World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:12, 2007

Trajectory-Based Modified Policy Iteration

R. Sharma, and M. Gopal

***Abstract*—**This paper presents a new problem solving approach that is able to generate optimal policy solution for finite-state stochastic sequential decision-making problems with high data efficiency. The proposed algorithm iteratively builds and improves an approximate Markov Decision Process (MDP) model along with cost-to-go value approximates by generating finite length trajectories through the state-space. The approach creates a synergy between an approximate evolving model and approximate cost-to-go values to produce a sequence of improving policies finally converging to the optimal policy through an intelligent and structured search of the policy space. The approach modifies the policy update step of the policy iteration so as to result in a speedy and stable convergence to the optimal policy. We apply the algorithm to a non-holonomic mobile robot control problem and compare its performance with other Reinforcement Learning (RL) approaches, *e.g*., a) Q-learning, b) Watkins $Q(\lambda)$, c) SARSA(λ).

***Keywords*—**Markov Decision Process (MDP), Mobile robot, Policy iteration, Simulation.

I. Introduction

We consider simulation-based methods for controlling stochastic sequential decision-making problems or learning methods for optimizing the policy of an agent interacting with an environment. Simulation is an effective tool for analyzing systems for which a perfect analytic representation may not be available. Simulation can be used for comparative evaluation of policies on the basis of empirically generated information for a given policy [2].

Trajectory based Modified Policy Iteration (TMPI) is an approach which, starting from the simulation of the system, iteratively builds up the MDP model and cost-to-go values for the state-action pairs and utilizes them to generate a sequence of policies finally converging to an optimal policy solution for the underlying system. It is an attempt to find a middle path between the methods that focus on the use of simulation to approximate cost-to-go values, e.g., Neuro Dynamic Programming (NDP) [1] and the MDP model parameters approximation approaches [2]. TMPI generates estimates for the cost-to-go values and the MDP model parameters from the same simulated trajectory to carry out a model based policy update. It exploits the inherent structure of the MDP formulation, policy evaluation capabilities of TD (λ)[15] and ability of the simulation to focus on the most relevant parts of the state space to discover the optimal policy with least agent-environment interactions.

State space aggregation has been used to reduce the cardinality of the state space by suitably partitioning the state space into a number of aggregated state subsets. Each subset is then treated as a single state yielding a new aggregated MDP, characterized by associated rewards and transition probabilities. Thus any process can be modeled as a finite state MDP using state aggregation [2]. The new process can often be solved exactly by standard MDP solution techniques, then the process can be disaggregated and the aggregated solution adjusted appropriately for application on the original model. TMPI can be used to discover the optimal policy either using the real experience or simulated experience as a surrogate to the real experience (as used in our case), to update both the value function estimates and the model parameters. Essentially TMPI comprises of:

(i) **Model Learning**: Improve the Model.
(ii) **Direct RL**: Improve the Value Function.

In TMPI these two learning's occur intermittently aiding each other as shown below in Fig. 1:

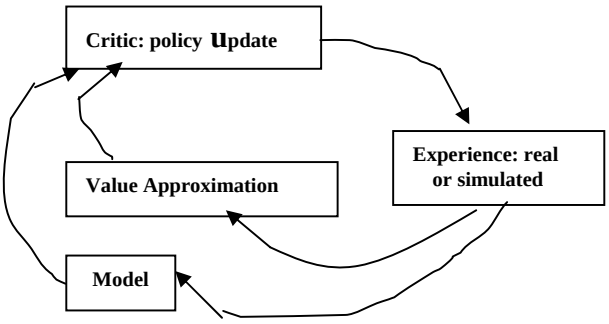

Fig. 1 TMPI Algorithm Representation
Direct RL method: TD (λ) with eligibility coefficients

TMPI modifies standard policy iteration algorithm [7], the policy update step in particular, resulting in quick convergence to optimal policy. Usual policy iteration algorithm fixes a policy μ, evaluates the associated cost-to-go function J^{μ} using TD(λ) or any other method and then performs a policy update. This is the standard framework of "Actor/Critic Systems", the actor uses the policy μ to control the system while the critic observes the rewards/costs and tries to compute J^{μ}.

