Lagrange Dual Decomposition for Finite Horizon Markov Decision Processes

Thomas Furmston and David Barber

Department of Computer Science,
University College London,
Gower Street, London, WC1E 6BT, UK

Abstract. Solving finite-horizon Markov Decision Processes with stationary policies is a computationally difficult problem. Our dynamic dual decomposition approach uses Lagrange duality to decouple this hard problem into a sequence of tractable sub-problems. The resulting procedure is a straightforward modification of standard non-stationary Markov Decision Process solvers and gives an upper-bound on the total expected reward. The empirical performance of the method suggests that not only is it a rapidly convergent algorithm, but that it also performs favourably compared to standard planning algorithms such as policy gradients and lower-bound procedures such as Expectation Maximisation.

Keywords: Markov Decision Processes, Planning, Lagrange Duality

1 Markov Decision Processes

The Markov Decision Process (MDP) is a core concept in learning how an agent should act so as to maximise future expected rewards [1]. MDPs have a long history in machine learning and related areas, being used to model many sequential decision making problems in robotics, control and games, see for example [1–3]. Fundamentally, an MDP describes how the environment responds when the agent performs an action $a_t \in \mathcal{A}$ when the environment is in state $s_t \in \mathcal{S}$. More formally, an MDP is described by an initial state distribution $p_1(s_1)$, transition distributions $p(s_{t+1}|s_t, a_t)$, and a reward function $R_t(s_t, a_t)$. For a discount factor γ the reward is defined as $R_t(s_t, a_t) = \gamma^{t-1}R(s_t, a_t)$ for a stationary reward $R(s_t, a_t)$, where $\gamma \in [0, 1)$. At the t^{th} time-point a decision is made according to the policy π_t, which is defined as a set of conditional distributions over the action space,

$$\pi_t(a|s) = p(a_t = a|s_t = s, \pi_t).$$

A policy is called stationary if it is independent of time, *i.e.* $\pi_t(a_t|s_t) = \pi(a|s)$, $\forall t \in \{1, \dots, H\}$ for some policy π. For planning horizon H and discrete states and actions, the total expected reward (the policy utility) is given by

$$U(\pi_{1:H}) = \sum_{t=1}^{H} \sum_{s_t, a_t} R_t(s_t, a_t) p(s_t, a_t|\pi_{1:t}) \tag{1}$$

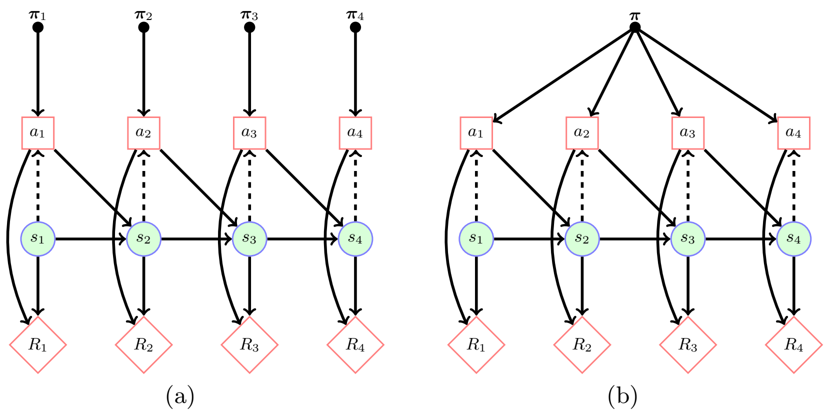

Fig. 1. (a) An influence diagram representation of an unconstrained finite horizon ($H = 4$) MDP. Rewards depend on the state and action, $R_t(s_t, a_t)$. The policy $p(a_t|s_t, \pi_t)$ determines the decision and the environment is modeled by the transition $p(s_{t+1}|s_t, a_t)$. Based on a history of actions, states and reward, the task is maximize the expected summed rewards with respect to the policy $\boldsymbol{\pi}_{1:H}$. (b) For a stationary policy there is a single policy $\boldsymbol{\pi}$ that determines the actions for all time-points, which adds a large clique to the influence diagram.

where $p(s_t, a_t|\pi_{1:t})$ is the marginal of the joint state-action trajectory distribution

$$p(s_{1:H}, a_{1:H}|\pi) = p(a_H|s_H, \pi_H)\left\{\prod_{t=1}^{H-1} p(s_{t+1}|s_t, a_t)p(a_t|s_t, \pi_t)\right\}p_1(s_1). \quad (2)$$

Given a MDP the learning problem is to find a policy π (in the stationary case) or set of policies $\pi_{1:H}$ (in the non-stationary case) that maximizes (1). That is, we wish to find

$$\pi^*_{1:H} = \underset{\pi_{1:H}}{\text{argmax}}\, U(\pi_{1:H}).$$

In the case of infinite horizon $H = \infty$, it is natural to assume a stationary policy, and many classical algorithms exist to approximate the optimal policy [1].

In this paper we concentrate exclusively on the finite horizon case, which is an important class of MDPs with many practical applications. We defer the application of dual decomposition techniques to the infinite horizon case to another study.

