On Policy Iteration as a Newton's Method and Polynomial Policy Iteration Algorithms

Omid Madani
Department of Computing Science
University of Alberta
Edmonton, AL
Canada T6G 2E8
madani@cs.ualberta.ca

Abstract

Policy iteration is a a popular technique for solving Markov decision processes (MDPs). It is easy to describe and implement, and has excellent performance in practice. But not much is known about its complexity. The best upper bound remains exponential, and the best lower bound is a trivial $\Omega(n)$ on the number of iterations, where n is the number of states.

This paper improves the upper bounds to a polynomial for policy iteration on MDP problems with special graph structure. Our analysis is based on the connection between policy iteration and the Newton's method for finding the zero of a convex function. The analysis offers an explanation as to why policy iteration is fast. It also leads to polynomial bounds on several variants of policy iteration for MDPs for which the linear programming formulation requires at most two variables per inequality (MDP(2)). The MDP(2) class includes deterministic MDPs under discounted and average reward criteria. The bounds on the run times include $O(mn^2 \log m \log W)$ on MDP(2) and $O(mn^2 \log m)$ for deterministic MDPs, where m denotes the number of actions and W denotes the magnitude of the largest number in the problem description.

Keywords: Markov decision processes, computational complexity, algorithm design and analysis, dynamic programming, local search

1 Introduction

Markov decision processes offer a clean and rich framework for problems of control and decision making under uncertainty[BDH99; RN95]. A set of central problems in this family is the fully observable Markov decision problems under infinite-horizon criteria [Ber95]. We refer to these as MDP problems in this paper. Not only are the MDP problems significant on their own, but solutions to these problems are used repeatedly in solving problem variants such as stochastic games and partially observable MDPs [Sha53; Han98]. In an MDP model, the system is in one of a finite set of states at any time point. In each state an agent has a number of actions to choose from. Execution of an action gives the agent a reward and causes a stochastic change in the system state. The problem is, given a full description of the system and actions, to find a policy, that is a mapping from states to actions, so that the expected (discounted) total reward over an indefinite (or infinite) number of action executions is maximized.

Policy improvement is a key technique in solving MDPs. It is simple to describe and easy to implement, and quickly converges to optimal solutions in practice. The improvement method begins with an arbitrary policy, and improves the policy iteratively until an optimal policy is found. In each improvement step, the algorithm changes choice of action for a subset of the states, which leads to an improved policy. These algorithms differ on how the states are picked. In addition to policy improvement algorithms, other methods for solving MDPs include algorithms for linear programming, but variants of policy improvement are preferred due to speed and ease of implementation [Lit96; Han98; GKP01]. We remark that policy improvement can be viewed as a special linear programming solution method [Lit96; Ber95].

Unfortunately, the worst-case bounds on several implementations of policy improvement are exponential [MC94]. The exponential lower bounds have been shown for those policy improvement algorithms that in each iteration attempt to pick a single most promising state to improve, and are established on plausible heuristics for such a choice, such as looking ahead one or a constant number of steps. Let us call these 'selective' policy improvement. In a sense, the bounds imply that attempting to be smart about choosing which state to improve can lead to an exponentially long path to the optimal. On the other hand, the policy improvement technique is naturally implemented in a manner in which all states are examined, and any improvable state changes action. We will refer to this variation as policy iteration (PI) (see Section 2.1). It is known that PI has no worse than pseudo-polynomial[1] time [Ber95], while the exponential lower bounds on selective algorithms apply irrespective of the number representation [Lit96]. This suggests that the constraints on how PI advances may be inherently different than those for the selective policy improvement algorithms, and leaves hope for PI. But quantifying the advancement of PI has been difficult. The best upper bound on PI, besides the pseudo-polynomial bound, is also exponential $O(2^n/n)$

[1]That is, polynomial if the numbers are written in unary.

[MS99], where n is the number of states and each state has two actions. This upper bound is derived using certain partial order and monotonicity properties of the policy space. No lower bound other than the trivial $\Omega(n)$ on the number of iterations is known.