In standard policy iteration the policy μ is held fixed till the critic's estimate converges to J^{μ}. The critic then passes this J^{μ}

Manuscript received March 18, 2005.

R. Sharma is a research scholar with the Electrical Engineering Department, Indian Institute of Technology, Delhi and a faculty at NSIT, Delhi, India. (Phone: (91) 011- 27943497; fax: (91) 011- 25099022; e-mail: rajneesh496@rediffmail.com).

M. Gopal, is a Senior Professor with the Department of Electrical Engineering, Indian Institute of Technology, Delhi, Hauz Khas, New Delhi, India. (e-mail: mgopal@ee.iitd.ernet.in).

value to the actor who forms a new policy by minimising the right hand side of the Bellman's equation, i.e., at each state i, action u is chosen that minimises:

$$\sum_{j=1}^{n} p_{ij}(u)(g(i,u)+J^{\mu}) \quad \forall u \in U(i) \tag{1}$$

where $U(i)$ = action set at state i , j is the successor state for (i ,u) pair and n = number of feasible successor states for state i.

This may be highly computationally intensive as the critic's evaluation may converge slowly to the true J^{μ} values. Optimistic policy iteration [5] attempts to address this issue by carrying out more frequent policy updates, i.e., without waiting for the policy update step to converge to the true J^{μ} values. The method however, assumes availability of a valid model. Bertsekas et. al. [1], further impose two assumptions:

(i)The policy evaluation algorithm is sound, i.e., with policy μ held fixed the J value as evaluated by critic converge to J^{μ}.

(ii)The critic communicates to actor infinite number of times.

According to Bertsekas [1], under these assumptions, if the sequence of policies generated by the actor converges then the limit must be an optimum policy. They further state that even then the algorithm may converge to a value different from J^*(optimum value) or fail to converge. When such incomplete J^{μ} evaluations are used in conjunction with a model, which is far from perfect, this could lead to much worse results with more model updates [6].

Fig. 2 below shows details of the TMPI system. At the end of a trajectory, we evaluate expected cost-to-go for all feasible state-action pairs and the optimal action at a state corresponds to the state-action pair with minimum expected cost-to-go value. For any state-action pair the expected cost-to-go value is evaluated based on the current value of all the feasible successor state-action pairs and current approximation for the one step costs and transition probabilities.

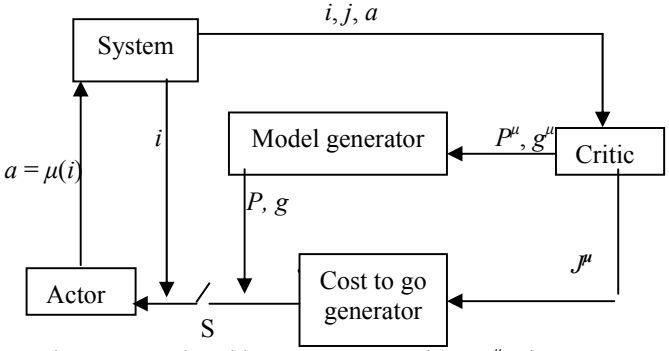

Fig. 2 TMPI: The critic computes approximate J^{μ} using restart TD(λ),it further computes P^{μ}, g^{μ} .The switch S closes at the end of a simulated trajectory. Actor updates policy μ based on P, g , J vectors. i =current state, action a = $\mu(i)$, j = successor state

Starting with a given simulation model (or one created through observations as the System/Process evolves in time (Fig. 2), define states, actions and objective function for the aggregated MDP model. The TMPI approach then generates trajectories through the state space; each trajectory is generated in accordance with an updated policy. The trajectories are generated asynchronously, i.e., from randomly chosen initial states. Each time a trajectory visits a particular state-action pair (i,a) it generates a sample for $J^{\mu}(i,a)$ where μ = current policy. Within a trajectory these samples are combined using restart TD(λ)[15]. This estimate of $J^{\mu}(i,a)$ for the states visited by the current trajectory is suitably added to earlier aggregated estimate of $J(i,a)$ from previous trajectories yielding current estimate of $J(i,a)$. Similar aggregation is done for samples of $g^{\mu}(i,a)$ and $P^{\mu}(i,j,a)$ to yield P and g estimates. The policy update step then uses these aggregated estimates for producing an improved policy to generate next trajectory through the state space.