1.1 Non-stationary Policies

It is well known that an MDP with a non-stationary policy can be solved using dynamic programming [3] in $O\left(S^2AH\right)$ for a number of actions $A = |\mathcal{A}|$, states $S = |\mathcal{S}|$, and horizon H, see algorithm(1). This follows directly from the linear

Algorithm 1 Dynamic Programming non-stationary MDP solver.

```
$\beta_H(s_H, a_H) = R_H(s_H, a_H)$
**for** $t = H-1, \ldots, 1$ **do**
    $a^*_{t+1}(s_{t+1}) = \underset{a_{t+1}}{\text{argmax}}\ \beta_{t+1}(s_{t+1}, a_{t+1})$
    $\beta_t(s_t, a_t) \equiv R_t(s_t, a_t) + \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t)\beta_{t+1}(s_{t+1}, a^*_{t+1}(s_{t+1}))$
**end for**
$a^*_1(s_1) = \underset{a_1}{\text{argmax}}\ \beta_1(s_1, a_1)$
The optimal policies are deterministic with $\pi^*_t(a_t|s_t) = \delta\left(a_t - a^*_t(s_t)\right)$
```

chain structure of the corresponding influence diagram fig(1a) [4]. The optimal policies resulting from this procedure are deterministic, where for each given state all the mass is placed on a single action.

1.2 Stationary Policies

While unconstrained finite-horizon MDPs can be solved easily through dynamic programming[1] this is not true when the policy is constrained to be stationary. For stationary policies the finite horizon MDP objective is given by

$$U(\pi) = \sum_{t=1}^{H} \sum_{s_t, a_t} R_t(s_t, a_t) p(s_t, a_t|\pi), \tag{3}$$

where $p(s_t, a_t|\pi)$ is the marginal of the trajectory distribution, which is given by

$$p(s_{1:t}, a_{1:t}|\pi) = \pi(a_t|s_t) \left\{ \prod_{\tau=1}^{t-1} p(s_{\tau+1}|a_\tau, s_\tau)\pi(a_\tau|s_\tau) \right\} p_0(s_1).$$

Looking at the influence diagram of the stationary finite horizon MDP problem fig(1b) it can be seen that restricting the policy to be stationary causes the influence diagram to lose its linear chain structure. Indeed the stationary policy couples all time-points together and a dynamic programming solution no longer exists, making the problem of finding the optimal policy π^* much more complex. Another way of viewing the complexity of stationary policy MDPs is in terms of Bellman's *principal of optimality* [3]. A control problem is said to satisfy the principal of optimality if, given a state and time-point, the optimal action is independent of the trajectory preceding that time-point. In order to construct a dynamic programming algorithm it is essential that the control problem satisfies this principal. It is easy to see that while MDPs with non-stationary policies satisfy this principal of optimality, hence permitting a dynamic programming solution, this is not true for stationary policy MDPs.

[1] In practice, the large state-action space can make the $O\left(S^2AH\right)$ complexity impractical, an issue that is not addressed in this study.

While it is no longer possible to exactly solve stationary policy finite-horizon MDP's using dynamic programming there still exist algorithms that can approximate optimal policies, such as policy gradients [5] or Expectation Maximisation (EM) [6, 7]. However, policy gradients is susceptible to to getting trapped in local optima while EM can have poor convergence rate [8]. It is therefore of theoretical, and practical, interest to construct an algorithm that approximately solves this class of MDPs and doesn't suffer from these problems.

2 Dual Decomposition

Our task is to solve the stationary policy finite-horizon MDP. Here our intuition is to exploit the fact that solving the unconstrained (non-stationary) MDP is easy, whilst solving the constrained MDP is difficult. To do so we use dual decomposition and iteratively solve a series of unconstrained MDPs, which have a modified non-stationary reward structure, until convergence. Additionally our dual decomposition provides an upper bound on the optimal reward $U(\pi^*)$.

The main idea of dual decomposition is to separate a complex primal optimisation problem into a set of easier *slave* problems (see appendix(A) and *e.g.* [9, 10]). The solutions to these slave problems are then coordinated to generate a new set of slave problems in a process called the *master* problem. This procedure is iterated until some convergence criterion is met, at which point a solution to the original primal problem is obtained.

As we mentioned in section(1) the stationary policy constraint results in a highly connected influence diagram which is difficult to optimise. It is therefore natural to decompose this constrained MDP by relaxing this stationarity constraint. This can be achieved through Lagrange relaxation where the set of non-stationary policies, $\pi_{1:H} = (\pi_1, \ldots, \pi_H)$, is adjoined to the objective function

$$U(\pi_{1:H}, \pi) = \sum_{t=1}^{H} \sum_{s_t, a_t} R(s_t, a_t) p(s_t, a_t | \pi_{1:t}), \tag{4}$$

with the additional constraint that $\pi_t = \pi$, $\forall t \in \{1, ..., H\}$. Under this constraint, we see that (4) reduces to the primal objective (3). We also impose that all π_t and π are restricted to the probability simplex. The constraints that each π_t is a distribution will be imposed explicitly during the slave problem optimisation. The stationary policy π will be constrained to be a distribution through the equality constraints $\pi = \pi_t$.

