Conditional Random Fields for Multi-agent Reinforcement Learning

Xinhua Zhang XINHUA.ZHANG@ANU.EDU.AU
Douglas Aberdeen DOUG.ABERDEEN@ANU.EDU.AU
S.V. N. Vishwanathan VISHY@MAIL.RSISE.ANU.EDU.AU
Statistical Machine Learning, NICTA
Research School of Information Sciences & Engineering, Australian National University, Canberra, Australia

Abstract

Conditional random fields (CRFs) are graphical models for modeling the probability of labels given the observations. They have traditionally been trained with using a set of observation and label pairs. Underlying all CRFs is the assumption that, conditioned on the training data, the labels are independent and identically distributed (iid). In this paper we explore the use of CRFs in a class of temporal learning algorithms, namely policy-gradient reinforcement learning (RL). Now the labels are no longer iid. They are actions that update the environment and affect the next observation. From an RL point of view, CRFs provide a natural way to model joint actions in a decentralized Markov decision process. They define how agents can communicate with each other to choose the optimal *joint* action. Our experiments include a synthetic network alignment problem, a distributed sensor network, and road traffic control; clearly outperforming RL methods which do not model the proper joint policy.

1. Introduction

Conditional random fields (CRFs) have been studied in batch settings, where parameters are optimized over a training set; and online settings, where parameters are updated after each iid sample is observed. However, there is little work on CRFs for modelling temporal problems such as control or time-series prediction. The reinforcement learning (RL) community, on the other hand, has done work on decentralized (multi-agent) control. RL algorithms optimize a long-term measure of temporally delayed rewards in controlled systems. This paper seeks to improve decentralized RL methods by using CRF models to exploit the structure between agents exhibited in many decentralized RL domains. Examples include sensor networks, traffic routing for roads or networks, pursuer-evader problems, and job-shop scheduling.

Bernstein et al. (2000) proved that the complexity of learning optimal coordination in decentralized RL is generally NEXP-hard in the number of agents. The simplest algorithms assume all agents are independent, learning to cooperate implicitly via an appropriate reward function (Bagnell & Ng, 2006). More advanced algorithms explicitly share information about states, values, or *proposed* actions (Boutilier, 1999), but still avoid modeling the optimal joint policy. Our work is similar to Guestrin et al. (2002), which does model the optimal joint policy, using the underlying structure to factorise Q-values and choose joint actions. In contrast, our approach focuses on directly optimizing a joint probability distribution over preferred actions. Furthermore, we draw on the wealth of approximate inference methods for graphical models, and CRFs in particular, to evaluate and optimize policies that would otherwise be intractable despite the structured representation.

Traditionally, CRFs use batch training algorithms to learn model $p(y|\boldsymbol{x};\boldsymbol{\theta})$, the probability of a label y, conditioned on observable variables $\boldsymbol{x}$ with the CRF parameters $\boldsymbol{\theta}$ (Lafferty et al., 2001). During training we iterate through a set of training instances $(X, Y) := (\{\boldsymbol{x}_i\}_{i=1}^n, \{y_i\}_{i=1}^n)$, finding $\boldsymbol{\theta}^* := \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|X, Y)$. To predict the label for a novel observation $\boldsymbol{x}'$ we select $y' := \arg\max_y p(y|\boldsymbol{x}';\boldsymbol{\theta}^*)$. In this paper, we show that the same inference methods used for CRFs can be used to sample node actions from a *joint* stochastic RL policy. We also show how to optimize this joint policy by estimating the gradients of the long-term reward with respect to the policy parameters. Similar

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

methods could be used for RL policies based on arbitrary graphical models. From the CRF point of view, we propose a method of using CRFs for modeling temporal processes.

Section 2 and Section 3 are devoted to describing graphical models and reinforcement learning respectively, with particular emphasis on CRFs and policy-gradient methods for RL. We then elaborate on the combination of CRF and RL in Section 4. Section 5 describes our experiments before concluding.

2. Conditional Random Fields

CRFs are a probabilistic framework for labeling and segmenting data. Unlike hidden Markov models (HMMs) and Markov random fields (MRFs), which model the joint density $p(\boldsymbol{x}, y)$ over inputs $\boldsymbol{x}$ and labels y, CRFs directly model $p(y|\boldsymbol{x})$ for a given input observation $\boldsymbol{x}$. Furthermore, instead of maintaining a per-state normalisation, which leads to the so-called label bias problem, CRFs use a global normalisation that allows them to take global interactions into account (Lafferty et al., 2001). We now introduce conditional exponential families, and describe CRFs as a special instance of conditional exponential family.