We show the effectiveness of the proposed algorithm on a simulated steering control of a mobile robot [8] and compare its performance against other benchmark RL algorithms i.e., Q Learning [12][14], Watkins $Q(\lambda)$[17] and SARSA(λ)[7][9].

II. Theoretical Issues and TMPI Details

We consider a Markov decision process with finite state and action set. State space consists of finite set S = {1,2,…..,n} of states and finite set U={U(1),U(2),….U(n)},defining possible actions at each state. With each state-action pair (i,a) we associate transition probabilities $P_{ij}(a)$ and one step cost $g(i,a)$(assuming random one step costs). We define policy μ as a mapping μ: $S \rightarrow U$. Given a policy μ, the state evolution is the well-known Markov chain with transition probability defined as:

$$P\{x_{t+1} = (j, \mu(j)) \mid x_t = (i, \mu(i))\} = P_{ij}(\mu(i)) \tag{2}$$

We view a new system consisting of the original states (1,2,……,n) together with all pairs (i,a) where $i \in S$ and $a \in U(i)$.The expected cost of a policy μ, starting from initial state $(i, \mu(i))$ for finite horizon case, i.e., for N stage problem:

$$J^{\mu}(i,\mu(i)) = E[\alpha^N G(i_N, \mu(i_N)) + \sum_{k=0}^{N-1} \alpha^k g(i_k, \mu(i_k))] \tag{3}$$

where $\alpha^N G(i_N, \mu(i_N))$ = Terminal Cost
and α = discount factor

A. TMPI Algorithm

Starting with an arbitrary policy (specifying randomly chosen action from the set of feasible actions at each state), we generate a sequence of policies μ_1, μ_2,…… We simulate a trajectory through the state space as per the current policy μ, each starting from state action pair $(i, \mu(i))$ where i is the randomly chosen starting state and $\mu(i)$ is the action specified at i by μ.

At the start of each trajectory we initialize following vectors:

$$g^{\mu} = 0,\ P^{\mu} = 0 \begin{cases} \text{MDP Model Parameters} \\ \text{for policy } \mu \end{cases} \tag{4}$$

$$J^{\mu} = 0 \tag{5}$$

This represents initial lack of information of these parameters and values.

As we simulate the process, $(i, \mu(i))$ pairs are visited and we update the parameters and values as:

(i) For the J^μ updates we use the on-line variant of the restart TD(λ) [10]of optimistic policy iteration: when the k-th step in a trajectory has been simulated, i.e., from $(i_k, \mu(i_k))$ to $(i_{k+1}, \mu(i_{k+1}))$. We get the temporal difference:

$$d_k = g^\mu(i_k, \mu(i_k)) + \alpha\, J^\mu(i_{k+1}, \mu(i_{k+1})) - J^\mu(i_k, \mu(i_k)) \quad (6)$$

where $g^\mu(i_k, \mu(i_k)) = \text{One-step cost of taking } \mu(i_k) \text{ from } i_k$

We then update J^μ values for the $(i, \mu(i))$ pairs visited by the trajectory as:

$$J^\mu_{k+1}(i, \mu(i)) = J^\mu_k(i, \mu(i)) + \gamma_k(i, \mu(i))\, e_k((i, \mu(i))\, d_k \quad (7)$$