2.1 Naive Dual Decomposition

A naive procedure to enforce the constraints that $\pi^t = \pi$, $\forall t \in \{1, .., H\}$ is to form the Lagrangian

$$U(\lambda_{1:H}, \pi_{1:H}, \pi) = \sum_{t=1}^{H} \sum_{s,a} \left\{ R_t(s, a) p_t(s, a | \pi_{1:t}) + \lambda_t(s, a) \left[\pi_t(a|s) - \pi(a|s) \right] \right\},$$

where we have used the notation $p_t(s, a|\pi_{1:t-1}) \equiv p(s_t = s, a_t = a|\pi_{1:t-1})$, and we have included the Lagrange multipliers, $\lambda_{1:H}$.

To see that this Lagrangian cannot be solved efficiently for $\pi_{1:H}$ through dynamic programming consider the optimisation over π_H, which takes the form

$$\max_{\pi_H} \sum_{s,a} \Big\{ R_H(s,a)\pi_H(a|s)p_H(s|\pi_{1:H-1}) + \lambda_H(s,a)\pi_H(a|s) \Big\}.$$

As $p_H(s|\pi_{1:H-1})$ depends on all previous policies this slave problem is computationally intractable and does not have a simple structure to which dynamic programming may be applied. Note also that, whilst the constraints are linear in the policies, the marginal $p(s_t, a_t|\pi_{1:t})$ is non-linear and no simple linear programme exists to find the optimal policy. One may consider the Jensen inequality

$$\log p_t(s|\pi_{1:t-1}) \geq H(q) + \sum_{\tau=1}^{t} \langle \log p(s_\tau|s_{\tau-1}, \pi_\tau)\rangle_q + \langle \log \pi_\tau \rangle_q$$

for variational q and entropy function $H(q)$ (see for example [11]) to decouple the policies, but we do not pursue this approach here, seeking a simpler alternative. From this discussion we can see that the naive application of Lagrange dual decomposition methods does not result in a set of tractable slave problems.

3 Dynamic Dual Decomposition

To apply dual decomposition it is necessary to express the constraints in a way that results in a set of tractable slave problems. In section(2.1) we considered the naive constraint functions

$$g_t(a, s, \pi, \pi_t) = \pi_t(a|s) - \pi(a|s), \tag{5}$$

which resulted in an intractable set of slave problems. We now consider the following constraint functions

$$h_t(a, s, \pi, \pi_{1:t}) = g_t(a, s, \pi, \pi_t)p_t(s|\pi_{1:t-1}).$$

Provided $p_t(s|\pi_{1:t-1}) > 0$, the zeros of the two sets of constraint functions, $g_{1:H}$ and $h_{1:H}$, are equivalent[2]. Adjoining the constraint functions $h_{1:H}$ to the objective function (4) gives the Lagrangian

$$L(\lambda_{1:H}, \pi_{1:H}, \pi) = \sum_{t=1}^{H} \sum_{s,a} \{ (R_t(s,a) + \lambda_t(s,a))\, \pi_t(a|s)p_t(s|\pi_{1:t-1}) \\ - \lambda_t(s,a)\pi(a|s)p_t(s|\pi_{1:t-1}) \} \tag{6}$$

[2] In the case that $p_t(s|\pi_{1:t-1}) = 0$, the policy $\pi_t(a|s)$ is redundant since the state s cannot be visited at time t.

We can now eliminate the original primal variables π from (6) by directly performing the optimisation over π, giving the following set of constraints

$$\sum_t \lambda_t(s,a)p_t(s|\pi_{1:t-1}) = 0, \quad \forall(s,a) \in \mathcal{S} \times \mathcal{A}. \tag{7}$$

Once the primal variables π are eliminated from (6) we obtain the dual objective function

$$L(\lambda_{1:H}, \pi_{1:H}) = \sum_{t=1}^{H}\sum_{s,a}\Big\{(R_t(s,a) + \lambda_t(s,a))\pi_t(a|s)p_t(s|\pi_{1:t-1})\Big\}, \tag{8}$$

where the domain is restricted to $\pi_{1:H}, \lambda_{1:H}$ that satisfy (7) and $\pi_{1:H}$ satisfying the usual distribution constraints.

3.1 The Slave Problem

We have now obtained a dual decomposition of the original constrained MDP. Looking at the form of (8) and considering the Lagrange multipliers $\lambda_{1:H}$ as fixed, the optimisation problem over $\pi_{1:H}$ is that of a standard non-stationary MDP. Given the set of Lagrange multipliers, $\lambda_{1:H}$, we can define the corresponding slave MDP problem

$$U_\lambda(\pi_{1:H}) = \sum_{t=1}^{H}\sum_{s,a}\tilde{R}_t(a,s)\pi_t(a|s)p_t(s|\pi_{1:t-1}), \tag{9}$$

where the slave reward is given by

$$\tilde{R}_t(a,s) = R_t(a,s) + \lambda_t(a,s). \tag{10}$$

The set of slave problems $\max_{\pi_{1:H}} U_\lambda(\pi_{1:H})$ is then readily solved in $O\left(AS^2H\right)$ time using algorithm(1) for modified rewards $\tilde{R}_t$.