2.1. Conditional Exponential Families

Given observation $\boldsymbol{x} \in \mathcal{X}$, a conditional exponential family (CEF) distribution over labels $y \in \mathcal{Y}$ (we assume all spaces are finite), parameterized by the natural parameter $\boldsymbol{\theta} \in \mathbb{R}^d$, can be written in its canonical form as

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \exp(\langle \phi(\boldsymbol{x}, y), \boldsymbol{\theta} \rangle - g(\boldsymbol{\theta}|\boldsymbol{x})). \quad (1)$$

Here, vector $\phi(\boldsymbol{x}, y)$ is the *sufficient statistics* of the distribution $p(y|\boldsymbol{x};\boldsymbol{\theta})$, $\langle \cdot, \cdot \rangle$ denotes the inner product, and $g(\cdot)$ is the log-partition function for normalization:

$$g(\boldsymbol{\theta}|\boldsymbol{x}) := \ln \sum\nolimits_{y \in \mathcal{Y}} \exp(\langle \phi(\boldsymbol{x}, y), \boldsymbol{\theta} \rangle) \quad (2)$$

It is known that the log-partition function is also the cumulant generating function of the CEF

$$\partial g(\boldsymbol{\theta}|\boldsymbol{x}) / \partial \boldsymbol{\theta} = \mathbb{E}_{p(y|\boldsymbol{x};\boldsymbol{\theta})}[\phi(\boldsymbol{x}, y)]. \quad (3)$$

The sufficient statistics $\phi(\boldsymbol{x}, y)$ (also called potentials) represent salient features of the input observations, and typically depend on the applications and CRF design.

2.2. Clique Decomposition Theorem

More generally, we consider the structured output case where $\boldsymbol{y} \in \mathcal{Y}^m$ (m nodes). The clique decomposition theorem essentially states that if the conditional density $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ factorizes according to a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ on $\boldsymbol{y}$, then the sufficient statistics $\phi(\boldsymbol{x}, \boldsymbol{y})$ decompose into terms over the maximal cliques $\mathcal{C} = \{c_1, \ldots, c_n\}$ of $\mathcal{G}$ (Altun et al., 2004):

$$\phi(\boldsymbol{x}, \boldsymbol{y}) = vec\{\phi_c(\boldsymbol{x}, \boldsymbol{y}_c) | c \in \mathcal{C}\} \quad (4)$$

$$p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) = \exp\left(\sum_{c \in \mathcal{C}} \langle \phi_c(\boldsymbol{x}, \boldsymbol{y}_c), \boldsymbol{\theta}_c \rangle - g(\boldsymbol{\theta}|\boldsymbol{x}) \right), \quad (5)$$

where the *vec* operator concatenates vectors, c indexes the set of maximal cliques $\mathcal{C}$, and $\boldsymbol{y}_c$ is the label configuration for nodes in clique c. For convenience, we will assume that all maximal cliques have size two, *i.e.*, an edge between nodes i and j has a feature ϕ_{ij} associated with it. We will also associate potentials ϕ_i to single nodes i. Node features represent the observation of state available at each node. The edge features encode the communication between nodes about their features and potential actions.

CRFs are examples of conditional exponential families with special graphs. For 1-D CRFs, the graph is a chain, so the edge features are $\phi_{i,i+1}(\boldsymbol{x}, \boldsymbol{y}_i, \boldsymbol{y}_{i+1})$. For 2-D grid CRFs, the edge features are $\phi_{iji'j'}(\boldsymbol{x}, \boldsymbol{y}_{ij}, \boldsymbol{y}_{i'j'})$ where nodes are indexed by double coordinates and $|i - i'| + |j - j'| = 1$.

2.3. Inference and Gradient Computations

CRF training procedures usually minimize the negative log-posterior of the parameters given the observation/label training set. As we will see in Section 3.1, policy-gradient algorithms instead draw samples from (5) given the parameters $\boldsymbol{\theta}$ and most recent observation $\boldsymbol{x}$. This involves computing the log-partition function, which can be done efficiently by exploiting the clique structure

$$\exp(g(\boldsymbol{\theta}|\boldsymbol{x})) = \sum_{\boldsymbol{y} \in \mathcal{Y}^m} \prod_{c \in \mathcal{C}} \exp(\langle \phi_c(\boldsymbol{x}, \boldsymbol{y}_c), \boldsymbol{\theta}_c \rangle). \quad (6)$$

Policy-gradient algorithms also require the gradient of the log probability of sampled labels/actions $\tilde{\boldsymbol{y}}$

$$\begin{aligned} &\frac{\partial}{\partial \boldsymbol{\theta}} \ln\ p(\tilde{\boldsymbol{y}}|\boldsymbol{x};\boldsymbol{\theta}) \\ &= \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{c \in \mathcal{C}} \langle \phi_c(\boldsymbol{x}, \tilde{\boldsymbol{y}}_c), \boldsymbol{\theta}_c \rangle - g(\boldsymbol{\theta}|\boldsymbol{x}) \\ &= vec\{\phi_c(\boldsymbol{x}, \tilde{\boldsymbol{y}}_c) - \mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})}\left[\phi_c(\boldsymbol{x}, \boldsymbol{y}_c)\right] | c \in \mathcal{C}\}, \end{aligned} \quad (7)$$

where the last step exploits (4) and (3). Let z be

the constant normalisation term $\exp(g(\boldsymbol{\theta}|\boldsymbol{x}))$, then the expected sufficient statistics for a fixed clique c' is[1]

