few practical bounds exist for API with the L_2 norm minimization [14], such as LSPI [12].

L_∞-API is shown in Algorithm 2, where $f(\pi)$ is calculated using the following program:

$$\begin{array}{ll} \underset{\phi, v}{\text{minimize}} & \phi \\ \text{s.t.} & (I - \gamma P_\pi)v + \mathbf{1}\phi \geq r_\pi \\ & -(I - \gamma P_\pi)v + \mathbf{1}\phi \geq -r_\pi \\ & v \in \mathcal{M} \end{array} \tag{5}$$

Here I denotes the identity matrix. We are not aware of a convergence or a divergence proof of L_∞-API, and this analysis is beyond the scope of this paper.

Algorithm 2: Approximate policy iteration, where $f(\pi)$ denotes a custom value function approximation for the policy π.

$\pi_0, k \leftarrow$ rand, 1 ;
while $\pi_k \neq \pi_{k-1}$ **do**
  $\tilde{v}_k \leftarrow f(\pi_{k-1})$;
  $\pi_k(s) \leftarrow \arg\max_{a \in \mathcal{A}} r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s', s, a) \tilde{v}_k(s) \quad \forall s \in \mathcal{S}$;
  $k \leftarrow k + 1$

We propose *Optimistic Approximate Policy Iteration* (OAPI), a modification of API. OAPI is shown in Algorithm 2, where $f(\pi)$ is calculated using the following program:

$$\begin{array}{ll} \underset{\phi, v}{\text{minimize}} & \phi \\ \text{s.t.} & Av \geq b \quad (\equiv (I - \gamma P_\pi)v \geq r_\pi \;\; \forall \pi \in \Pi) \\ & -(I - \gamma P_\pi)v + \mathbf{1}\phi \geq -r_\pi \\ & v \in \mathcal{M} \end{array} \tag{6}$$

In fact, OAPI corresponds to Algorithm 1 applied to ABP because Eq. (6) corresponds to Eq. (4) with fixed θ. Then, using Proposition 7, we get the following corollary.

Corollary 9. *Optimistic approximate policy iteration converges in finite time. In addition, the Bellman residual of the generated value functions monotonically decreases.*

OAPI differs from L_∞-API in two ways: 1) OAPI constrains the Bellman residuals by 0 from below and by ϕ from above, and then it minimizes ϕ. L_∞-API constrains the Bellman residuals by ϕ from both above and below. 2) OAPI, like API, uses only the current policy for the upper bound on the Bellman residual, but uses *all* the policies for the lower bound on the Bellman residual.

L_∞-API cannot return an approximate value function that has a lower Bellman residual than ABP, given the optimality of ABP described in Theorem 5. However, even OAPI, an approximate ABP algorithm, performs comparably to L_∞-API, as the following theorem states.

Theorem 10. *b Assume that L_∞-API converges to a policy π and a value function v that both satisfy: $\phi = \|v - L_\pi v\|_\infty = \|v - Lv\|_\infty$. Then $\tilde{v} = v + \frac{\phi}{1-\gamma}\mathbf{1}$ is feasible in Eq. (4), and it is a fixed point of OAPI. In addition, the greedy policies with respect to $\tilde{v}$ and v are identical.*

The proof is based on two facts. First, $\tilde{v}$ is feasible with respect to the constraints in Eq. (4). The Bellman residual changes for all the policies identically, since a constant vector is added. Second, because L_π is greedy with respect to $\tilde{v}$, we have that $\tilde{v} \geq L_\pi \tilde{v} \geq L\tilde{v}$. The value function $\tilde{v}$ is therefore transitive-feasible. The full proof can be found in [19].

To summarize, OAPI guarantees convergence, while matching the performance of L_∞-API. The convergence of OAPI is achieved because given a non-negative Bellman residual, the greedy policy also minimizes the Bellman residual. Because OAPI ensures that the Bellman residual is always non-negative, it can progressively reduce it. In comparison, the greedy policy in L_∞-API does not minimize the Bellman residual, and therefore L_∞-API does not always reduce it. Theorem 10 also explains why API provides better solutions than ALP, as observed in [10]. From the discussion above, ALP can be seen as an L_1-norm approximation of a single iteration of OAPI. L_∞-API, on the other hand, performs many such ALP-like iterations.

6 Empirical Evaluation

As we showed in Theorem 10, even OAPI, the very simple approximate algorithm for ABP, can perform as well as existing state-of-the art methods on factored MDPs. However, a deeper understanding of the formulation and potential solution methods will be necessary in order to determine the full practical impact of the proposed methods. In this section, we validate the approach by applying it to the mountain car problem, a simple reinforcement learning benchmark problem.

We have so far considered that all the constraints involving z are present in the ABP in Eq. (4). Because the constraints correspond to all state-action pairs, it is often impractical to even enumerate

(a) L_∞ error of the Bellman residual

Features	100	144
OAPI	0.21 (0.23)	0.13 (0.1)
ALP	13. (13.)	3.6 (4.3)
LSPI	9. (14.)	3.9 (7.7)
API	0.46 (0.08)	0.86 (1.18)

(b) L_2 error of the Bellman residual

Features	100	144
OAPI	0.2 (0.3)	0.1 (1.9)
ALP	9.5 (18.)	0.3 (0.4)
LSPI	1.2 (1.5)	0.9 (0.1)
API	0.04 (0.01)	0.08 (0.08)

Table 1: Bellman residual of the final value function. The values are averages over 5 executions, with the standard deviations shown in parentheses.

them. This issue can be addressed in at least two ways. First, a small randomly-selected subset of the constraints can be used in the ABP, a common approach in ALP [9, 5]. The ALP sampling bounds can be easily extended to ABP. Second, the structure of the MDP can be used to reduce the number of constraints. Such a reduction is possible, for example, in factored MDPs with L_∞-API and ALP [10], and can be easily extended to OAPI and ABP.

