Gaussian Processes for Sample Efficient Reinforcement Learning with RMAX-like Exploration

Tobias Jung and Peter Stone

Department of Computer Science

University of Texas at Austin

{tjung,pstone}@cs.utexas.edu

Outline:

1. Motivation & framework
2. Technical implementation
3. Experiments

Part I:
Motivation & Overview

This is what we want to do (and why)

Objective: dynamic programming

Consider: Time-discrete decision process $t = 0, 1, 2, \ldots$ with

- $\mathcal{X} \subset \mathbb{R}^D$ state space (continuous), $\mathcal{A}$ action space (finite)
- Transition function $x_{t+1} = f(x_t, a_t)$ (deterministic)
- Reward function $r(x_t, a_t)$ (immediate payoff)

Goal: For any x_0 find actions $a_0, a_1, \ldots$ such that $\sum_{t \geq 0} \gamma^t r(x_t, a_t)$ is maximized.

Dynamic programming: (value iteration)

- If transitions f and reward r **are known**, we can solve

$$Q = \mathcal{T}Q, \qquad \text{where } (\mathcal{T}Q)(x,a) := r(x,a) + \gamma \max_{a'} Q(f(x,a), a') \quad \forall x, a$$

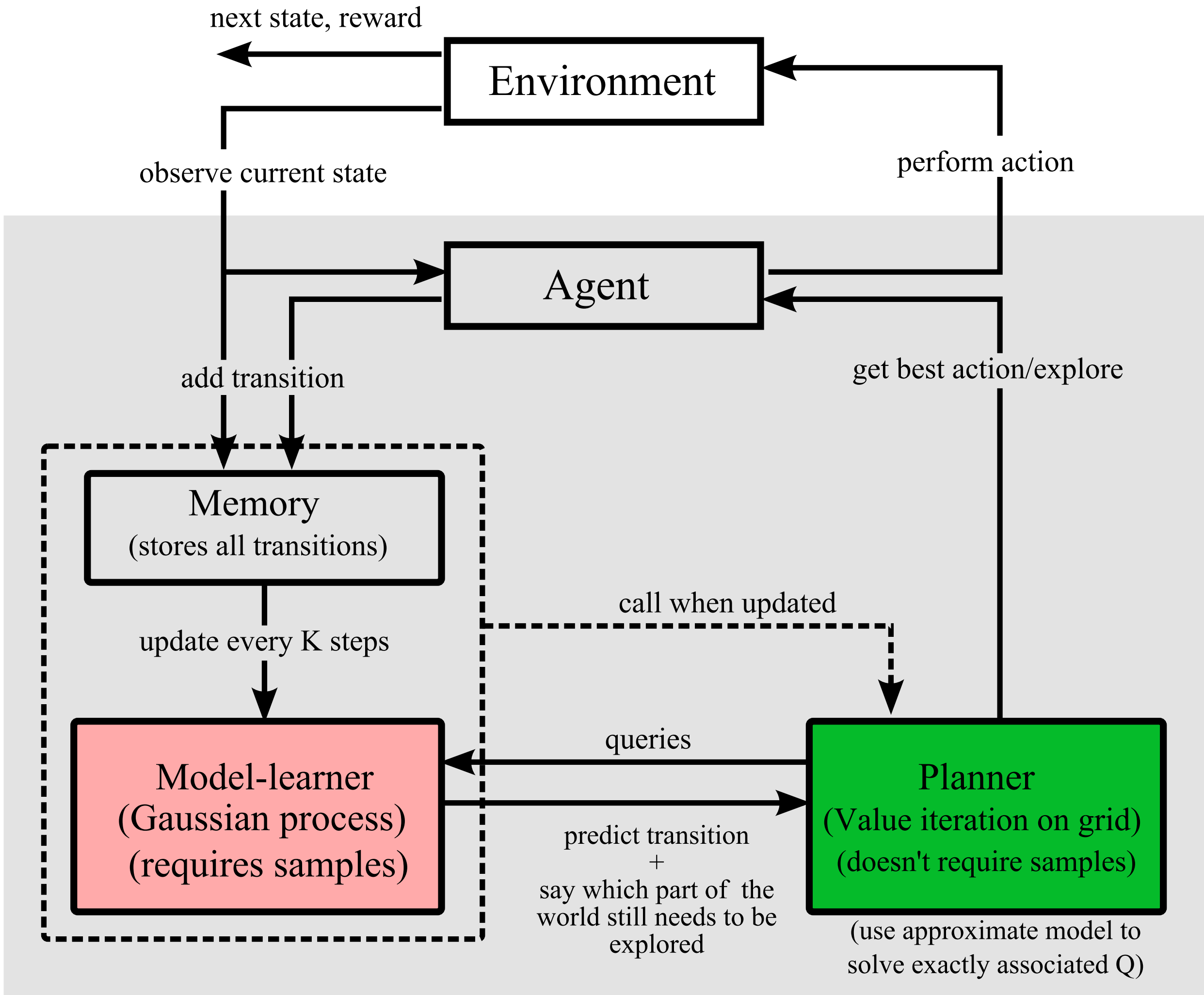
  to obtain Q^*, the optimal value function.
- Once Q^* is calculated, best action in x_t is simply $\operatorname{argmax}_a Q^*(x_t, a)$.