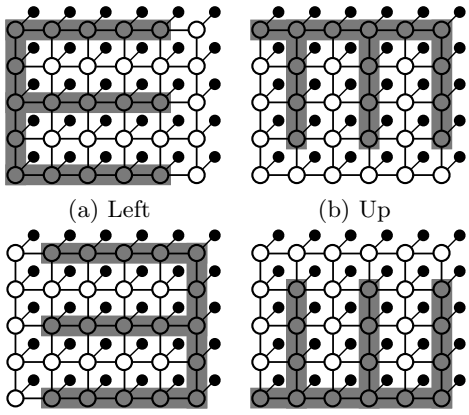
$$\begin{aligned}\mathbb{E}_{p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})}\left[\phi_{c'}(\boldsymbol{x},\boldsymbol{y}_{c'})\right] &= \sum_{\boldsymbol{y}\in\mathcal{Y}^m}\phi_{c'}(\boldsymbol{x},\boldsymbol{y}_{c'})p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) \\ &= z^{-1}\sum_{\boldsymbol{y}\in\mathcal{Y}^m}\phi_{c'}(\boldsymbol{x},\boldsymbol{y}_{c'})\exp\sum_{c\in\mathcal{C}}\langle\phi_c(\boldsymbol{x},\boldsymbol{y}_c),\boldsymbol{\theta}_c\rangle \\ &= z^{-1}\sum_{\boldsymbol{y}\in\mathcal{Y}^m}\prod_{c\in\mathcal{C}}\tilde{\phi}_c^{c'}(\boldsymbol{x},\boldsymbol{y}_c)\exp\langle\phi_c(\boldsymbol{x},\boldsymbol{y}_c),\boldsymbol{\theta}_c\rangle\,, \quad (8)\end{aligned}$$

where $\tilde{\phi}_c^{c'}(\boldsymbol{x},\boldsymbol{y}_c) := \phi_{c'}(\boldsymbol{x},\boldsymbol{y}_{c'})$ if $c = c'$, and a vector of ones otherwise. Note that both (6) and (8) are in the familiar sum-product form, which can be computed exactly by belief propagation (BP) (*e.g.*, Chapter 26 of MacKay, 2003). This has time complexity $O(N\,|\mathcal{Y}|^{w+1})$, where N is the number of nodes and w is the tree width of the graph, *i.e.*, the size of its largest clique minus 1 after the graph is optimally triangulated. For trees and 1-D CRFs (chains), $w = 1$, so that calculating (8) directly is feasible. However, for more general cases like 2-D CRFs (grids), the tree width w is prohibitively high, and one has to resort to approximate approaches, *e.g.*, sampling and variational methods. We now briefly describe one such method used in our experiments.

2.4. Tree MCMC Sampler for CRFs

The tree Markov chain Monte Carlo (MCMC) sampler of Hamze & de Freitas (2004) is a state-of-the-art algorithm for sampling from posterior distributions and computing expectations of sufficient statistics in undirected graphical models with regular structure and high tree width. Its basic form works on pairwise MRFs or CRFs whose cliques are either nodes or edges.

The algorithm exploits the property that MRFs can be split into several disjoint trees (see Figure 1 for four different choices of partitions). Although BP on the whole MRF is prohibitively expensive, it is cheap to run BP on each of the two trees (their tree width $w = 1$). So a natural idea is to combine analytical and sampling steps: conditioned on a sample of one of the trees, use BP to compute the exact joint conditional distribution of the other tree and draw a sample from it; then alternate between the two trees. Moreover, knowing the exact conditional distribution over the trees makes it possible to Rao-Blackwellise the sampler to reduce the variance (Casella & Robert, 1996). Each partition of the tree has to exclude some edges. In order to reduce the variance in the expectation estimates of these edges, and to cover all edges in the graph, we need to partition the graph in several different ways. This leads to the four partitions in Figure 1.



(a) Left (b) Up (c) Right (d) Down

Figure 1: Four different partitions of a 5-by-6 CRF. Nodes in shaded and white regions are the two trees and the small black circles represent observations.

Empirically tree sampling is considerably more efficient than other partition based sampling schemes and the naïve Gibbs sampler, and with provable faster geometric convergence rate and lower variance (Hamze & de Freitas, 2004).

3. Reinforcement Learning

A Markov decision process (MDP) consists of a finite set of states $s \in \mathcal{S}$ of the world, actions $y \in \mathcal{Y}$ available to the agent in each state, and a reward function $r(s)$ for each state s. In a partially observable MDP (POMDP), the controller sees only an observation $\boldsymbol{x} \in \mathcal{X}$ of the current state, sampled stochastically from an unknown distribution $p(\boldsymbol{x}|s)$. Each action y determines a stochastic matrix $\boldsymbol{P}(y) = [p(s'|s,y)]$ of transition probabilities from state s to state s' given action y. The methods discussed in this paper do not assume explicit knowledge of $\boldsymbol{P}(y)$ or of the observation process. All policies are stochastic, with a probability of choosing action y given state s, and parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ of $p(y|\boldsymbol{x};\boldsymbol{\theta})$. The evolution of the state s is Markovian, governed by an $|\mathcal{S}| \times |\mathcal{S}|$ transition probability matrix $\boldsymbol{P}(\boldsymbol{\theta}) = [p(s'|s;\boldsymbol{\theta})]$ with entries

$$p\left(s'|s;\boldsymbol{\theta}\right) = \sum\nolimits_{y\in\mathcal{Y}} p\left(y|s;\boldsymbol{\theta}\right) p\left(s'|s,y\right). \quad (9)$$