This paper describes a measure of advancement for PI that offers an explanation of why PI is fast. While we don't derive polynomial bounds for PI on general MDPs, we give the first polynomial bounds for PI on many significant subclasses of MDPs and related problems. Fig 1 shows the MDP problems in context. The problems get roughly harder from left to right, though the arrow lengths are not to scale. Our results apply to the enclosed problems. Let m be the number of actions, and W be the largest number in magnitude. We give an $O(n^2 \log nW)$ bound on the number of iterations of PI on MDP graphs that have a special structure: there exists a certain state b such that any sequence of visited states under any policy repeats state b before repeating another state. Each iteration takes $O(m)$ or $O(m + n^2)$ depending on the edges (Section 4.1). For the rest of the enclosed problems, we show that a variant of PI is polynomial. We discuss the problems next.

The general MDP problem can be formulated as a linear program (LP) in which the feasibility constraints have at most *three* variables per inequality. The subclasses of the MDPs for which we give polynomial policy improvement can be summarized as those for which the corresponding LPs require at most *two* variables per inequality (Section 4.4). We use the names MDP(2) and MDP(3) to differentiate. In the LP formulation of the MDP, finding the feasibility region under the system of inequalities is sufficient to solve the problem. We have shown that, conversely, extension of policy iteration solves the two variable per inequality linear feasibility (TVPI) problem and have given a polynomial bound for one such type of algorithm. The way policy improvement solves the TVPI problem is different from previous algorithms, as will be described in the paper. The best bound that we give for policy improvement on $MDP(2)$ is $O(mn^2 \log m \log W)$ and for deterministic MDPs (discounted or average reward) is $O(mn^2 \log m)$, but we expect that the bounds can be improved.

A main tool we use is the analysis of Newton's method for finding the zero of a function [Rad92; Mad00]. It was known that policy iteration was a form of Newton's method, but the well-known local quadratic convergence of Newton's method is not useful for deriving polynomial bounds. The analysis given here relates the way the process converges to attributes of the problem size. We point out that our results on deterministic MDPs do not require this analysis.

We begin with the definition of the MDP problems and policy iteration. Then we show how PI is a Newton's method on a single state problem, and describe its analysis that is used in showing polynomial bounds. We then show how this tool can be leveraged to establish polynomial bounds for various policy improvement algorithms on different problems. Due to space limitations, we omit the proofs.

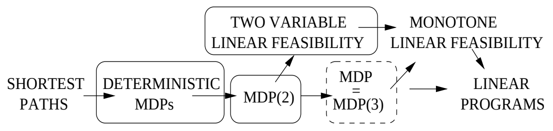

Figure 1: MDP problems in context. Problems are generalized in the direction of the arrows. Our results apply to the enclosed problems.

2 Preliminaries

A Markov decision process (MDP) consists of a set V of n vertices or system states, and for each vertex $v \in V$, a finite set E_v of edge choices or actions. Time is discretized and at each time point t, $t = 0, 1, \cdots$ the system occupies one vertex $v^{(t)}$, which is called the state (or vertex) of the system at time t, and $v^{(0)}$ is the initial state of the system. The means of change in vertex is edge choice. Each edge can branch out and have one or more end-vertices (Fig. 3a). By choosing edge $e \in E_u$ when the system is in state u, a *reward* of $r(e) \in R$ is obtained and the system transitions to an end-vertex v, with probability $Pr(e,v)$, where $\sum_{v \in V} Pr(e,v) = 1$. The effects of edges, *i.e.* the rewards and the transition probabilities, do not change with time, and they are completely specified as part of the problem instance. We will also use shorthand r_e to denote reward of edge e. A *policy* $\mathcal{P}$ is a mapping assigning to each vertex v a single edge choice $\mathcal{P}(v) \in E_v$. The value (vector) of a policy, denoted $\mathcal{V}_\mathcal{P}$ is a vector of n values, where $\mathcal{V}_\mathcal{P}[i]$ is the *value* of vertex v_i under policy $\mathcal{P}$, defined as the expectation of total reward $\sum_{t=0}^{\infty} r(\mathcal{P}(v^{(t)}))$, when $v^{(0)} = v_i$. We assume here that $\mathcal{V}_\mathcal{P}[i]$ is bounded and well behaved for any policy $\mathcal{P}$ and initial vertex[2].