Problems:

- Usually f and r are not known a priori $\Longrightarrow$ **learned from samples.**
- (State-action space "too big" to do VI, $\Leftrightarrow$ will largely ignore this)

$\Longrightarrow$ **Our goal: want to improve sample efficiency.**

Model-based reinforcement learning



Remark: throughout the paper we will assume that the reward function is specified a priori.

$\Longrightarrow$ **Sample efficiency of RL wholly depends on sample efficiency of model learner.**

Overview of the talk

Benefits of model-based RL:

- More sample efficient than model-free (however, also more computationally expensive) :
  - Samples only used to learn model, but not as "test-points" in value iteration.
  - **Sample efficiency of RL wholly depends on sample efficiency of model learner.**
- (Model can be reused to solve different tasks in same environment.)

Model-based RL: requires us to worry about 3 things

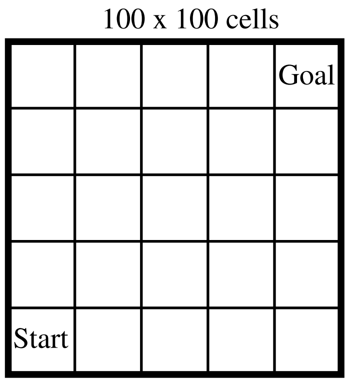
1. How to implement planner? Here: simple interpolation on grid. (not part of this paper)
2. How to implement model-learner?
3. How to implement exploration?

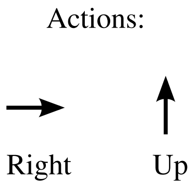
Our contribution GP-RMAX: model-learner=Gaussian process regression

- Fully Bayesian: provides natural (un)certainty for each prediction.
- Automated, data-driven hyperparameter selection.
- Framework for feature selection: find & eliminate irrelevant variables/directions:
  - improves generalization & prediction performance $\Longrightarrow$ faster model learning.
  - improves uncertainty estimates $\Longrightarrow$ more efficient exploration.
- Experiments indicate highly sample-efficient online RL possible.

Motivation: GP+ARD Can Reduce Need for Exploration

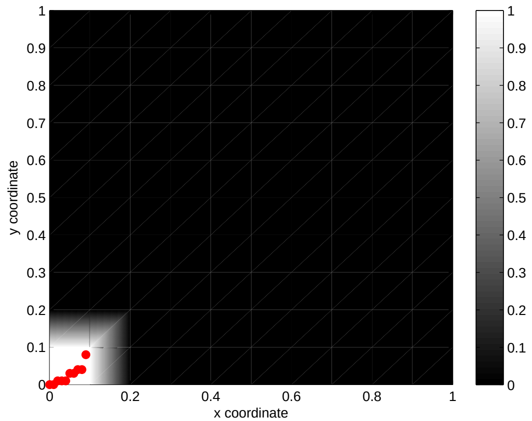
Example: compare three approaches for model learning in a 100×100 gridworld.