We assume an average reward setting where the task is to find a policy, or equivalently the parameter $\boldsymbol{\theta}$, which maximizes

$$R(\boldsymbol{\theta}) := \lim_{T\to\infty}\frac{1}{T}\,\mathbb{E}_{\boldsymbol{\theta}}\left[\sum_{t=0}^{T-1} r(s_t)\right], \quad (10)$$

[1] For notational convenience we assume componentwise multiplication of vectors in the last step of (8).

The expectation $\mathbb{E}_{\boldsymbol{\theta}}$ is over the distribution of state trajectories $\{s_0, s_1, \dots\}$ induced by $\boldsymbol{P}(\boldsymbol{\theta})$.

The core idea of this paper is to treat CRF distributions over labels, $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$, exactly as joint distributions over multi-agent RL actions, *i.e.*, a stochastic policy. Each node in the CRF will represent a single RL agent. The joint stochastic policy will give the probability of a vector of actions $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. The observations available to agent/node i are represented by the sufficient statistics $\phi_i(\boldsymbol{x}, \boldsymbol{y}_i)$. However, we also need the edge "observations" $\phi_{ij}(\boldsymbol{x}, \boldsymbol{y}_i, \boldsymbol{y}_j)$ to represent the information that can be communicated between neighboring agents i and j. Thus all we need for a CRF-RL model is a family of RL algorithms that directly optimizes stochastic policies.

3.1. Policy-Gradient Algorithms

Policy-gradient (PG) algorithms optimize polices by performing gradient ascent on a parameterized policy (Williams, 1992; Sutton et al., 2000; Baxter & Bartlett, 2001). These algorithms require only a parameterized and differentiable policy model $p(y|\boldsymbol{x};\boldsymbol{\theta})$, and a way to compute the gradient of the long-term reward $R(\boldsymbol{\theta})$.

A number of algorithms (Williams, 1992; Baxter & Bartlett, 2001; Peters et al., 2005) compute a Monte Carlo approximation of the reward gradient: the agent interacts with the environment, producing an observation, action, reward sequence $\{\boldsymbol{x}_1, y_1, r_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T, y_T, r_T\}$.[2] For example, under mild technical assumptions, including ergodicity and bounding all the terms involved, Baxter & Bartlett (2001) obtain

$$\widehat{\frac{\partial R}{\partial \boldsymbol{\theta}}} = \frac{1}{T}\sum_{t=0}^{T-1} \frac{\partial}{\partial \boldsymbol{\theta}} \ln p(y_t|\boldsymbol{x}_t;\boldsymbol{\theta}) \sum_{\tau=t+1}^{T} \beta^{\tau-t-1} r_\tau \,, \quad (11)$$

where an eligibility discount $\beta \in [0, 1)$ implicitly assumes that rewards are exponentially more likely to be due to recent actions. Without it, rewards would be assigned over a potentially infinite horizon, resulting in gradient estimates with infinite variance. As β decreases, so does the variance, but the bias of the gradient estimate increases (Baxter & Bartlett, 2001). In practice, (11) and all other policy-gradient algorithms share the same core estimator that make use of an *eligibility trace*

$$\boldsymbol{e}_t = \beta \boldsymbol{e}_{t-1} + \left.\frac{\partial}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \ln p\left(y_t|\boldsymbol{x}_t;\boldsymbol{\theta}\right) \quad (12)$$

Now $\boldsymbol{\delta}_t = r_t \boldsymbol{e}_t$ is the gradient of $R(\boldsymbol{\theta})$ arising from assigning the instantaneous reward to *all* log probability gradients, where $\beta \in [0, 1)$ gives exponentially more credit to recent actions. Additionally, β may be 1.0 for finite-horizon problems (Williams, 1992). The different policy-gradient algorithms vary in how they use instant gradient estimates $\boldsymbol{\delta}_t$.

For the experiments in this paper we adopt an online variation of the *natural actor-critic* (NAC) algorithm (Peters et al., 2005; Richter et al., 2007). While the NAC algorithm uses the estimator (12), it improves performance over (11) by: 1) using a critic that approximates a projection of value function, with discount factor $\gamma \in [0, 1)$, to reduce variance of the gradient estimates; 2) using a clever choice of critic parametrization to naturalize gradients (Amari, 1998); and 3) using a least squares approach to solve for the naturalized gradients, making full use of simulated trajectories. Algorithm 1 is used in our experiments (Richter et al., 2007).

Algorithm 1 Online Natural Actor-Critic.