Define the *optimal value* of a vertex v, denoted x_v^*, to be the maximum value of vertex v over all policies. A desirable and simplifying property of MDPs is that there exists an *optimal policy* which simultaneously maximizes the value of all vertices [Ber95]. Therefore the MDP problem is to find an optimal policy or compute the optimal value of all vertices.

2.1 Policy Iteration

The generic policy improvement procedure is given in Fig 2. Evaluating a policy means finding its corresponding value vector. Policy improvement algorithms differ on how they improve the policy. In a common technique which we will refer to as (classic) policy iteration (PI), the improvement is done in 'parallel' as follows: each vertex u picks its best *edge* according to:

$$\arg\max_{e \in E_u} \left(r(e) + \sum_{v \in V} Pr(e,v) x_v^{(t-1)} \right), \ \ t \geq 1,$$

[2]Policy improvement algorithms can be extended to discover ill-defined problems as discussed later. Whenever a discount is used (i.e. maximize expected $\sum_{t=0}^{\infty} \beta^t r_{\mathcal{P}(v^{(t)})}$, with $\beta < 1$), the problem is well-defined, and a discount is modeled by each edge having a transition probability to an absorbing zero-reward state.

```
1. Begin with an arbitrary policy
2. Repeat
3.   Evaluate policy
4.   Improve policy
5. Until no change in policy
```

Figure 2: Policy improvement.

where $x_v^{(t-1)}$ denotes the value of vertex v from the policy computed in iteration $t-1$ or initial policy if $t-1=0$. Performing the policy improvement step yields a policy with a value vector that componentwise has values as high or greater than the value vector of the old policy, and as long as an optimal policy is not found, it can be shown that in at least one component the value vector improves. Therefore the algorithm takes a finite number of iterations to converge to an optimal policy. Policy evaluation can be done in polynomial time by matrix inversion in general, but is easier in the problems discussed below. The parallel improvement step also takes a polynomial $\theta(m)$ time. Therefore each iteration is polynomial.

2.2 MDP(3) and MDP(2)

Any MDP problem can be polynomially reduced to one such that for each edge the maximum number of possible end vertices is two, by introducing extra vertices and edges as necessary. We will refer to such a problem as MDP(3). A generic MDP(3) edge is shown in Fig 3a. Thus each edge in MDP(3) is a branching edge, or a hyper-edge, and is parameterized by at most *three* numbers: its reward and two transition probabilities to the two end vertices. We will also be analyzing a restriction we call MDP(2), where each edge e connects two vertices, and is parameterized by two numbers[3], reward r_e, and transition probability μ_e, where $\mu_e \leq 1$. The remaining probability $1-\mu_e$ is understood to go to an absorbing state with no reward. Therefore an MDP(2) problem is viewed as a problem on a directed graph where policies are subgraphs with one or more cycles and every vertex has a path to some cycle. The MDP(2) problem is a generalization of the discounted deterministic MDP, where in the latter all edges have equal transition probability which is equal to the $1-\beta$, where β is the discount factor.

We call an edge of u ending in vertex v, a u-v edge. Similarly a path starting at u and ending in v is a u-v path.