$\forall\, (i, \mu(i)) \in (i, a)$ pairs of the trajectory

where

$$\gamma_k(i, \mu(i)) = \frac{1}{N_k(i, \mu(i))}$$

$= \text{Step size coefficient for pair } (i, \mu(i))$

$N_k(i, \mu(i)) = \text{Number of visits to a } (i, \mu(i))$ pair within the trajectory for policy μ and

$e_k((i, \mu(i)) = \text{eligibility coefficient for } (i, \mu(i))$ pair which are determined as per restart TD(λ) as:

$$e_k((i, \mu(i)) = \begin{cases} 1 & \text{if } (i_k, \mu(i_k)) = (i, \mu(i)) \\ \alpha\lambda\, e_{k-1}(i, \mu(i)) & \text{otherwise} \end{cases} \quad (8)$$

Thus the eligibility of $((i, \mu(i))$ pair is the degree to which it has been visited in the recent past within a given trajectory.

(ii) One step costs are updated as:

$$g^\mu_{k+1}(i_k, \mu(i_k)) = g^\mu_k(i_k, \mu(i_k)) + \gamma_k(i_k, \mu(i_k))[g(i_k, \mu(i_k)) - g^\mu_k(i_k, \mu(i_k))]$$

where $g(i_k, \mu(i_k)) = \text{random one step reward/cost}$ (9)

(iii) P^μ update:

Let N^μ_α = number of visits to $\alpha = (i, a)(a = \mu(i))$ by the simulated trajectory under the policy μ and let $N^\mu_{\alpha\beta}$ = number of simulated transitions from $\alpha = (i, a)$ to $\beta = (j, b)(b = \mu(j))$. We define transition probability as:

$$P^\mu_{\alpha\beta} \cong N^\mu_{\alpha\beta} \Big/ N^\mu_\alpha \cong N^a_{\alpha\beta} \Big/ N^a_\alpha \quad (10)$$

Next we show that the probability as defined above reduces to that defined for a transition from state i to state j (defined on original state space) under action $a = \mu(i)$ if the policy is held fixed. Under a fixed policy μ transitions take place as:

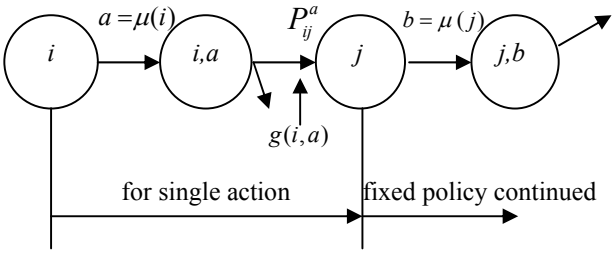

Fig. 3 Equivalence between state transition probabilities and state-action transition probabilities

As shown above in Fig. 3, under a fixed policy μ, we move from state i to (i,a) deterministically, then next state j is reached with probability P^a_{ij} and a cost $g(i,a)$ is incurred. Now if the policy is kept fixed we move deterministically to (j,b). Thus for a fixed policy i and (i,a) coincide and similarly j & (j,b) are same.

$$\text{Thus } P^a_{ij} \cong N^a_{ij} \Big/ N^a_i \text{ is same as } P^\mu_{\alpha\beta} \cong N^\mu_{\alpha\beta} \Big/ N^\mu_\alpha \quad (11)$$

where N^a_{ij} = number of transitions from i to j under $a = \mu(i)$

and N^a_i = number of visits to state i under action $a = \mu(i)$

$= N^\mu_\alpha$

This empirically generated estimate (equation 11) of the true relative frequency (or probability) of the transition will become more accurate with generation of more simulated trajectories. The transition matrix for the MDP model consists of m matrices where m=max $[U(i)]$ for i = 1,2,……n. Each (state-action transition) matrix corresponds to one action and each row in a matrix gives transition probability distribution for a state under that action.

A trajectory simulated as per a policy would provide us with probability distribution for only a single action in each state. We, therefore, need to include different actions at each state by using some random steps (as per a random policy) in the initial phase of the procedure or the pseudo-stochastic policy. This facilitates some amount of exploration in the search for optimal policy. By intelligently exploring the action space we can eventually generate enough information to obtain true probability distribution for each state-action pair in an iterative manner.