$t = 1,\ A_1^{-1} = I,\ \boldsymbol{\theta}_1 = [0],\ \boldsymbol{e}_1 = [0]$
α=step size, γ=critic discount, β=actor discount
Get observation $\boldsymbol{x}_1$
while not converged **do**
  Sample action $\tilde{\boldsymbol{y}}_t \sim p(\cdot|\boldsymbol{x}_t, \boldsymbol{\theta}_t)$
  $\boldsymbol{e}_t = \beta \boldsymbol{e}_{t-1} + [\left.\frac{\partial}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \ln p\left(\tilde{\boldsymbol{y}}_t|\boldsymbol{x}_t;\boldsymbol{\theta}\right)^\intercal, \boldsymbol{x}_t^\intercal]^\intercal$
  Do actions $\tilde{\boldsymbol{y}}_t$
  Get reward r_t
  $\boldsymbol{\delta}_t = r_t \boldsymbol{e}_t$
  Get observation $\boldsymbol{x}_{t+1}$
  $\boldsymbol{w}_t = [\left.\frac{\partial}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \ln p(\tilde{\boldsymbol{y}}_t|\boldsymbol{x}_t, \boldsymbol{\theta})^\intercal, \boldsymbol{x}_t^\intercal]^\intercal - \gamma[0^\intercal, \boldsymbol{x}_{t+1}^\intercal]^\intercal$
  $\epsilon_t = 1 - t^{-1}$
  $\boldsymbol{u}_t = (\epsilon_t^{-1} - 1) A_{t-1}^{-1} \boldsymbol{e}_t$
  $\boldsymbol{q}_t^\intercal = \epsilon_t^{-1} \boldsymbol{w}_t^\intercal A_{t-1}^{-1}$
  $A_t^{-1} = \epsilon_t^{-1} A_{t-1}^{-1} - \frac{\boldsymbol{u}_t \boldsymbol{q}_t^\intercal}{1 + \boldsymbol{q}_t^\intercal \boldsymbol{e}_t}$
  $[\boldsymbol{d}_t^\intercal, \boldsymbol{v}_t^\intercal]^\intercal = A_t^{-1} \boldsymbol{\delta}_t$
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \boldsymbol{d}_t$
  $t \leftarrow t + 1$
end while

3.2. Decentralized Multi-Agent RL

Decentralized (PO)MDPs assume a number of agents, each with a local observation of the state space. Here the action $\boldsymbol{y}$ becomes a vector giving the action for each agent. In the general case, optimal decision making in Decentralized MDPs is NEXP-hard in the number of agents (Bernstein et al., 2000), due to the combinatorial degree of communication required between the agents to coordinate actions. Many approximate approaches exist including no communication (Peshkin et al., 2000); explicit actions to communicate state in-

[2] We use r_t as shorthand for $r(s_t)$, making it clear that only the reward value is known, not the underlying state.

formation (Boutilier, 1999); local sharing of value functions (Schneider et al., 1999); and others with varying degrees of formalism. Under a common global reward, and some forms of local reward (Bagnell & Ng, 2006), agents that do not communicate can learn to cooperate implicitly to maximize the global reward (Boutilier, 1999). However, unless each agent has access to the full state description, they will generally not be able to act optimally. Our contribution is to introduce a mechanism for agents to efficiently — due to the graph structure of the CRF — communicate in order to converge to a joint policy. Our choice of policy-gradient algorithms is motivated by their ease of integration with CRFs, but they have the additional benefit of being guaranteed to converge (possibly to a poor local maximum) despite the use of function approximation and partial observability. Our model of multi-agent learning is similar to Guestrin et al. (2002), which uses an exact form of BP for factorising Q-values and choosing jointly optimal actions, and hence may still be intractable for high tree width graphs.

4. Conditional Random Fields for RL

Applying CRFs for distributed RL is relatively straightforward: simply assume that the policy, $P(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$, of the POMDP factorizes according to a CRF. The agents correspond to the label nodes, and the edges encode the spatial or temporal collaboration between agents. In order to apply PG methods one needs: a) the ability to draw an action from the policy model (step 5 of Algorithm 1, and Eqns. (5),(6)); and b) computation of the gradient of the log-probability of the sampled action (step 6,11 and Eqns. (7),(8)). Efficient implementations rely on approximate sampling algorithms like the tree MCMC sampler described in Section 2.4. One can also easily verify that exponential families in general, and CRFs in particular, satisfy the mild technical conditions required for PG methods to converge, as long as all features are bounded.

Interestingly, the CEF policy representation (1) implements exactly the *soft-max* stochastic policy with linear feature combinations commonly encountered in RL applications, *e.g.*, Richter et al. (2007). Only the edge features prevent the trivial factorization of the distribution into independent agents that was demonstrated by Peshkin et al. (2000).

From the perspective of multi-agent RL, CRFs make efficient decentralized RL possible. By using conditional independence assumptions, the search space of the policy-gradient methods factorizes, leading to faster learning. Also, even though a CRF requires only local connections between agents, global interactions are still incorporated by belief propagation.

From a graphical models point of view, our technique is different from the usual online or offline training methods in two important ways. The training data is no longer iid. The action at time-step t stochastically determines the input at time-step $t+1$. Furthermore, the evaluation metric is no longer a loss function but a reward function that depends on both the current state.