3.2 The Master Problem

The master problem consists of minimising the Lagrangian upper bound *w.r.t.* the Lagrange multipliers. From the general theory of Lagrange duality, the dual is convex in λ, so that a simple procedure such as a subgradient method should suffice to find the optimum. At iteration i, we therefore update

$$\lambda_t^{i+1} = \lambda_t^i - \alpha_i \partial_{\lambda_t} L(\pi_{1:H}, \lambda_{1:H}) \tag{11}$$

where α_i is the i^{th} step size parameter and $\partial_{\lambda_t} L(\pi_{1:H}, \lambda_{1:H})$ is the subgradient of the dual objective *w.r.t.* λ_t. As the subgradient contains the factor $p_t(s|\pi_{1:t}^{i-1})$, which is positive and independent of the action, we may consider the simplified update equation

$$\lambda_t^{i+1} = \lambda_t^i - \alpha_i \pi_t^i.$$

Once the Lagrange multipliers have been updated through this subgradient step they need to be projected down on to the feasible set, which is defined through the constraint (7). This is achieved through a projected subgradient method, see *e.g.* [12, 13]. We enforce (7) through the projection

$$\lambda_t^{i+1}(s,a) \leftarrow \lambda_t^{i+1}(s,a) - \sum_{\tau=1}^{H} \rho_\tau(s) \lambda_\tau^{i+1}(s,a), \tag{12}$$

where we define the state-dependent time distributions

$$\rho_\tau(s) \equiv \frac{p_\tau(s|\pi_{1:\tau-1})}{\sum_{\tau'=1}^{H} p_{\tau'}(s|\pi_{1:\tau'-1})}. \tag{13}$$

One may verify this ensures the projected λ_t^{i+1} satisfy the constraint (7).

3.3 Algorithm Overview

We now look at two important aspects of the dual decomposition algorithm; obtaining a primal solution from a dual solution and interpreting the role the Lagrange multipliers play in the dual decomposition.

Obtaining a Primal Solution - A standard issue with dual decomposition algorithms is obtaining a primal solution once the algorithm has terminated. When strong duality holds, *i.e.* the duality gap is zero, then $\pi = \pi_t \ \forall t \in \{1, \ldots, H\}$ and a solution to the primal problem can be obtained from the dual solution. However, this will not generally be the case and we therefore need to specify a way to obtain a primal solution. We considered two approaches; in the first we take the mean of the dual solutions

$$\pi(a|s) = \frac{1}{H} \sum_{t=1}^{H} \pi_t(a|s), \tag{14}$$

while in the second we optimise the dual objective function *w.r.t.* π to obtain the dual primal policy $\pi(a|s) = \delta_{a,a^*(s)}$, where

$$a^*(s) = \operatorname*{argmin}_a \sum_t \lambda_t(s,a) p_t(s|\pi_{1:t-1}), \tag{15}$$

and the λ_t are taken before projection.

A summary of the complete dynamic dual decomposition procedure is given in algorithm(2).

Algorithm 2 Dual Decomposition Dynamic Programming

```
Initialize the Lagrange multipliers $\lambda = 0$.
repeat
    Solve Slave Problem: Solve the finite horizon MDP with non-stationary rewards
        $\tilde{R}_t(s,a) = \lambda_t^i(s,a) + R_t(s,a),$
    using algorithm(1), to obtain the optimal non-stationary policies $\pi_{1:H}^i$.
    Subgradient Step: Update the Lagrange multipliers:
        $\lambda_t^{i+1} = \lambda_t^i - \alpha_i \pi_t^i.$
    Projected Subgradient Step: Project the Lagrange multipliers to the feasible set:
        $\lambda_t^{i+1} \leftarrow \lambda_t^{i+1} - \sum_{\tau=1}^{H} \rho_\tau \lambda_\tau^{i+1}.$
until $|U(\pi) - L(\lambda_{1:H}, \pi_{1:H})| < \epsilon$, for some convergence threshold $\epsilon$.
Output a feasible primal solution $\pi(a|s)$.
```

Interpretation of the Lagrange Multipliers - We noted earlier that the slave problems correspond to an unconstrained MDP problem with non-stationary rewards given by (10). The Lagrange multiplier $\lambda_t(a,s)$ therefore either encourages, or discourages, π_t to perform action a (given state s) depending on the sign of $\lambda_t(a,s)$. Now consider how the Lagrange multiplier $\lambda_t(s,a)$ gets updated at the i^{th} iteration of the dual decomposition algorithm. Prior to the projected subgradient step the update of Lagrange multipliers takes the form

$$\lambda_t^{i+1}(s,a) = \lambda_t^i(s,a) - \alpha_i \pi_t^i(a|s),$$

where $\pi_{1:H}^i$ denotes the optimal non-stationary policy of the previous round of slave problems. Noting that the optimal policy is deterministic gives

$$\lambda_t^{i+1}(s,a) = \begin{cases} \lambda_t^i(s,a) - \alpha_i & \text{if } a = \underset{a}{\text{argmax}}\ \pi_t^i(a|s) \\ \lambda_t^i(s,a) & \text{otherwise.} \end{cases}$$

Once the Lagrangian multipliers are projected down to the space of feasible parameters through (12) we have

$$\lambda_t^{i+1}(s,a) = \begin{cases} \bar{\lambda}_t^i(s,a) + \alpha_i \big(\sum_{\tau \in N_a^i(s)} \rho_\tau - 1 \big), & \text{if } a = \underset{a}{\text{argmax}}\ \pi_t^i(a|s) \\ \bar{\lambda}_t^i(s,a) + \alpha_i \sum_{\tau \in N_a^i(s)} \rho_\tau & \text{otherwise} \end{cases}$$

where $N_a^i(s) = \big\{ t \in \{1, ..., H\} | \pi_t^i(a|s) = 1 \big\}$ is the set of time-points for which action a was optimal in state s in the last round of slave problems. We use the notation $\bar{\lambda}_t^i = \lambda_t^i - \langle \lambda_t^i \rangle_\rho$, where $\langle \cdot \rangle_\rho$ means taking the average *w.r.t.* to the distribution (13).