In the mountain-car benchmark, an underpowered car needs to climb a hill [23]. To do so, it first needs to back up to an opposite hill to gain sufficient momentum. The car receives a reward of 1 when it climbs the hill. In the experiments we used a discount factor $\gamma = 0.99$.

The experiments are designed to determine whether OAPI reliably minimizes the Bellman residual in comparison with API and ALP. We use a uniformly-spaced linear spline to approximate the value function. The constraints were based on 200 uniformly sampled states with all 3 actions per state. We evaluated the methods with the number of the approximation features 100 and 144, which corresponds to the number of linear segments.

The results of ABP (in particular OAPI), ALP, API with L_2 minimization, and LSPI are depicted in Table 1. The results are shown for both L_∞ norm and uniformly-weighted L_2 norm. The runtimes of all these methods are comparable, with ALP being the fastest. Since API (LSPI) is not guaranteed to converge, we ran it for at most 20 iterations, which was an upper bound on the number of iterations of OAPI. The results demonstrate that ABP minimizes the L_∞ Bellman residual much more consistently than the other methods. Note, however, that all the considered algorithms would perform significantly better given a finer approximation.

7 Conclusion and Future Work

We proposed and analyzed approximate bilinear programming, a new value-function approximation method, which provably minimizes the L_∞ Bellman residual. ABP returns the *optimal* approximate value function with respect to the Bellman residual bounds, despite the formulation with regard to transitive-feasible value functions. We also showed that there is no asymptotically simpler formulation, since finding the closest value function and solving a bilinear program are both NP-complete problems. Finally, the formulation leads to the development of OAPI, a new convergent form of API which monotonically improves the objective value function.

While we only discussed approximate solutions of the ABP, a deeper study of bilinear solvers may render optimal solution methods feasible. ABPs have a small number of essential variables (that determine the value function) and a large number of constraints, which can be leveraged by the solvers [15]. The L_∞ error bound provides good theoretical guarantees, but it may be too conservative in practice. A similar formulation based on L_2 norm minimization may be more practical.

We believe that the proposed formulation will help to deepen the understanding of value function approximation and the characteristics of existing solution methods, and potentially lead to the development of more robust and widely-applicable reinforcement learning algorithms.

Acknowledgements

This work was supported by the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0171. We also thank the anonymous reviewers for their useful comments.

References

[1] Pieter Abbeel, Varun Ganapathi, and Andrew Y. Ng. Learning vehicular dynamics, with application to modeling helicopters. In *Advances in Neural Information Processing Systems*, pages 1–8, 2006.

[2] Daniel Adelman. A price-directed approach to stochastic inventory/routing. *Operations Research*, 52:499–514, 2004.

[3] Kristin P. Bennett and O. L. Mangasarian. Bilinear separation of two sets in n-space. Technical report, Computer Science Department, University of Wisconsin, 1992.

[4] Dimitri P. Bertsekas and Sergey Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, LIDS, 1997.

[5] Guiuseppe Calafiore and M.C. Campi. Uncertain convex programs: Randomized solutions and confidence levels. *Mathematical Programming, Series A*, 102:25–46, 2005.

[6] Alberto Carpara and Michele Monaci. Bidimensional packing by bilinear programming. *Mathematical Programming Series A*, 118:75–108, 2009.

[7] Daniela P. de Farias. *The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application*. PhD thesis, Stanford University, 2002.

[8] Daniela P. de Farias and Ben Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51:850–856, 2003.

[9] Daniela Pucci de Farias and Benjamin Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, 2004.

[10] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[11] Reiner Horst and Hoang Tuy. *Global optimization: Deterministic approaches*. Springer, 1996.

[12] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[13] O. L. Mangasarian. The linear complementarity problem as a separable bilinear program. *Journal of Global Optimization*, 12:1–7, 1995.

[14] Remi Munos. Error bounds for approximate policy iteration. In *International Conference on Machine Learning*, pages 560–567, 2003.

[15] Marek Petrik and Shlomo Zilberstein. Anytime coordination using separable bilinear programs. In *Conference on Artificial Intelligence*, pages 750–755, 2007.

[16] Marek Petrik and Shlomo Zilberstein. Average reward decentralized Markov decision processes. In *International Joint Conference on Artificial Intelligence*, pages 1997–2002, 2007.

[17] Marek Petrik and Shlomo Zilberstein. A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research*, 35:235–274, 2009.

[18] Marek Petrik and Shlomo Zilberstein. Constraint relaxation in approximate linear programs. In *International Conference on Machine Learning*, pages 809–816, 2009.

[19] Marek Petrik and Shlomo Zilberstein. Robust value function approximation using bilinear programming. Technical Report UM-CS-2009-052, Department of Computer Science, University of Massachusetts Amherst, 2009.

[20] Warren B. Powell. *Approximate Dynamic Programming*. Wiley-Interscience, 2007.

[21] Martin L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 2005.

[22] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[23] Richard S. Sutton and Andrew Barto. *Reinforcement learning*. MIT Press, 1998.

[24] Istvan Szita and Andras Lorincz. Learning Tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941, 2006.

[25] Ronald J. Williams and Leemon C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. In *Yale Workshop on Adaptive and Learning Systems*, 1994.