Superficially, our setup looks similar to dynamic Bayesian networks (DBNs). DBNs are directed graphical models (in contrast to CRFs which are undirected graphical models) used to represent models that evolve with time. Typically DBNs are used for a) filtering: monitor the hidden system state s over time by computing $p(s_t|\boldsymbol{x}_1 \dots \boldsymbol{x}_t)$; b) prediction: computing $p(s_{t+1}|\boldsymbol{x}_1 \dots \boldsymbol{x}_t)$; or c) smoothing: computing $p(s_{t-1}|\boldsymbol{x}_1 \dots \boldsymbol{x}_t)$. In an RL context DBNs have been used to estimate the state transition matrix $\boldsymbol{P}(\boldsymbol{y})$, as well as the distribution $p(\boldsymbol{x}|s)$, in order to resolve partial observability in a POMDP (Theocharous et al., 2004). In contrast, we use CRFs as a policy, rather than as a state transition model.

We have shown how to learn *reactive* policies that ignore the fact that the true MDP state is unknown. Fortunately, PG methods still converge in this case. To take partial observability into account we could encode observation history, or a belief state (if $\boldsymbol{P}(\boldsymbol{y})$ and $p(\boldsymbol{x}|s)$ are known), into the sufficient statistics.

5. Experimental Results

We performed experiments on one toy domain to demonstrate why a joint policy is important, and two benchmark decentralized RL domains.

5.1. Grid Alignment

We constructed an abstract traffic domain, where it was known that agents would have to coordinate their actions in order to perform well, even in the case of a common global reward. Traffic flows along the edges of an $n \times n$ grid, always traversing to the opposite edge of the grid without turning (see Figure 2). Each intersection grid lines is an agent that controls a gate. The actions of a gate allow traffic to flow vertically or horizontally at each time step. Traffic units arrive with Pr=0.5 per time step per boundary node (but only the top and left boundaries). Importantly, traffic cannot flow until *all* the gates on the traffic's path line up. When this happens, all waiting traffic for that line propagates through instantly and each unit of traffic contributes +1 to a global reward. One or more

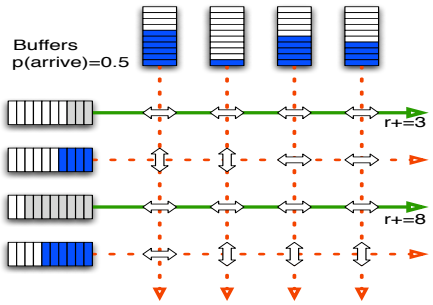


Figure 2: Abstract grid alignment domain.

Table 1: Learning algorithm parameters.

Domain	α_{ind}	α_{node}	α_{edge}	β	γ	runs
Grid	.002	.00001	.002	0.6	0.5	100
DSN	.0005	.00025	.0005	0.6	0.5	100
Traffic	.001	N/A	.01	0.9	0.95	50

misaligned gates blocks traffic, causing the length 10 buffer to fill up as traffic arrives. Full buffers drop traffic. Two observations per node indicate the normalised number of traffic units waiting for the node to align vertically, and horizontally. Edge features are 1 if the two nodes agree on an alignment, 0 otherwise. The optimal policy is for the *all* the n^2 gates to align in the orientation of the most waiting traffic, but since each node only knows how many traffic units are waiting for it, it must "negotiate" with neighbors on which way to align.

Learning Parameters: The CRF model is a 2-D grid, with nodes for each agent, and edges for all nodes connected by a grid line. Due to the 2-D nature of the CRF, MCMC estimation of the log partition function, and its gradient were required. MCMC estimation needed 10 tree samples. To initialize the tree sampler, we randomly picked a spanning tree of the whole graph and sampled from the tree. Empirically, this allows us to obtain a good estimation of the distributions much faster than starting with independent node randomization. Other parameters, including the number of independent learning runs, are summarized in Table 1. To prevent poor local maxima when $n > 5$ we needed step sizes for edge feature parameters α_{edge} to be larger than for node feature parameters α_{node}, boosting the effect of edge features on the policy.

Results: The optimal reward is the grid size n. Figure 3 shows the CRF RL approach compared to a naïve implementation with independent agents. The CRF approach obtains the optimal reward all the way to grid size 10 (100 nodes), at which point some runs fail to reach the optimal policy. The number of iterations

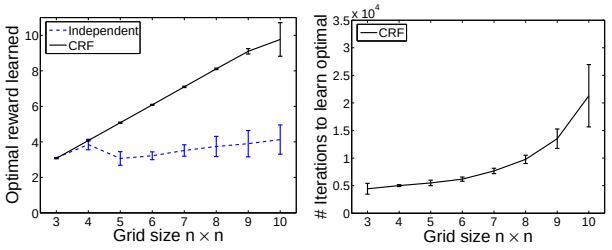


Figure 3: Average reward over the last 1000 steps, and iterations to optimality.

required to reach the optimal reward for the first time is shown on the right panel.