3 Policy Iteration as a Newton's Method

Consider the MDP single vertex problem (Fig. 4), where the vertex v has a finite number of self-arc edges each parameterized by the pair (r_e, μ_e) of reward and transition probability, and we assume $\mu_e < 1$. The value of each (policy) is $\frac{r_e}{1-\mu_e}$, and the optimal value of the vertex would be $x^* = \max_e \frac{r_e}{1-\mu_e}$ and can be discovered in a single pass

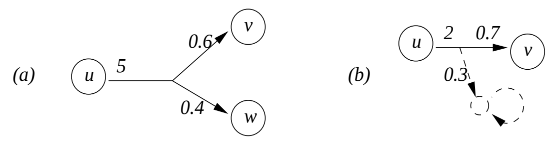

Figure 3: (a) An edge (action) with reward of 5 in an MDP (or an MDP(3)). (b) An edge in an MDP(2) has at most one (none-absorbing) end-vertex.

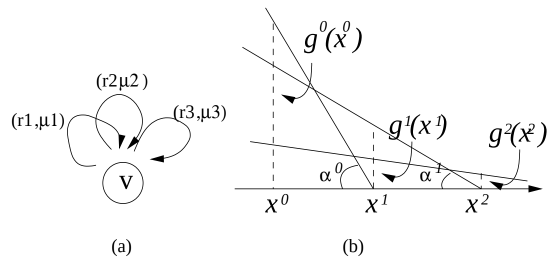

Figure 4: (a)A single-state MDP with a choice of 3 edges shown. (b) Newton's method for finding the zero of a function.

over the edges. As we will see later, the edges will expand to policies and therefore there may be an exponential number of them. Now consider the behavior of policy iteration. The algorithm begins with an arbitrary edge and computes the value x of v under the edge. In selecting the next edge, it picks the edge with maximum immediate reward $r_e + \mu_e(x)$, or equivalently, the edge with maximum *gain* $r_e + \mu_e(x) - x = r_e + (\mu_e - 1)x$. We will refer to $g_(e)(x) = r_e + (\mu_e - 1)x$ as the the *gain function* corresponding to e. Policy iteration evaluates the new policy, and we can verify that the new value is at the zero of the gain function, i.e. x, where $g_e(x) = 0$. Policy iteration then repeats with choosing and evaluating the edge with highest gain at the new x value, until the highest gain is zero. As shown in Fig 4 the process corresponds to finding the zero of a convex function (the upper envelope of the gain functions) using Newton's method. Therefore, the question in this case comes down to providing an effective measure of progress for the process. Note that, as discussed in [Lit96], the well-known local quadratic convergence results on Newton's method are not sufficient for proof of polynomial run time. In [Mad00] a bound is derived on the distance of each zero ($x^{(i)}$) to the zero of the optimal gain function, which is shown to geometrically decrease with each iteration, but the proof is rather long. We describe a result which has a simpler proof and a more direct geometric interpretation. It is derived in the context of the analysis of Newton's method in solving fractional linear combinatorial problems [Rad92]. From Fig 4, it is not hard to see that both the gains $g^{(i)}(x^{(i)})$ and slopes $\mu^{(i)} - 1 = \frac{g^{(i)}(x^{(i)})}{x^{(i)} - x^{(i+1)}}$ are converging to zero. However, a stronger constraint that quantifies the convergence rates can be derived:

Lemma 3.1 *[Rad92] The values and slopes of the sequence*

[3]The relationship between MDP, MDP(3), and MDP(2) is similar to that among SAT, SAT(3), and SAT(2). In fact, policy iteration algorithms (and other algorithms that solve MDPs) can be seen as attempting to find the feasibility of a set of linear inequalities. In this sense, solving MDPs correspond most directly to satisfying a conjunction of horn clauses in logic. See Section 4.4.

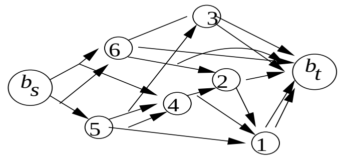

Figure 5: An MDP adag graph rolled out (b replaced by b_s and b_t). Transitions to the absorbing state, edge rewards and transition probabilities, are not shown.

of gain functions satisfy: $\frac{g^{(i+1)}(x^{(i+1)})}{g^{(i)}(x^{(i)})} + \frac{\mu^{(i+1)}-1}{\mu^{(i)}-1} \leq 1$.