Noting that ρ_t is a distribution over t means that the terms $\sum_{\tau \in N_a^i(s)} \rho_\tau$ and $\left(\sum_{\tau \in N_a^i(s)} \rho_\tau - 1\right)$ are positive and negative respectively. The projected sub-gradient step therefore either adds (or subtracts) a positive term to the projection, $\bar{\lambda}$, depending on the optimality of action a (given state s at time t) in the slave problem from the previous iteration. There are hence two possibilities for the update of the Lagrange multiplier; either the action was optimal and a lower non-stationary reward is allocated to this state-action-time triple in the next slave problem, or conversely it was sub-optimal and a higher reward term is attached to this triple. The master algorithm therefore tries to readjust the Lagrange multipliers so that (for each given state) the same action is optimal for all time-points, *i.e.* it encourages the non-stationary policies to take the same form. Additionally, as $|N_a^i(s)| \to H$ then $\sum_{\tau \in N_a^i(s)} \rho_\tau \to 1$, which means that a smaller quantity is added (or subtracted) to the Lagrange multiplier. The converse happens in the situation $|N_a^i(s)| \to 0$. This means that as $|N_a^i(s)| \to 1$ the time-points $t \notin N_a^i(s)$ will have a larger positive term added to the reward for this state-action pair, making it more likely that this action will be optimal given this state in the next slave problem. Additionally, those time-points $t \in N_a^i(s)$ will have a smaller term subtracted from their reward, making it more likely that this action will remain optimal in the next slave problem. The dual decomposition algorithm therefore automatically weights the rewards according to a 'majority vote'. This type of behaviour is typical of dual decomposition algorithms and is known as resource allocation via pricing [12].

4 Experiments

We ran our dual decomposition algorithm on several benchmark problems, including the *chain* problem [14], the *mountain car* problem [1] and the *puddle world* problem [15]. For comparison we included planning algorithms that can also handle stationarity constraints on the policy, in particular Expectation Maximisation and Policy Gradients.

Dual Decomposition Dynamic Programming (DD DP)

The overall algorithm is summarised in algorithm(2) in which dynamic programming is used to solve the slave problems. In the master problem we used a predetermined sequence of step sizes for the subgradient step. Taking into account the discussion in section(3.3) we used the following step sizes

$$\alpha_n = n^{-1} \max_{s,a} R(s,a).$$

In the experiments we obtained the primal policy using both the time-averaged policy (14) and the dual primal policy (15). We found that both policies obtained a very similar level of performance and so we show only the results of (14) to make the plots more readable. We declared that the algorithm had converged when the duality gap was less than 0.01.

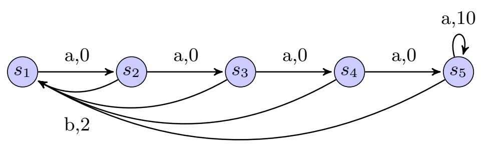

Fig. 2. The chain problem state-action transitions with rewards $R(s_t, a_t)$. The initial state is state 1. There are two actions a, b, with each action being flipped with probability 0.2.

Expectation Maximisation (EM)

The first comparison we made was with the MDP EM algorithm [6, 16, 17, 11, 7] where the policy update at each M step takes the form

$$\pi^{\text{new}}(a|s) \propto \sum_{t=1}^{H} \sum_{\tau=1}^{t} q(s_\tau = s, a_\tau = a, t),$$

and q is a reward weighted trajectory distribution. For a detailed derivation of this finite horizon model based MDP EM algorithm see *e.g.* [11, 7].

Policy Gradients (PG) - Fixed Step Size.

In this algorithm updates are taken in the direction of the gradient of the value function *w.r.t.* to the policy parameters. We parameterised the policies with a *softmax* parameterisation

$$\pi(a|s) = \frac{\exp \gamma(s, a)}{\sum_{a'} \exp \gamma(s, a')}, \qquad \gamma(s, a = 1) = 0, \tag{16}$$

and took the derivative *w.r.t.* the parameters $\boldsymbol{\gamma}$. During the experiments we used a predetermined step size sequence for the gradient steps. We considered two different step sizes parameters

$$\alpha_1(n) = n^{-1}, \quad \alpha_2(n) = 10n^{-1},$$

and selected the one that gave the best results for each particular experiment.

Policy Gradients (PG) - Line Search

We also ran the policy gradients algorithm (using the parameterisation (16)) with a line search procedure during each gradient step[3].

Expectation Maximisation - Policy Gradients (EM PG)

The algorithm was designed to accelerate the rate of convergence of the EM algorithm, while avoiding issues of local optima in policy gradients [18]. The algorithm begins by performing EM steps until some switching criterion is met, after which policy gradient updates are used.