The TMPI algorithm makes use of the Markov property [1]

$$|\delta| < 0.05\,\text{m} \quad \text{and} \quad \alpha < \pi/6$$

for combining information generated by successive trajectories. Markov property states that the probability of the state transition P^a_{ij} depends only on the states i, j and the action a and not on the previous transition history or alternatively state transition probability distribution is independent of the policy being used if two policies assign same action to a particular state.

B. Aggregating Information

We combine the estimates produced by successive policies /trajectories by forming a weighted sum of the contributions of each policy as per the amount of experience provided by a

policy at a state-action pair, i.e., let η and ξ be two policies and $a(\eta)$ and $a(\xi)$ be the action a as applied within policy η and ξ respectively then:

$$P_{\alpha\beta}^{a(\eta)} = N_{\alpha\beta}^{\eta} \Big/ N_{\alpha}^{\eta} \qquad P_{\alpha\beta}^{a(\xi)} = N_{\alpha\beta}^{\xi} \Big/ N_{\alpha}^{\xi} \tag{12}$$

$$P_{\alpha\beta}^{a(aggregate)} = N_{\alpha\beta}^{a} = N_{\alpha}^{\eta} \Big/ N_{\alpha} * P_{\alpha\beta}^{a(\eta)} + N_{\alpha}^{\xi} \Big/ N_{\alpha} * P_{\alpha\beta}^{a(\xi)}$$

$$= N_{\alpha}^{\eta} \Big/ N_{\alpha} * N_{\alpha\beta}^{\eta} \Big/ N_{\alpha}^{\eta} + N_{\alpha}^{\xi} \Big/ N_{\alpha} * N_{\alpha\beta}^{\xi} \Big/ N_{\alpha}^{\xi}$$

$$= \frac{N_{\alpha\beta}^{\eta} + N_{\alpha\beta}^{\xi}}{N_{\alpha}} = \frac{N_{\alpha\beta}^{\eta} + N_{\alpha\beta}^{\xi}}{N_{\alpha}^{\eta} + N_{\alpha}^{\xi}}$$

The data from the current simulated trajectory is combined with data from all previous trajectories. Thus previously available data is treated as if it had come from a single simulated trajectory. By forming such a weighted sum, we would be able to capture the system behavior to a greater extent. This combined information is used in the policy update step to generate an improved policy.

C. Policy Update Step

In policy iteration the policy is updated as: Q factor for the state-action pair (i,a) is computed for the policy μ :

$$Q^{\mu}(i,a) = \sum_{j=1}^{n} P_{ij}(a)[g(i,a) + \alpha J^{\mu}(j)] \tag{13}$$

Then policy is updated as:

$$\bar{\mu}(i) = \arg \min_{a \in U(i)} Q^{\mu}(i,a) \tag{14}$$

where n = number of feasible successor states for i.

In TMPI approach we generate J^{μ} $(i,\mu(i))$ values from a simulated trajectory for the visited $(i,\mu(i))$ pairs. We then form a weighted sum of these samples from successive trajectories to generate $J(i,a)$ values ,which are more or less policy independent. We then perform a policy update based on these current $J(i,a)$ values. Thus the policy update does not solely depend on the data generated by the policy followed during the current simulated trajectory.

We find:

$$Q(i,a) = \sum_{j=1}^{n} P_{ij}(a)[g(i,a) + \alpha J(j,a)] \tag{15}$$

$$= \text{Expected cost to go for pair } (i,a)$$

And update the policy as:

$$\bar{\mu}(i) = \arg \min_{a \in U(i)} Q(i,a) \tag{16}$$

Thus instead of using the current-policy dependent measure, i.e., $Q^{\mu}(i,a)$ values for updating the policy (as in standard policy iteration), the algorithm forms and utilizes a policy independent measure, i.e., $Q(i,a)$ values in the policy update step of policy iteration.