Intuitively, the geometric constraints on the process forces either the gain (height) or the angle and its tangent to decrease significantly in each iteration. The proof is simple, and works by writing the gains and slopes in terms of their definitions with substitutions and simplifications. As $\frac{g^{(i+1)}(x^{(i+1)})}{g^{(i)}(x^{(i)})} \geq 0$ and $\frac{\mu^{(i+1)}-1}{\mu^{(i)}-1} \geq 0$, we conclude that either $\frac{g^{(i+1)}(x^{(i+1)})}{g^{(i)}(x^{(i)})} \leq 1/2$ or $\frac{\mu^{(i+1)}-1}{\mu^{(i)}-1} \leq 1/2$. This result suffices for showing polynomial run-times, as we will see next.

4 Polynomial Policy Improvement

4.1 Almost Acyclic MDP Graphs

Consider the following special MDP graph, which we call *adag*, for *almost* a *directed acyclic graph*. In adags, there is an ordering on the vertices, so that edges of a vertex i can branch or end in lower numbered vertices, a special 'bottleneck' vertex b, or an absorbing zero-reward vertex, only. The edges of vertex b can branch to any vertex. Thus all interesting cycles must go through b. The analysis of policy iteration on this problem can be reduced to the above Newton analysis as follows. Consider rolling out the graph so that it becomes acyclic as follows: We may think of removing the cycles in the policy by replacing vertex b by two vertices, b_s (start vertex) and b_t (target), where all branches of vertices leading to b now end in b_t, and all the edges of b belong to b_s.

Now, for any policy P, we define the *value function* $f_{u,p}(x) = r_{u,p} + \mu_{u,p}x$, which is interpreted as the expected value obtained if one starts at vertex u and follows the actions prescribed by the policy, until one arrives at vertex b_t, at which time the value x of b_t is collected. $\mu_{u,p}$ and $r_{u,p}$ can be expressed in terms of the immediate neighbors of u in the policy, $\mu_{u,p} = \sum_{v\in V} Pr(e,v)\mu_{v,p}$ and $\mu_{u,p} = \sum_{v\in V} r_e + Pr(e,v)r_{v,p}$, and can be computed right to left for any policy, with $\mu_{b_t,p} = 1$ and $r_{b_t,p} = 0$. If we apply policy iteration, with any assigned value x for b_t, in at most i iterations, the value of any vertex u of order i is equal to the highest it can reach, and the policy attained is a highest valued/gain policy at value x, which we denote by P_h (x is known from the context).

Consider the original graph and policy iteration progressing on it. The functions $f_{u,p}$, for any vertex u and policy P remain well defined ($f_{b,p}(x)$ is the value vertex b_s obtains if b_t has value x and policy P is used). We just saw that if value of b remains constant for n iterations, b obtains the policy that gives it the highest value in at most n iterations. During policy iteration, values improve in general, and value of b (b_s) may increase from one iteration to the next. Nevertheless, the following lemma holds:

Lemma 4.1 *Consider policy iteration on the adag graph, and the value of* b, $x^{(t_0)}$ *at any iteration* t_0, *as policy iteration progresses, and let* P_h *be the highest valued policy at iteration* t_0. *Then for any vertex* u *with order* i, *vertex* u *has value equal or exceeding* $f_{u,p_h}(x^{(t)})$ *at all iterations* $t \geq t_0 + i$.

The lemma is proved by induction on the order of vertices, where vertex b has order n. We can now make the connection to the Newton process. Each policy P_h highest valued at a some value of b yields one such gain function, $f_{b,P_h}(x^{(t)}) - x^{(t)}$, where $x^{(t)}$ is value of b at time t. Lemma 4.1 states that such a policy or better one is obtained in at most n iterations, and as policies are evaluated, the point where its gain is zero is surpassed in at most n iterations. All we need is then to bound the maximum and the minimum of the gains and slopes corresponding to policies. Recall that W is the largest number in the input representation.