[3] We used the conjgrad.m routine in the Netlab toolbox - `http://www1.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/`.

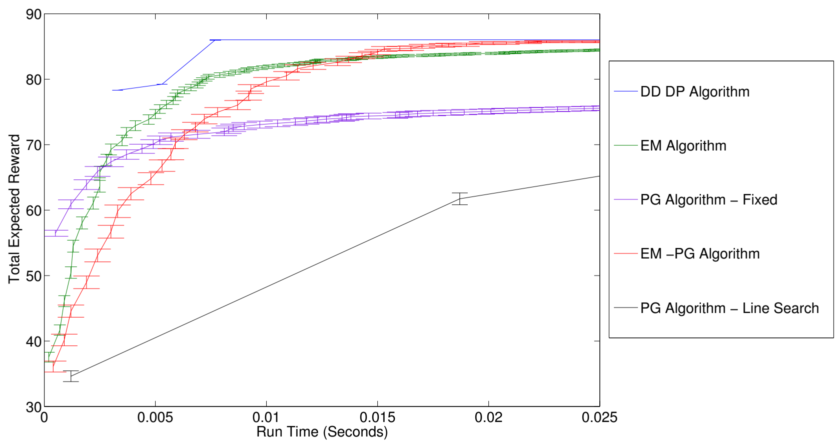

Fig. 3. Chain experiment with total expected reward plotted against run time (in seconds). The plot shows the results of the DD DP algorithm (blue), the EM algorithm (green), policy gradients with fixed step size (purple), policy gradients with a line search (black) and the switching EM-PG algorithm (red). The experiment was repeated 100 times and the plot shows the mean and standard deviation of the results.

There are different criteria for switching between the EM algorithm and policy gradients. In the experiments we use the criterion given in [19] where the EM iterates are used to approximate the amount of hidden data (and an additional estimate of the error of $\hat{\lambda}$)

$$\hat{\lambda} = \frac{|\pi^{(t+1)} - \pi^{(t)}|}{|\pi^{(t)} - \pi^{(t-1)}|}, \qquad \hat{\epsilon} = \frac{|\pi^{(t+1)} - \pi^{(t)}|}{1 + |\pi^{(t+1)}|}. \tag{17}$$

In the experiment we switched from EM to policy gradients when $\hat{\lambda} > 0.9$ and $\hat{\epsilon} < 0.01$. During policy gradients steps we used fixed step size parameters.

4.1 Chain Problem

The *chain* problem [14] has 5 states each having 2 possible actions, as shown in fig(2). The initial state is 1 and every action is flipped with 'slip' probability $p_{\text{slip}} = 0.2$, making the environment stochastic. If the agent is in state 5 it receives a reward of 10 for performing action 'a', otherwise it receives a reward of 2 for performing action 'b' regardless of the state. In the experiments we considered a planning horizon of $H = 25$ for which the optimal stationary policy is to travel down the chain towards state 5, which is achieved by always selecting action 'a'.

The results of the experiment are shown in fig(3). We can see that the DD DP algorithm consistently outperforms the other algorithms, converging to the

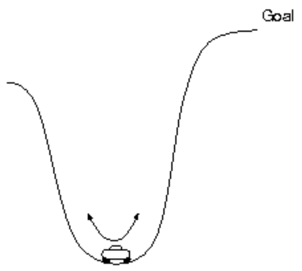

Fig. 4. A graphical illustration of the mountain car problem. The agent (driver) starts the the problem at the bottom of a valley in a stationary position. The aim of the agent is to get itself to the right most peak of the valley.

global optimum in an average of 3 iterations. On the other hand the policy gradient algorithms often got caught in local optima in this problem, which is because the gradient of the initial policy is often in the direction of the local optima around state 1. The EM and EM-PG algorithms perform better than policy gradients, being less susceptible to local optima. Additionally, they were able to get close to the optimum in the time considered, although neither of these algorithms actually reached the optimum.

4.2 Mountain Car Problem

In the *mountain car* problem the agent is driving a car and its state is described by its current position and velocity, denoted by $x \in [-1, 1]$ and $v \in [-0.5, 0.5]$ respectively. The agent has three possible actions, $a \in \{-1, 0, 1\}$, which correspond to *reversing*, *stopping* and *accelerating* respectively. The problem is depicted graphically in fig(4) where it can be seen that the agent is positioned in a valley. The leftmost peak of the valley is given by $x = -1$, while the rightmost peak is given by $x = 1$. The continuous dynamics are nonlinear and are given by

$$v^{\text{new}} = v + 0.1a - 0.0028\cos(2x - 0.5), \qquad x^{\text{new}} = x + v^{\text{new}}.$$

At the start of the problem the agent is in a stationary position, *i.e.* $v = 0$, and its position is $x = 0$. The aim of the agent is to maneuver itself to the rightmost peak, so the reward is set to 1 when the agent is in the rightmost position and 0 otherwise. In the experiment we discretised the position and velocity ranges into bins of width 0.1, resulting in $S = 231$ states. A planning horizon $H = 25$ was sufficient to reach the goal state.