III. Simulation Model and Results

We give results of an experiment involving steering control of a simulated non-holonomic mobile robot. We attempt two basic problems under assumption of constant velocity tracking.

(i) Reducing distance to a line.
(ii) Trajectory tracking.

Details of the simulation model used for the mobile robot can be found in [3][8]. A state is given by the distance δ in meters from the line and orientation α in radians with respect to the line $s_t = [\delta \; \alpha]$. The state space is defined as:

$$-1 \le \delta \le +1$$
$$0 \le \alpha \le 2\pi$$

δ has been discretised, with a resolution of 0.05 m while orientation has been discretised with a resolution of $\pi/6$ or 30^{o}, yielding a total of 492 states and three discrete actions at each state.

We have tried to incorporate effect of noise/disturbances by assuming that the incremental distance moved by the robot between two consecutive time steps can vary by $\pm 20\,\%$ from the value as predicted by the model, i.e., suppose if $\delta_{k+1} - \delta_k = 0.2\,\text{m}$ as given by the model then we may take $\delta_{k+1} - \delta_k = (0.2 + 0.04 * n)\;\text{m}$ where n is uniform noise in [-1 1]. Similarly, angle of the robot α is also perturbed by $\pm 20\,\%$. We carry out the model updates as well as the Value Function updates based on this simulated experience. At the end of an episode/trajectory we use the current model and value function approximates to generate an improved policy.

A. Robot Initialization and Trajectory Generation

At the start of each simulated trial, the robot is initialized from $\delta = \pm 0.8\,\text{m}$ and random initial orientation. Starting from this state (δ, α), the robot takes actions as per the current ε-soft policy, i.e., with probability ε it takes uniformly random action and with probability (1- ε) it takes action as per the current greedy policy, thus generating a trajectory through the state space. A trial is terminated when:

(i) The robot violates the state-space boundaries, i.e., if

$$|\delta| > 1\text{m}$$

This is reckoned to as a negative result.

(ii) The robot correctly tracks the target for 10 consecutive steps. This is reckoned to as a positive result.

(iii) If the maximum trajectory length (120 robot steps) is exceeded.

Two tasks have been attempted:

(a) Distance Reduction**:** After initialization, the robot has to reduce distance to the line x=0 and satisfy $|\delta| < 0.05\,\text{m}$ for 10 consecutive steps.

(b) Trajectory Tracking**:** Track a reference trajectory (e.g., + ve x-axis) for 10 consecutive steps, i.e., satisfy:

At the end of a trajectory, estimates from the current trajectory are combined with the estimates from previous trajectories to generate aggregate estimate of the model parameters and then carry out the policy update using the current aggregated estimates. The updated policy is used to generate next trajectory. A trial consists of generating a fixed number of such trajectories (500 in our case). Each trial starts

with no a-priori knowledge of the system, i.e., no transition information. Results have been averaged for 10 trials for comparison between TMPI and other algorithms.

The basis of comparison being:

1. Data efficiency, i.e., minimum number of robot transitions needed to learn robust behavior or no negative result.
2. Degree of optimality of the optimal policy discovered, i.e., average number of robot steps to achieve the goal (averaged over 25 trajectories).

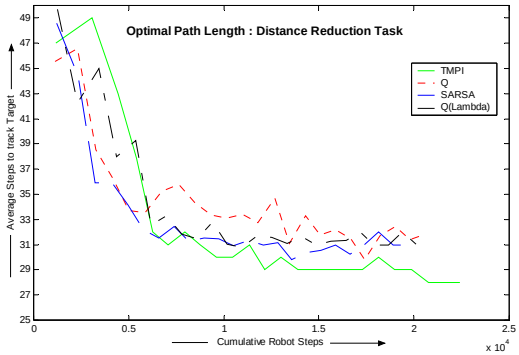

Fig. 4 Algorithm Comparison: Degree of Optimality of discovered policy for the Distance Reduction task