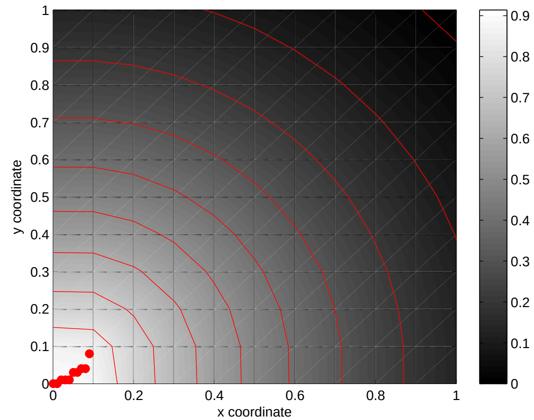


$$
\begin{aligned}
x_{new}^{right} &= x_{old} + 0.01 \\
y_{new}^{right} &= y_{old} \\
x_{new}^{up} &= x_{old} \\
y_{new}^{up} &= y_{old} + 0.01
\end{aligned}
$$

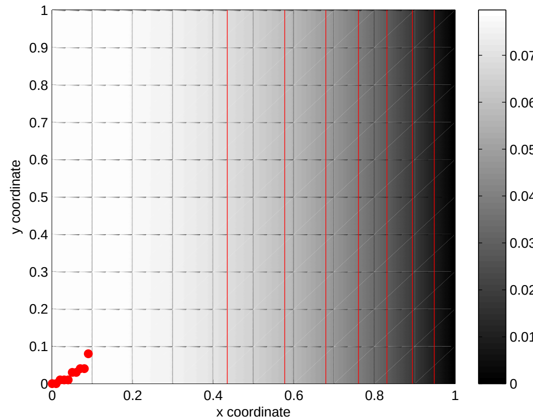
After observing 20 transitions, we plot how **certain** each model is about its predictions for "right":



10×10 grid



Hand-tuned uniform RBF



GP with ARD kernel

GP+ARD detects that the y-coordinate is irrelevant $\Longrightarrow$ reduced exploration $\Longrightarrow$ faster learning.

Part II:
Technical implementation

This is how we do it

a. Model learning with GPs

Model learning with GPs

General idea:

- Have to learn D-dim transition function $\mathbf{x}' = f(\mathbf{x}, a)$.
- To do this, we combine multiple univariate GPs.

Training:

- Data consists of transitions $\{(\mathbf{x}_t, a_t, \mathbf{x}'_t)\}_{t=1}^{N}$, where $\mathbf{x}'_t = f(\mathbf{x}_t, a_t)$ and $\mathbf{x}_t, \mathbf{x}'_t \in \mathbb{R}^D$.
- Train independently one GP for each state variable, action.
  - $\mathcal{GP}_{ij}$ models i-th state variable under action $a = j$
  - $\mathcal{GP}_{ij}$ has hyperparameters $\vec{\theta}_{ij}$ found from minimizing marginal likelihood

$$\min_{\vec{\theta}_{ij}} \mathcal{L}(\vec{\theta}_{ij}) = -\frac{1}{2} \log\det(\mathbf{K}_{\vec{\theta}_{ij}} + \sigma\mathbf{I}) - \frac{1}{2}\mathbf{y}^{\mathsf{T}}(\mathbf{K}_{\vec{\theta}_{ij}} + \sigma\mathbf{I})^{-1}\mathbf{y} - \frac{n}{2}\log 2\pi$$