Lemma 4.2 *We have* $nW^2 \geq g^{(i)}(x^{(i)}) \geq W^{-3n}$ *and* $1 \geq 1 - \mu^{(i)} \geq W^{-1}$,

The number of iterations is consequently $O(n^2 \log nW)$. We expect this bound is improvable. Policy evaluation on the MDP(3) adags (when the edges' branches are constant) takes only $O(n)$ time, but on an MDP adag, when an edge can to $\theta(n)$ vertices, it takes $O(n^2)$.

Theorem 4.3 *Policy iteration takes respectively* $O(mn^2 \log nW)$ *and* $O((m + n^2)n^2 \log nW)$ *on MDP(3) and MDP adags.*

To summarize, we showed that we may define value and gain functions for policies, and each highest gain policy correspond to a line in Fig. 4. Even though there is an exponential number of policies, policy iteration efficiently (within n iterations) surpasses the zero of the current highest gain policy.

We remark that Lemma 4.1 holds in other variations to the policy improvement step, and in particular if the policy improvement is performed sequentially rather than in parallel (such as Gauss-Siedel [Ber95]). We can also use a shortest paths algorithms to speed up finding P_h on adags: by updating the vertex values and edge choices in an increasing order, a policy P_h is discovered in $O(m)$ time rather than $O(mn)$ in an MDP(3) for example, and in $O(m + n^2)$ for MDPs. Thus the above runtimes reduce by a factor of n.

A plausible direction to extend the analysis to general MDPs is to consider increasing the number of bottleneck vertices. However, we haven't succeeded in such an approach: it appears that a novel measure of progress or potential function, is needed over two or more bottleneck vertices. Nevertheless, we can show that the Newton analysis leads to polynomial policy improvement algorithms on many problems as shown below.

```
1. Freeze all vertices on a choice of edge
2. Repeat $n$ times
3.     Unfreeze a vertex.
4.     Apply policy iteration until no improvement
```

Figure 6: Freezing Policy Iteration.

4.2 MDP(2)

We now show that a 'freezing' policy iteration algorithm has a run-time of nT where T denotes the run-time of policy iteration on an MDP(2) adag (edges are of MDP(2) type), already shown polynomial above. We remark that the same algorithm works on many problem variations, including deterministic MDPs under discounted (see below) or average rewards. The freezing policy iteration works as given in Fig. 6, and has n phases. A frozen vertex does not change choice of edge during policy iteration. In each phase, one more vertex is unfrozen to select any of its edges in a policy improvement step. Each phase begins with the optimal policy found in the last phase. The basic observation is that in a single phase the source of improvements in the value of any vertex must be due to a path to the most recently unfrozen vertex b (by optimality of the policy from last phase). In particular, any new cycle created in the phase must have b in it. We now see the connection to adags. If no new cycle is created in the phase, it is not hard to see that no more than n policy iterations are needed to find a best policy for the phase. Otherwise, the above analyses apply to bound the number of cycles and iterations. A similar speed-up idea also applies, in this case using a Dijkstra style shortest path algorithm as there are cycles in the graph (but no 'negative cost' cycles in a shortest paths setting as edge probabilities is no more than 1). We omit the details.

Theorem 4.4 *The freezing policy iteration algorithm finds an optimal policy in time $O(mn^3 \log(nW)$ on MDP(2) problems. The freezing algorithm with Dijkstra style shortest paths runs in $O(mn^2 \log(m) \log(W))$.*

The polynomial bound for freezing policy iteration is independent of the order of unfreezing, and is strong evidence that plain policy iteration is polynomial on MDP(2) problems. The freezing technique does not extend to MDP(3) problems: assume all edge rewards are zero except for the edges of the last unfrozen vertex. The problem is then equivalent to a full MDP problem where the objective is to increase the probability of reaching a certain vertex. As edges can have branches, cycles that do not include the unfrozen vertex remain significant.