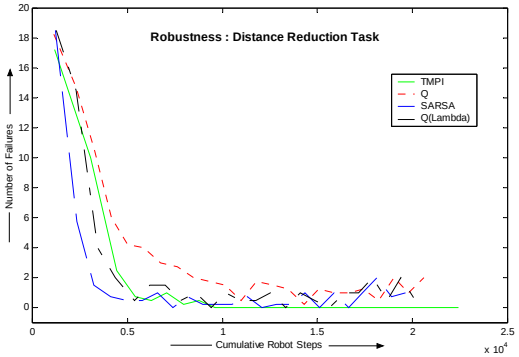

Fig. 5 Algorithm Comparison: Robustness of discovered policy for the Distance Reduction task

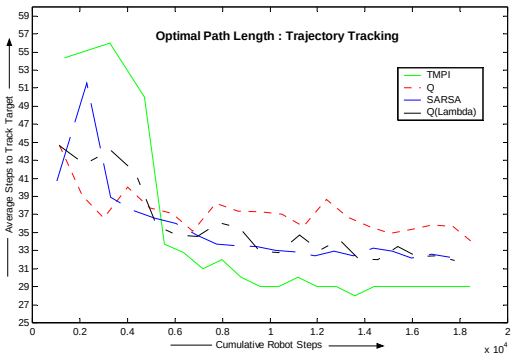

Fig. 6 Algorithm Comparison: Degree of Optimality of discovered policy for the Trajectory Tracking task

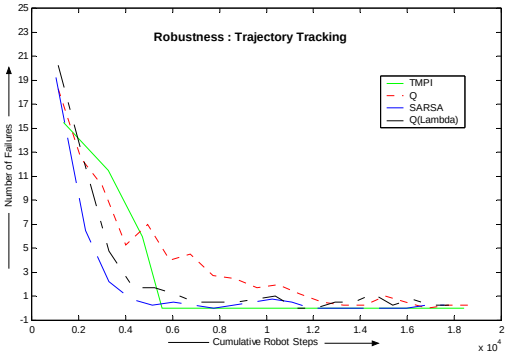

Fig. 7 Algorithm Comparison: Robustness of discovered policy for the Trajectory Tracking task

TABLE I
DISTANCE REDUCTION TASK: PERFORMANCE COMPARISON

Algorithm	Min steps	Avg. steps (min steps)	Avg. steps (robust policy)
TMPI	32.5	5407	7081
Q	34.5	10986	Not Achieved
$Q(\lambda)$	32	7002	Not Achieved
SARSA(λ)	31.5	6567	Not Achieved

TABLE II
TRAJECTORY TRACKING TASK: PERFORMANCE COMPARISON

Algorithm	Min steps	Avg. steps (Min steps)	Avg. steps (Robust policy)
TMPI	32	5557	4714
Q	38	9584	Not Achieved
$Q(\lambda)$	35	9562	Not Achieved
SARSA(λ)	34	6902	Not Achieved

TMPI as well as other approaches have been implemented and simulated using MATLAB^{TM}. The discount factor α is set to 0.9 as in [3], λ is set to 0.6 and simulation step-time T is taken as 0.1 sec for all the trials. The results as depicted in Tables I, and II have been averaged over 10 trials with each trial consisting of 500 trajectories and each trajectory has 120 simulated steps of the robot.

As can be seen from Fig. 4 to 7 and Tables I to II, TMPI approach discovers the optimal policy with less transition information, i.e., with a smaller number of interactions between robot and environment and the degree of optimality of the policy discovered by TMPI is comparable to those found by the other RL algorithms. With a proper choice of trajectory length the degree of optimality of the policy (minimum average steps to achieve goal) can be further improved upon. Finally, in TMPI implementation high exploration level is required only during the initial stages while in all other algorithms we need to maintain exploration.