  - Once trained, $\mathcal{GP}_{ij}$ produces for any state $\mathbf{x}^*$
    - Prediction $\tilde{f}_i(\mathbf{x}^*, a = j) := \mathbf{k}_{\vec{\theta}_{ij}}(\mathbf{x}^*)^{\mathsf{T}}(\mathbf{K}_{\vec{\theta}_{ij}} + \sigma\mathbf{I})^{-1}\mathbf{y}$.
    - Uncertainty $\tilde{c}_i(\mathbf{x}^*, a = j) := k_{\vec{\theta}_{ij}}(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_{\vec{\theta}_{ij}}(\mathbf{x}^*)^{\mathsf{T}}(\mathbf{K}_{\vec{\theta}_{ij}} + \sigma\mathbf{I})^{-1}\mathbf{k}_{\vec{\theta}_{ij}}(\mathbf{x}^*)$.
- At the end, predictions of individual state variables are stacked together.

Automatic relevance determination

Automated procedure for hyperparameter selection:

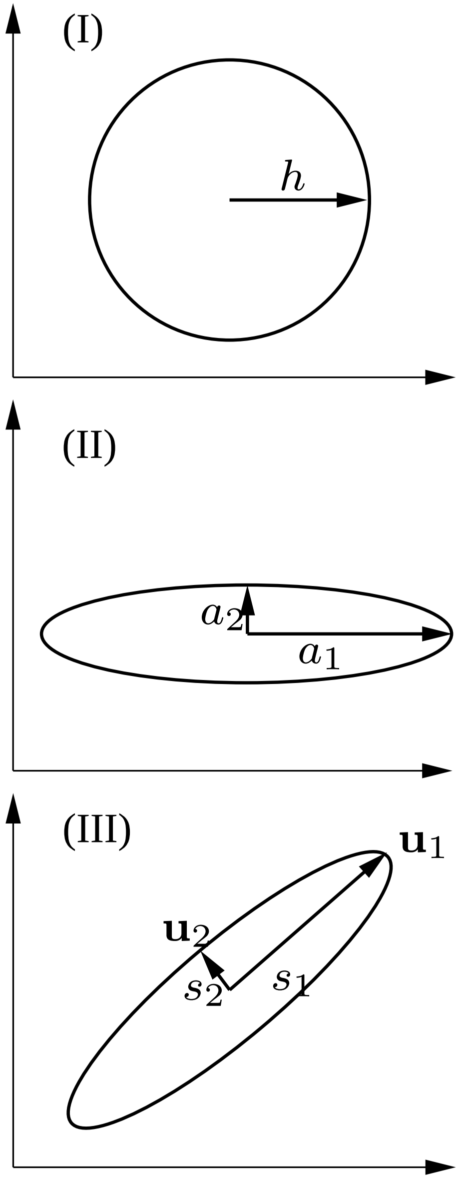
- $\Longrightarrow$ can use cov with larger number of hyperparameters
  (infeasible to set by hand)
- $\Longrightarrow$ better fit regularities of data, remove what is irrelevant

Covariance: We consider three variants of the form:

$$k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}') = v_0 \exp\left\{-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Omega} (\mathbf{x} - \mathbf{x}')\right\} + b$$

with scalar hyperparameters v_0, b and matrix $\boldsymbol{\Omega}$ given by

- **Variant I:** $\boldsymbol{\Omega} = h\mathbf{I}$.
- **Variant II:** $\boldsymbol{\Omega} = \mathrm{diag}(a_1, \ldots, a_D)$.
- **Variant III:** $\boldsymbol{\Omega} = \mathbf{M}_k\mathbf{M}_k^\top + \mathrm{diag}(a_1, \ldots, a_D)$.