As we can see from fig(5) the conclusions from this experiment are similar to those of the chain problem. Again the DD DP algorithm consistently outperformed all of the comparison algorithms, converging to the global optimum in roughly 7 iterations, while the policy gradients algorithms were again susceptible to local optima. The difference in convergence rates between the DD DP algorithm and both the EM and EM-PG algorithms is more pronounced here, which is due to an increase in the amount of hidden data in this problem.

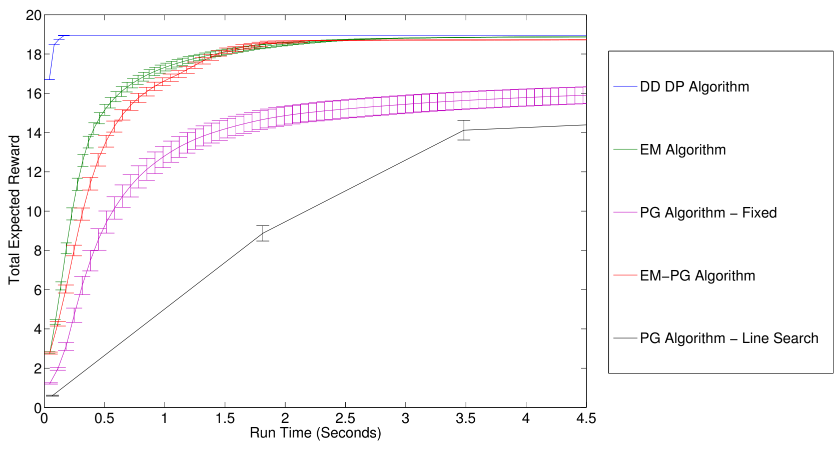

Fig. 5. Mountain car experiment with total expected reward plotted against run time (in seconds). The plot shows the results for the DD DP algorithm (blue), the EM algorithm (green), policy gradients with fixed step size (purple), policy gradients with a line search (black) and the switching EM-PG algorithm (red). The experiment was repeated 100 times and the plot shows the mean and standard deviations of the experiments.

4.3 Puddle World

In the *puddle world* problem [15] the state space is a continuous 2-dimensional grid $(x, y) \in [0, 1]^2$ that contains two puddles. We considered two circular puddles (of radius 0.1) where the centres of the puddles were generated uniformly at random over the grid $[0.2, 0.8]^2$. The agent in this problem is a robot that is depicted by a point mass in the state space. The aim of the robot is to navigate itself to a goal region, while avoiding areas of the state space that are covered in puddles. The initial state of the robot was set to the point $(0, 0)$. There are four discrete actions (*up*, *down*, *left* and *right*) each moving the robot 0.1 in that direction. The dynamics where made stochastic by adding the Gaussian noise $\mathcal{N}(0, 0.01)$ to each direction. A reward of 1 is received for all states in the goal region, which is given by those states satisfying $x + y \geq 1.9$. A negative reward of $-40(1 - d)$ is received for all states inside a puddle, where d is the distance from the centre of the puddle. In the experiment we discretised the x and y dimensions into bins of width 0.05, which gave a total of $S = 441$ states. In this problem we found that setting the planning horizon to $H = 50$ was sufficient to reach the goal region.

The results of the puddle world experiment are shown in fig(6). For the range of step sizes considered we were unable to obtain any reasonable results for the policy gradients algorithm with fixed step sizes or the EM-PG algorithm, so we omit these from the plot. The policy gradients with line search performed poorly and consistently converged to a local optimum. The DD DP algorithm converged

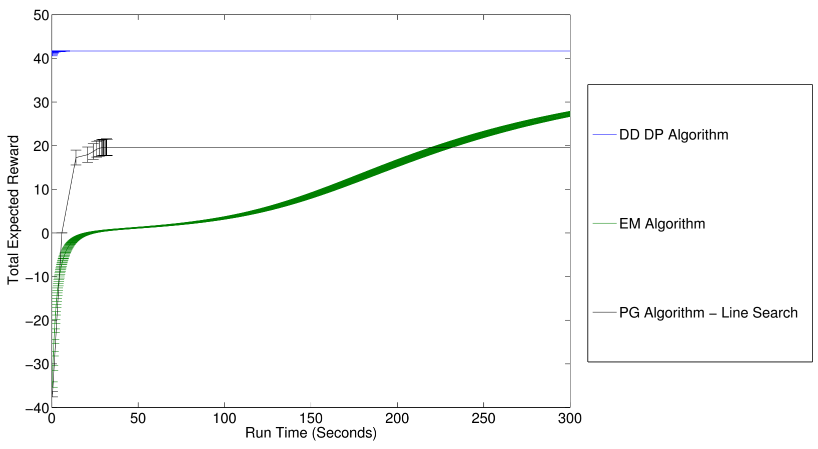

Fig. 6. Puddle world experiment with total expected reward plotted against run time (in seconds). The plot shows the results for the DD DP algorithm (blue), the EM algorithm (green) and policy gradients with line search (black).

after around 7 seconds of computation (this is difficult to see because of the scale needed to include the EM results) which corresponds to around 30 iterations of the algorithm. In this problem the MDP EM algorithm has a large amount of hidden data and as a result has poor convergence, consistently failing to converge to the optimal policy after 1000 EM steps (taking around 300 seconds).