4.3 Deterministic MDPs

In deterministic MDPs, all edges have equal probabilities. In case of discounted MDPs, edge probability is $\mu = 1 - \beta$, where $\beta < 1$ is the discount factor. We do not need the analysis of the Newton's method for analyzing deterministic problems. Consider a deterministic adag graph unrolled, and any path (policy) from any vertex v to b_t. The corresponding value function $f_{u,p}(x) = r_P + \mu_P(x)$ has slope μ^k where k is the length of the path. Thus paths of the same length correspond to value functions with equal slope, i.e. parallel increasing lines, with longer paths having smaller slope, and for each path length there is a single highest gain policy of *a fixes length* at all values of b. There are at most n such paths. We thus have,

Theorem 4.5 *Policy iteration takes $O(mn^2)$ on discounted deterministic adag graphs. Freezing policy iteration solves discounted deterministic MDPs in $O(mn^3)$, and the algorithm with the shortest path sub-routine solves the problem in $O(mn^2 \log(m))$.*

Deterministic average reward problems, and minimum cost-to-time ratio cycle problems also have policy iteration algorithms, and similar arguments show that the freezing variants have similar polynomial bounds. The best known strongly polynomial[4] algorithm for average-reward problems have $\theta(mn)$ time. An interesting question is whether the number of policies with highest value for an MDP or MDP(2) is also polynomial in n and m. An $O(mn^{\log n})$ bound for MDP(2) is given [Mad00], and a similar lower bound is given in [Car83] for a related parametric shortest paths problem, which we expect holds here.

4.4 Feasibility Checking

The linear programming formulation for an MDP has the constraints, $x_v \geq r_e + \sum_{u \in V} Pr(e, u)$, for each pair of vertex v and edge e in E_v, and the optimal value for each vertex v is the left most point of the feasible interval[5] for variable x_v [Ber95]. For an MDP(2) problem each inequality involves at most two variables. A linear feasibility problem is the problem of determining whether a system of linear inequalities is feasible, and the problem in which each inequality has at most two variables is *the two variable per inequality* problem, or TVPI (feasibility). The TVPI problem was first studied as a means of deriving polynomial time algorithms for general linear programs, but it also has applications in many network flow problems [Sho81; Meg83; Way99]. Algorithms solving TVPI, with a little modification, can report the end points of the interval of feasibility for each variable when the system is feasible. Therefore, from our comments above we observe that these algorithms also solve MDP(2) problems. We can show that the converse is true: policy improvement algorithms, appropriately extended, solve TVPI, and furthermore the techniques above show how to obtain polynomial bounds. We briefly discuss how the algorithms work.

All TVPI feasibility algorithms represent the problem in terms of a directed graph where each vertex corresponds to a variable and each inequality corresponds to an edge with two parameters: an inequality of the form $ax - by \geq c$, can be rewritten as $x \geq \mu y + c'$, for example, and an edge is directed from v_x to v_y with parameter c' and μ, where μ is the analogue of the transition probability in MDPs, but here it is possible that $\mu \geq 1$. In these graphs, cycles and paths to cycles are sources of upper and lower bounds

[4]Independent of number representation

[5]The property that the feasible region of a variable is a single contiguous interval follows from the fact that the feasible region of an LP system is convex.

for a variable's feasible range. Previous polynomial TVPI techniques (other than general linear programming) use binary search or parametric binary search to locate the intervals of feasibility of variables (see for example [Meg83; HN94]). Policy improvement algorithms solve the TVPI feasibility problem in two phases. Just like in the case of policy improvement on MDPs, during the algorithm, vertices have values, and vertex values converge to the left end-points of the feasibility intervals (similar to maximizing expected rewards) from the left (infeasible) side. In a second phase, right end-points are approached and discovered from their right side (similar to an MDP formulation with costs instead of rewards). Infeasibility may be detected during either phase. The complete description of algorithms, together with correctness and polynomial run-time is given in the full paper. A freezing version of policy iteration solves the problem in $O(mn^3 \log nW)$.