Note:

- (II), (III) contain adjustable parameters for every state variable
- Setting them automatically from data $\Longrightarrow$
  Model selection automatically determines their relevance
- Can use likelihood scores to prune irrelevant state variables.

b. Planning (with approximate model)

Value iteration in $\mathbb{R}^D$

Remember:

- Input to the planner is the current model.
- The current model "produces" for any (x, a)
  - $\tilde{f}(x, a)$, the predicted successor state
  - $\tilde{c}(x, a)$, the associated uncertainty (0=certain, 1=uncertain)

General idea:

- Value iteration on grid Γ_h + multidimensional interpolation.
- Instead of true transition function, simulate transitions with current model.
- As in RMAX integrate **"exploration"** into value updates. (Nouri & Littman 2009)

Algorithm: iterate $k = 1, 2, \ldots$: $\forall$ node $\xi_i \in \Gamma_h$, action a

$$Q_{k+1}(\xi_i, a) = (1 - \tilde{c}(\xi_i, a)) \cdot \Big[\underbrace{r(\xi_i, a)}_{\text{given a priori}} + \gamma \max_{a'} \underbrace{Q_k\big(\tilde{f}(\xi_i, a), a'\big)}_{\text{interpolation in } \mathbb{R}^D} \Big] + \tilde{c}(\xi_i, a) \cdot V_{\mathsf{MAX}}$$

Note:

- If $\tilde{c}(\xi_i, a) \approx 0$, no exploration.
- If $\tilde{c}(\xi_i, a) \approx 1$, state is artificially made more attractive $\Longrightarrow$ exploration.

Part III: Experiments

These are the results

Experimental setup

Examine what: examine online learning **performance** of GP-RMAX, that is,

- sample complexity, and
- quality of learned behavior

in various popular benchmark domains.

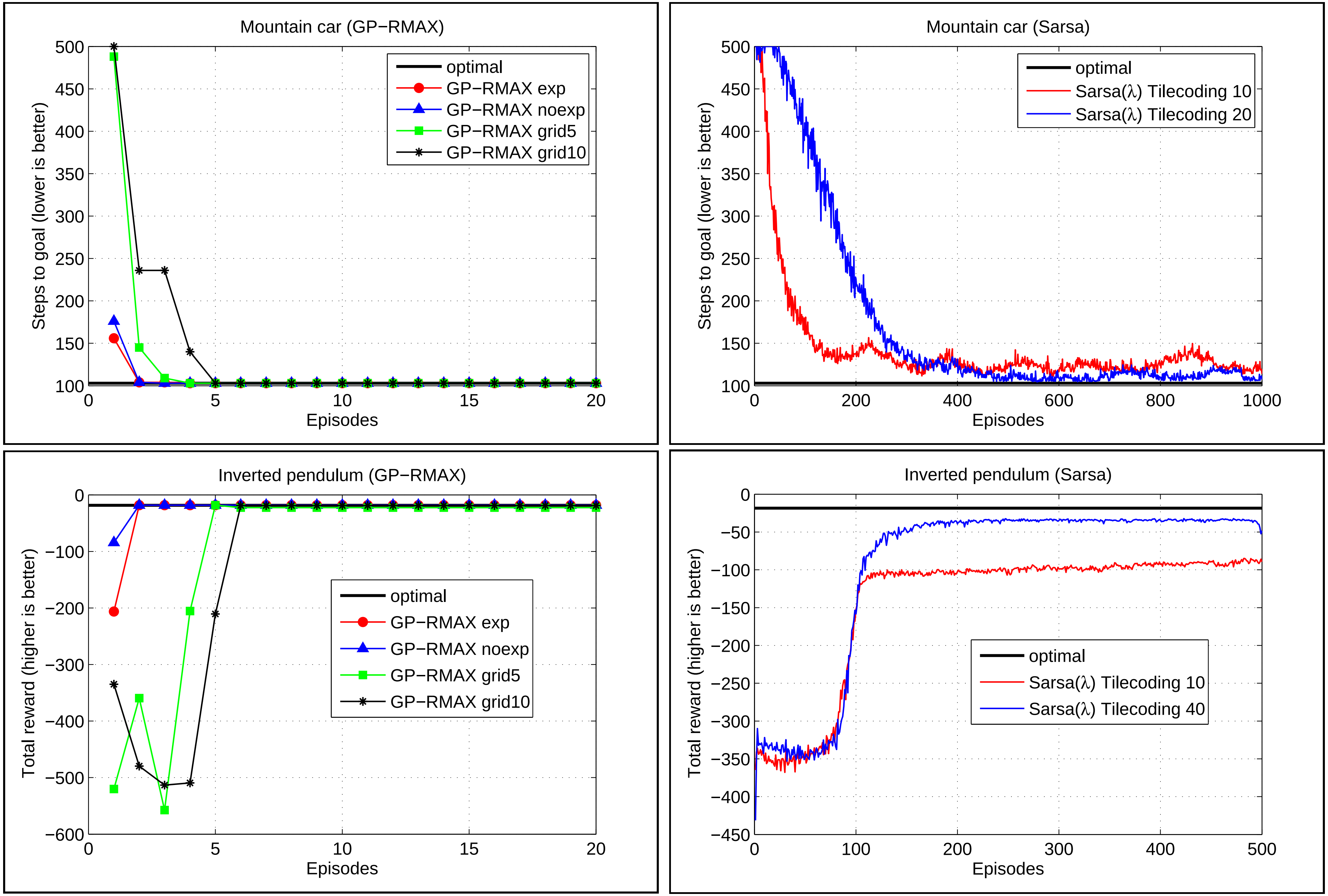
Domains:

- Mountain car (2D state space)
- Inverted pendulum (2D state space)
- Bicycle balancing (4D state space)
- Acrobot (swing-up) (4D state space)

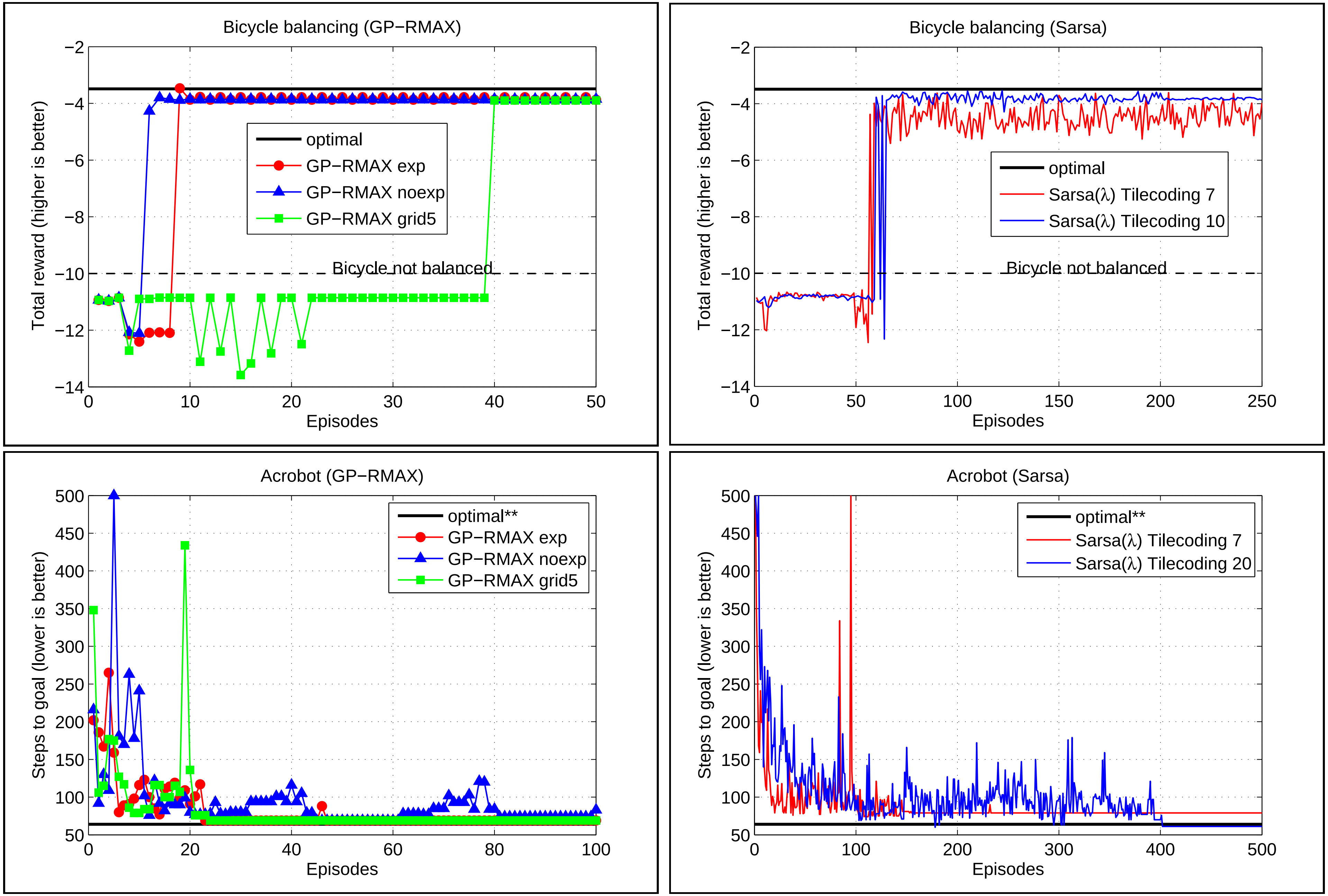
Contestants:

- Sarsa(λ) + tilecoding
- GP-RMAXexp (exploration where uncertainty is determinded from GP)
- GP-RMAXnoexp (no exploration)
- GP-RMAXgrid (exploration where uncertainty is determined from grid)

Results 2D domains



Results 4D domains



Finish

GP-RMAX:

- Online model-based RL that separates
  - function approximation in the model-learner **(which requires samples)**
  - from interpolation in planner **(which does not require samples).**
- Employs GPs with data-driven, automatic hyperparameter selection (feature selection):
  - improves generalization & prediction performance $\Longrightarrow$ **faster model learning**
  - improves uncertainty estimates $\Longrightarrow$ **more efficient exploration.**
- $\Longrightarrow$ Large gains over model-free RL possible (if model learning is "easier" than VF learning).

Limitations & future work:

- **Major problem:** planner relies on global value iteration
  - A naive grid is limited to low dimensionality.
  - More fancy grids (sparse, adaptive) might scale to higher dimensionality, but this is largely open research.
- **Minor problems:** doing away with our simplifying assumptions
  - deterministic state transitions (experiments done with well-behaved simulations)
  - known reward function
  - discrete (finite) actions

Related work

Closely related:

[1] A. Nouri and M. L. Littman. Dimension reduction and its application to model-based exploration in continuous spaces. ECML, 2010

[2] S. Davies. Multidimensional triangulation and interpolation for reinforcement learning. NIPS, 1996.

[3] T. Hester, M. Quinlan, and P. Stone. Generalized Model Learning for Reinforcement Learning on a Humanoid Robot. ICRA, 2010.

[4] N. K. Jong and P. Stone. Model-based exploration in continuous state spaces. In: 7th Symposium on Abstraction, Reformulation and Approximation, 2007.

Related:

[5] A. Bernstein and N. Shimkin. Adaptive-resolution reinformcement learning with efficient exploration. Machine Learning (published online 5 May 2010).

[6] R. Brafman and M. Tennenholtz. R-MAX, a general polynomial time algorithm for near-optimal reinforcement learning. JMLR, 3:213-231, 2002.

[7] M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. Neurocomputing, 72(7-9):1508-1524, 2009.

[8] L. Li, M. L. Littman, and C. R. Mansley. Online exploration in least-squares policy iteration. AAMAS, 2009

[9] A. Nouri and M. L. Littman. Multi-resolution exploration in continuous spaces. NIPS, 2008