IV. Conclusion

This paper presents a new approach to solve sequential stochastic decision-making problems termed as Trajectory based Modified Policy Iteration (TMPI), which uses the simulation of the original system/process to create and update an approximate aggregate Markov Decision Process (MDP) model and cost-to-go values. These approximations of the MDP model parameters and cost-to-go values act as inputs to a modified policy iteration procedure to generate a sequence of improving policies that finally converge to an optimal policy for the aggregated simulated system. Instead of evaluating policy dependent value function $Q^{\mu}(i,a)$ as in the standard "Actor-Critic" framework, TMPI tries to approximate a policy independent value function $Q(i,a)$ to update the policy. As the agent gains more experience by "Agent-Environment" interactions the approximating MDP model becomes a fairly rich and robust sequential optimization model.

The performance of the policies generated by TMPI for a simulated mobile robot steering control problem is compared against other benchmark RL algorithms, i.e., Q Learning, Watkins $Q(\lambda)$ and SARSA(λ). For the robot control task it is observed that TMPI requires less experience to achieve a robust behavior, i.e., no negative results, in comparison to other RL algorithms. TMPI could be applied on high dimensional state spaces by suitable choice of aggregation-disaggregation schemes and/or use of generic function approximtors (Neural Networks) [11] to generalize beyond experienced states. The proposed approach as a solution methodology deserves additional research and testing on more complex domains.

References

[1] D.P. Bertsekas and J.N. Tsitsiklis, Neurodynamic-Programming, Athena Scientific, Belmont MA, 1996.

[2] C.W. Zobel and W.T. Scherer, "Simulation-Based Policy Generation Using Large Scale Markov Decision Processes", *IEEE Transactions on Systems, Man and Cybernetics-Part A*: Systems & Humans, Vol. 31, No-6, November 2001.

[3] Stephan Ten Hagen, "Continuous state-space and *Q* learning for control of Non-linear systems", *Ph. D. Thesis*, Amsterdam University, 2001.

[4] C.G. Atkenson and J.C. Santamaria, "A Comparison of Direct and Model-Based Reinforcement Learning", Tech. Rep. GA 30332-0280, College of Computing, Georgia Institute of Technology, Atlanta.

[5] J.N Tsitsiklis, "On the Convergence of Optimistic Policy Iteration", *Journal of Machine Learning Research* 3 , pp. 59-72, 2002.

[6] Leonid Kuvayev, "Model-Based Reinforcement Learning with an Approximate Learned Model", *Master's Thesis*, Department of Computer Science, University of Massachusetts, 1997.

[7] L.P. Kaelbling, M.L. Littman and A.W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research* 4, pp. 237-285,1996.

[8] R. Fierro and F.L.Lewis, "Control of a Non-holonomic Mobile Robot using Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 9, No.4, July 1998.

[9] A.G. Barto, S.J. Bradtke and S.P. Singh, "Learning to Act Using Real-Time Dynamic Programming", *Artificial Intelligence*, 72(1), pp. 81-138, 1995.

[10] S.P. Singh & R.S. Sutton, " Reinforcement Learning with Replacing Eligibility Traces", *Machine Learning*, 22, pp.23-158, 1996.

[11] J.N. Tsiksiklis and B.V. Roy, " An Analysis of Temporal Difference Learning with Function Approximation", *IEEE Transactions on Automatic Control* 42(5), pp. 674- 690,1997.

[12] C.J.C.H. Watkins and P.Dayan, " Technical Note: *Q*-Learning", Machine *Learning*, 8(3/4), pp. 279-292, 1992.

[13] R.S.Sutton, " Learning to predict by the Methods of Temporal Differences", *Machine Learning*, 3, pp. 9-44,1988.

[14] C.J.C.H.Watkins, " Learning from Delayed Rewards", *Ph.D. Thesis*, Cambridge University, Cambridge, England, 1989.

[15] R.S.Sutton & A.G.Barto, " Reinforcement Learning: An introduction", *MIT Press*, Cambridge, Massachusetts, 1998.

[16] R.E.Bellman, "Dynamic Programming", Princeton University Press, *Princeton NJ*, 1957.

[17] P.Dayan, "The convergence of TD(λ) for general λ", Machine Learning, 8, pp. 341- 362,1992.