5.2. Sensor Networks

The distributed sensor network (DSN) problem is a sequential decision making variant of the distributed constraint optimization problem described in Dutech et al. (2005). The network consists of two parallel chains of an arbitrary, but equal, number of sensors. The area between the sensors is divided into cells. Each cell is surrounded by four sensors and can be occupied by a target. With equal probability targets can (from left to right) jump to the cell to its left, to its right, or remain where it is. Jumps that would cause a collision are not executed. The goal of the sensors to capture all targets. With initial configuration as in Figure 4, there are 37 distinct states. Each sensor can perform three actions resulting in a joint action space of $3^8 = 6561$ actions. The actions are: track a target in the cell to the left, cell to the right, or none. Every track action has a reward of -1. When in one time step at least three of the four surrounding sensors track a target, it is hit and its energy level is decreased by 1. Each target starts with an energy level of 3. When it reaches 0 the target is captured and removed. The three sensors involved in the capture are each provided with a reward of +10, and the goal is to maximize the total reward of all sensors. An epoch finishes when all targets are captured. If the DSN cannot capture all targets within 300 steps, the epoch is terminated, and a new epoch is started. A set of 50 randomly chosen initial states (with replacement) is cycled through for one episode. We run for 200 episodes and study the average reward of each episode. Finally, the whole process is independently repeated for 100 runs, and we report the average optimal reward and number of episodes.

Learning Parameters: We experimented with two alternative CRF models, a *cycle* in which neighboring sensors along the top and bottom are connected

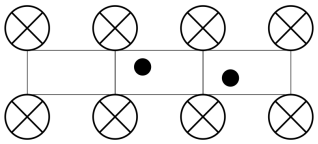


Figure 4: Sensor network domain with 8 sensors, 2 targets, and 3 cells.

as chains, and the cycle is completed with an edge between top and bottom sensors on the left and right ends. The *chain* is a more complex arrangement. Local sensors are bunched into groups of three (one top sensor, two bottom sensors and vice/versa). These meta-sensors form one CRF node. All the meta-sensors are connected in a 1-D chain, so (6) and (8) can efficiently estimate the log-partition function and its gradient.

Each node (or meta-node) has access to whether there is a target in its left and right cells (two binary values with two dummy cells at the two ends always observing no target). For the chain topology, the single edge feature is whether there are at least three out of four sensors focusing on their common cell. For the cycle topology, a single edge feature encodes whether connected sensors are focused on the same cell.

Results: The problem in Figure 4 has an optimal long-term average reward of 42. The best results from Dutech et al. (2005) use a distributed Q-learning approach where neighbors' Q-values are averaged (which implicitly assumes communication), achieving an average reward of less than 30. Figure 5 shows that CRF modeling with NAC achieves the optimal reward for this problem, and problems where the number of targets is increased up to 10. The ease with which we outperform the distributed Q-learning approach is not surprising, since the CRF allows sensors to agree on which target to focus on. We obtain similarly good results when the number of targets is fixed and more cells are added (Figure 6). The curious peak in the required iterations for 5 cells corresponds to the difficult situation where there are not enough sensors, *and* targets are able to jump around with few collisions. Adding more cells also adds more sensors, so that an individual sensor is rarely required to focus left and right at the same time. In both experiments the chain CRF does marginally better, probably due to the tighter coupling of the sensors and exact evaluation of the log-partition function.

5.3. Traffic Light Control

Many drivers have been frustrated by driving along a main street, to be constantly interrupted by red lights. This domain demonstrates learning an offset between neighboring intersections. The domain and simulator

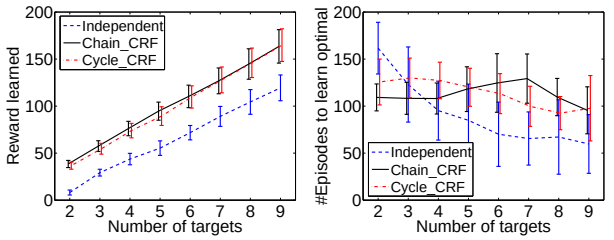


Figure 5: Results over 100 runs of the Sensor Network scenario, varying the number of targets.

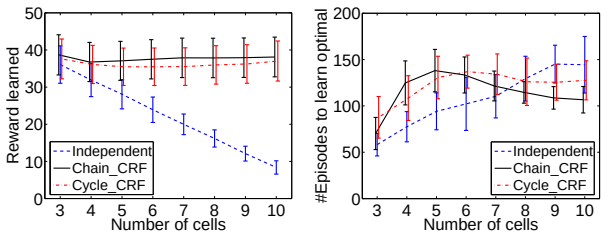


Figure 6: Results over 100 runs of the Sensor Network scenario, varying the number of cells.

code are from Richter et al. (2007), which contains the full implementation details. We model one main road with n controlled intersections and $n+1$ road links to traverse. It takes cars 2 time units to travel down a road to the next intersection. There are 4 actions, corresponding to the 4 traffic signal patterns that allow traffic to move through the intersection safely in any direction. For this experiment only the action that lets traffic drive straight ahead along the main road is useful. At each time step (about 5 seconds of real-time) the controller decides on the action for the next step. We do not restrict the order of actions, but importantly, we enforce the constraint that all actions must be activated at least once within 8 time steps so that vehicles on side streets would not wait forever. One car enters the leftmost end of the road every 4 time steps. We use realistic local rewards for each intersection: each intersection has an inductive loop sensor that can sense a car waiting, producing a -1 reward.