5 Discussion

We considered the problem of optimising finite horizon MDP's with stationary policies. Our novel approach uses dual decomposition to construct a two stage iterative solution method with excellent empirical convergence properties, often converging to the optimum within a few iterations. This compares favourably against other planning algorithms that can be readily applied to this problem class, such as Expectation Maximisation and policy gradients. In future work we would like to consider more general settings, including partially-observable MDPs and problems with continuous state-action spaces. Whilst both of these extensions are non-trivial, this work suggests that the application of Lagrange duality to these cases could be a particularly fruitful area of research.

References

1. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
2. N. Vlassis. A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 1(1):1–71, 2007.
3. D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, second edition, 2000.
4. R.D. Shachter. Probabilistic Inference and Influence Diagrams. *Operations Research*, 36:589–604, 1988.
5. R. Williams. Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.
6. M. Toussaint, A. Storkey, and S. Harmeling. *Bayesian Time Series Models*, chapter Expectation-Maximization Methods for Solving (PO)MDPs and Optimal Control Problems. Cambridge University Press, 2011. In press. See `userpage.fu-berlin.de/~mtoussai`.
7. T. Furmston and D. Barber. Efficient Inference in Markov Control Problems. In *Uncertainty in Artificial Intelligence*. North-Holland, 2011.
8. T. Furmston and D. Barber. An analysis of the Expectation Maximisation algorithm for Markov Decision Processes. Research Report RN/11/13, Centre for Computational Statistics and Machine Learning, University College London, 2011.
9. D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.
10. D. Sontag, A. Globerson, and T. Jaakkola. Introduction to Dual Decomposition for Inference. In S. Sra, S. Nowozin, and S. Wright, editors, *Optimisation for Machine Learning*. MIT Press, 2011.
11. T. Furmston and D. Barber. Variational Methods for Reinforcement Learning. *AISTATS*, 9(13):241–248, 2010.
12. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
13. N. Komodakis, N. Paragios, and G. Tziritas. MRF Optimization via Dual Decomposition: Message-Passing Revisited. In *IEEE 11th International Conference on Computer Vision, ICCV*, pages 1–8, 2007.
14. R. Dearden, N. Friedman, and S. Russell. Bayesian Q learning. *AAAI*, 15:761–768, 1998.
15. R. Sutton. Generalization in Reinforcment Learning: Successful Examples Using Sparse Coarse Coding. *NIPS*, (8):1038–1044, 1996.
16. M. Hoffman, A. Doucet, N. De Freitas, and A. Jasra. Bayesian Policy Learning with Trans-Dimensional MCMC. *NIPS*, (20):665–672, 2008.
17. M. Hoffman, N. de Freitas, A. Doucet, and J. Peters. An Expectation Maximization Algorithm for Continuous Markov Decision Processes with Arbitrary Rewards. *AISTATS*, 5(12):232–239, 2009.
18. R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. *ICML*, (20):672–679, 2003.
19. C. Fraley. On Computing the Largest Fraction of Missing Information for the EM Algorithm and the Worst Linear Function for Data Augmentation. Research Report EDI-INF-RR-0934, University OF Washington, 1999.
20. D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2011.

A Appendix: Dual Decomposition

We follow the description in [20]. A general approach is to decompose a difficult optimisation problem into a set of easier problems. In this approach we first identify tractable 'slave' objectives $E_s(x)$, $s = 1, \ldots, S$ such that the 'master' objective $E(x)$ decomposes as

$$E(x) = \sum_s E_s(x) \tag{18}$$

Then the x that optimises the master problem is equivalent to optimising each slave problem $E_s(x_s)$ under the constraint that the slaves agree $x_s = x$, $s = 1, \ldots, S$ [9]. This constraint can be be imposed by a Lagrangian

$$\mathcal{L}(x, \{x_s\}, \lambda) = \sum_s E_s(x_s) + \sum_s \lambda_s (x_s - x) \tag{19}$$

Finding the stationary point *w.r.t.* x, gives the constraint $\sum_s \lambda_s = 0$, so that we may then consider

$$\mathcal{L}(\{x_s\}, \lambda) = \sum_s E_s(x_s) + \lambda_s x_s \tag{20}$$

Given λ, we then optimise each slave problem

$$x_s^* = \underset{x_s}{\operatorname{argmax}} \ (E_s(x_s) + \lambda_s x_s) \tag{21}$$

The Lagrange dual is given by

$$\mathcal{L}_s(\lambda_s) = \max_{x_s} (E_s(x_s) + \lambda_s x_s) \tag{22}$$

In this case the dual bound on the primal is

$$\sum_s \mathcal{L}_s(\lambda_s) \geq E(x^*) \tag{23}$$

where x^* is the solution of the primal problem $x^* = \underset{x}{\operatorname{argmax}}\, E(x)$. To update λ one may use a projected sub-gradient method to minimise each $\mathcal{L}_s(\lambda_s)$

$$\lambda_s' = \lambda - \alpha x_s^* \tag{24}$$

where α is a chosen positive constant. Then we project,

$$\bar{\lambda} = \frac{1}{S} \sum_s \lambda_s', \qquad \lambda_s^{new} = \lambda_s' - \bar{\lambda} \tag{25}$$

which ensures that $\sum_s \lambda_s^{new} = 0$.