Policy improvement algorithms described above work only on the *monotone* feasibility problem, i.e. where each inequality has at most one variable with a positive coefficient: $ax - by \geq c$, with $a, b \geq 0, c \in R$. However, unrestricted feasibility problem can be reduced to the monotone kind for TVPI [HN94]. Likewise, we believe that policy improvement algorithms can be extended to solve the feasibility problem for systems of general monotone linear constraints (i.e. no restriction on the number of variables), and we conjecture that PI in particular is polynomial on these problems.

5 Summary and Discussion

We showed that policy iteration (PI) is polynomial on a special MDP graph and used the analysis to establish that many policy improvement algorithms are polynomial on a number of special MDPs. We also explained that the MDP linear programming formulation is a monotone linear feasibility problems and described extensions of policy improvement that solve the two variable per inequality linear feasibility (TVPI) problem in polynomial time. We expect that these algorithms will be among the most competitive for TVPI. Similar results have been observed in related settings [DG98].

The analysis of the Newton's method given was at the heart of the polynomial bounds for problems classes that are more general than deterministic MDPs in Fig. 1. We believe that the analysis provides an explanation as to why policy iteration is fast. In general, policy iteration begins with policies that give high immediate rewards but not necessarily high transition probabilities among the (good) states, and moves towards those policies with lower immediate rewards, but with higher transition probabilities resulting in higher long term accumulated rewards. The Newton analysis quantifies this process. We conjecture that proof methods along similar lines will establish PI and variants polynomial on MDPs. We remark that there are policy improvement algorithms for two-player game versions of MDPs for which no polynomial time algorithm is known [Sha53; Con92]. Again, a variant of policy improvement given for example in [HK66] has a similar interpretation as a Newton's method [PAI69]. This further motivates the analysis of PI, especially along the above lines.

References

[BDH99] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *JAIR*, pages 157–171, 1999.

[Ber95] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[Car83] P. J. Cartstensen. *The compelexity of some problems in parametric linear and combinatorial programming*. PhD thesis, U. of Michigan, 1983.

[Con92] A. Condon. The complexity of simple stochastic games. *Information and Computation*, 96(2):203–224, Feb. 1992.

[DG98] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 17(10):889–99, 1998.

[GKP01] C. Guestrin, D. Koller, and R. Parr. Max-norm projections for factored MDPs. In *AAAI*, pages 673–679, 2001.

[Han98] E. A. Hansen. *Finite Memory Control of Partially Observable Systems*. PhD thesis, U. of Mass., Amherst, 1998.

[HK66] A. Hoffman and R. Karp. On nonterminating stochastic games. *Management Science*, 12(5), 1966.

[HN94] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SICOMP*, 23(6):1179–1192, Dec 1994.

[Lit96] M. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown, 1996.

[Mad00] O. Madani. *Complexity Results for Infinite-Horizon Markov Decision Processes*. PhD thesis, U. of Washington, 2000.

[MC94] M. Melekopoglou and A. Condon. On the complexity of the policy improvement algorithm for markov decision processes. *ORSA Journal on Computing*, 6(2), 1994.

[Meg83] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SICOMP*, 12(2):347–353, 1983.

[MS99] Y. Mansour and S. Singh. On the complexity of policy iteration. In *Uncertainty in AI*, 1999.

[PAI69] M. Pollatschek and B. Avi-Itzhak. Algorithms for stochastic games with geometrical interpretation. *Management Science*, 15:399–413, 1969.

[Rad92] T. Radzik. Newton's method for fractional combinatorial optimization. In *33rd FOCS*, pages 659–669, 1992.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence:A Modern Approach*. Prentice Hall, 1995.

[Sha53] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of sciences USA*, 39:1095–1100, 1953.

[Sho81] R. Shostack. Deciding linear inequalities by computing loop residues. *J. ACM*, 28:769–779, 1981.

[Way99] K.D. Wayne. A polynomial combinatorial algorithm for generalized minimum cost flow. In *31st STOC*, 1999.