Learning Parameters: Each intersection is a CRF node that chooses from one of the four actions. For independent learners the only feature is a constant bias bit that allows learning of which phase is the most commonly used. No node features were given to the CRF. The edge features for the CRF version are an $a \times a$ binary matrix, where a is the number of actions. Bit i, j is on if the action of the left neighbor i from 2 time steps ago (the time required to drive down the road edge) matches the chosen action of the current intersection j. Edge parameters receive j's reward.

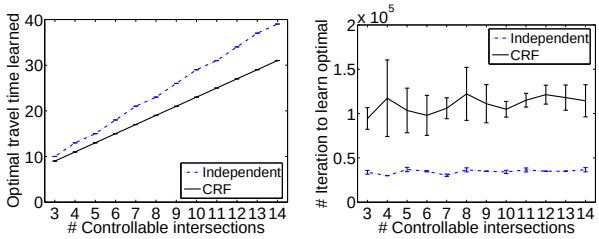


Figure 7: Results over 50 runs of the road traffic offset scenario. The X-axis is intersections. The left Y-axis is the travel time.

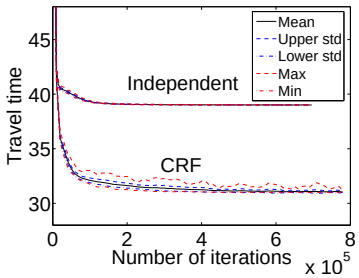


Figure 8: Convergence of NAC on the independent agents (upper curve) and on CRF model (lower curve). The number of controllable intersections is 14.

The edge feature matrix has exactly 1 bit set to true in the matrix in any step. Typically a road traffic network would be represented as an undirected graph. But this simple scenario is one long road, thus can be represented as a chain CRF.

Results: Figure 7 shows the results on the road traffic domain in the same style as previous results. Again, the CRF model clearly outperforms the independent agents approach. We observed, however, that early in learning both the independent agents and CRF learn that all traffic moves left to right. Giving the maximum possible time to this direction is a strong local minimum. After that, the independent agents fail to improve but the CRF model asymptotically approaches the optimal 0 waiting time.

Figure 8 shows convergence plots for the CRF approach versus the independent agents.

6. Conclusions

We have shown how to use CRFs to model control processes, or equivalently, how decentralized RL can be performed with CRF optimisation methods. Although all our examples have been related to controlling a process, a special case is where rewards are given simply for predicting the next input, i.e., time series prediction. From a reinforcement learning point of view we have presented an efficient policy-gradient solution to the difficult problem of optimizing joint actions in a decentralised (PO)MDP. Future work could explore RL in general graphical models, and how local rewards may be propagated through a graphical model.

Acknowledgements

NICTA is funded by the Australian Government's Backing Australia's Ability program and the Centre of Excellence program.

References

Altun, Y., Smola, A., & Hofmann, T. (2004). Exponential families for conditional random fields. In *UAI*, 2–9. Arlington, Virginia: AUAI Press.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*(2), 251–276.

Bagnell, J., & Ng, A. (2006). On local rewards and scaling distributed reinforcement learning. In *NIPS*.

Baxter, J., & Bartlett, P. (2001). Infinite-horizon policy-gradient estimation. *JAIR*, *15*, 319–350.

Bernstein, D., Givan, R., Immerman, N., & Zilberstein, S. (2000). The complexity of decentralized control of Markov decision processes. In *UAI*.

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *IJCAI*, 478–485.

Casella, G., & Robert, C. (1996). Rao-Blackwellisation of sampling schemes. *Biometrika*, *83*(1), 81–94.

Dutech, A., et al. (2005). Proc of reinforcement learning benchmarks and bake-offs II. In *NIPS 2005 Workshops*.

Guestrin, C., Lagoudakis, M., & Parr, R. (2002). Coordinated reinforcement learning. In *ICML*.

Hamze, F., & de Freitas, N. (2004). From fields to trees. In *UAI*.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic modeling for segmenting and labeling sequence data. In *ICML*.

MacKay, D. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge Univ. Press.

Peshkin, L., Kim, K.-E., Meuleau, N., & Kaelbling, L. (2000). Learning to cooperate via policy search. In *UAI*.

Peters, J., Vijayakumar, S., & Schaal, S. (2005). Natural actor-critic. In *ECML*.

Richter, S., Aberdeen, D., & Yu, J. (2007). Natural actor-critic for road traffic optimization. In *NIPS*.

Schneider, J., Wong, W.-K., Moore, A., & Riedmiller, M. (1999). Distributed value functions. In *ICML*.

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*.

Theocharous, G., Murphy, K., & Kaelbling, L. (2004). Representing hierarchical POMDPs as DBNs for multi-scale robot localization. In *ICRA*.

Williams, R. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.