Graph Based RL

- Reinforcement Learning for Continuous Tasks
  - General Problems…
  - Some Solutions

- Similar Existing Methods
  - Kernel Based Reinforcement Learning [Ormoneit00, Jong06]
  - Robust Combination of Local Controllers [Guestrin01]

- Graph Based RL
  - Sampling Strategy for the nodes
  - Optimal Exploration Strategies…
  - Benchmarks…
Most control problems are continuous in the state space, action space and time.

Nearly all RL methods need to discretize time
- Exception: Continuous Time RL [Doya00].
  - Mathematical treatment in continuous time
  - Implementation: We need to discretize time again

Many of them need to discretize the action space
- Nearly all Q-Learning Methods
- Exception: Actor Critic / Policy Gradient Methods
- For high quality movement trajectories, discretization can affect the quality of the policy considerably
RL for Continuous State Spaces

- Function Approximation
  - Parametric Function Approximation
    - Converges only in special cases
    - Convergence not to the true optimal Q-Function
    - Not consistent: More training data does not need to improve the policy
    - Local FA’s: Curse of dimensionality!!
  - Online Learning Methods (like Q-Learning)
    - May suffer from catastrophic forgetting
    - Learning can be very slow
      - Small Learning rates have to be used
      - Training data is not used efficiently
Kernel Based RL

- Non-Parametric function approximation method
- Instance Based:
  - For each action:
    - Set of states and successor states
      - Samples: \((x_s, a, y_s^a, r)\), where \(r = r(y_s^a, x_s, a)\)
  - Batch learning method
  - Discrete Set of actions is used
- Global optimality property
  - Additional training data always improves the quality of the Q-Function estimates
- Approximates transition probabilities and rewards instead of the value function
Kernel Based RL: Assumptions

- Generative Model: Simulator of the process
  - Black-Box:
    - Takes state x and action a
    - Output: Outcomming state y sampled according to dynamics of the MDP and reward r

- Training data:
  - For each action:  \( S^a = \{(x_s, y_s^a) | s = 1, \ldots, m\} \)
  - \( x_s \) has to be uniformly sampled in the state space
  - Quite restrictive and not realistic, only for offline-learning…
  - Needed for convergence proofs…
Kernel Based RL

- **Discrete Case:**
  - Reward and Value Function are piecewise constant on a finite number of partitions $B_1 \ldots B_N$
  - With sufficient training data we can estimate transition probabilities and rewards of the partitions (see Prioritized Sweeping).

- **Continuous Case:**
  - Smoothing with fuzzy partitions
  - Membership function:
    - Weighting kernel $k_{S^a,b}(x, x')$
    - \[ \phi^+ \left( \frac{\|x - x\|}{b} \right) / \sum_{(x_u, y_u^a) \in S^a} \phi^+ \left( \frac{\|x_u - x\|}{b} \right) \]
    - Bandwidth Parameter: $b$
Kernel Based RL

- **Notation:**
  - Iteration Operator
    - $Q_{t,a}^* = \Gamma_a J_{t+1}^*(x) = E[r(X_{t+1}, x, a) + \alpha J_{t+1}^*(X_{t+1}) | X_t = x, a_t = a].$
  - Maximum operator over all actions:
    - $J_t = \mathcal{T} Q_t$
  - Approximation of the Q-Function
    - $\hat{J}_a J(x) = \sum_{(x_s, y_s^a) \in S^a} k_{S^a, b}(x_s, x) [r \left( y_s^a, x_s, a \right) + \alpha J \left( y_s^a \right)].$
    - It is sufficient to find a set of function values at the locations $y_s^a$ which satisfies the condition $\hat{J}(y_s^a) = (\mathcal{T} \hat{\Gamma}_a J)(y_s^a)$
    - Q-Values of new states $x$ can then be derived by $\hat{J}_a J(x)$
Kernel Based RL

- Theorem 1: The approximate value iteration converges to a unique fixed point.
- Theorem 2: If we use an admissible shrinkage rate for the kernel bandwidth, the approximate value function converges to the optimal value function as the number of samples in each data set $S^a$ goes to infinity.
- First theoretical result which can guarantee convergence to optimal policy.
Kernel-Based Prioritized Sweeping [Jong06]

- Online version of Kernel Based RL
  - Sampling assumptions are hurt -> no convergence guarantee any more
- Prioritized Sweeping (discrete case)
  - Uses maximum Likelihood estimates of the transition probabilities and the reward.
    - If $n^{s,a} > n_{\text{known}}$
  - Also includes directed exploration policy for unknown transitions
    - Define absorbing state with value $V_{\text{max}}$ for all transitions with $n^{s,a} \leq n_{\text{known}}$
    - Optimistic Values are propagated through state space
      - => Directed exploration…
Online Kernel-Based RL

- Quantities needed for prioritized sweeping are estimated via kernels:
  \[ T(s, a, s') = \frac{1}{Z_{s,a}} \phi \left( \frac{d(s, s_i)}{b} \right), \text{ if } a_i = a \]
  \[ R(s, a) = \frac{1}{Z_{s,a}} \sum_{i | a_i = a} \phi \left( \frac{d(s, s_i)}{b} \right) r_i \]

- If \( Z_{s,a} \) is below a certain threshold add exploration transition to terminal node.
- First use of model-based directed exploration for continuous state spaces
Online KBRL

- Absolute Transitions:
  - Also assigns high probability to transitions in opposite direction.
  - Depends highly on the kernel width
    - Small kernel width needed
    - Needs many Samples…
- Better: Relative Transition
  - Computational much more expensive, 2nd Kernel needed…
Online KBRL

- **Pros:**
  - Model-Based directed exploration
  - Sampling Based: Better convergence properties than parametric methods

- **Cons:**
  - Computationally quite expensive
    - Needs a lot of training samples
    - No mechanism to adapt the density of states in relevant areas
  - Still uses discrete time and actions
Graph-Based RL

- Robust Combination of Local Controllers [Guestrin01]
  - Local Controllers have to be known
    - Simple controllers (e.g. connect 2 points with a straight line)
  - Randomized algorithm:
    - Nonparametric combination of local controllers
    - Generalizes probabilistic roadmaps: [Hsu et al.99]
      - Stochastic domains
        - Connect random samples in the state space with local controllers
        - Uses continuous actions and timesteps
    - Discounted MDPs
  - Stochastic motion planning:
    - Given some start and goal configurations, find a high probability of success path.
    - Quality of local controller:
      - Success Probability and Costs of local controller
Robust Combination of Local Controllers

- Sample Nodes uniformly in the state space
  - $X_1 \ldots X_{N-1}$, $X_0$...start state, $X_N$...goal state
- Simulate to estimate local connectivity
  - Estimate $p_{ij}$ for $k$ nearest neighbors of $i$
- Shortest path algorithm to find most probable path from $X_0$ to $X_N$
  - Connection Weights: $-\log p_{ij}$
  - Stochastic Domain: Exact position of obstacles is unknown
  - Path with highest probability:
Robust Combination of Local Controllers

- What about Costs?
  - MDPs find path with lowest expected cost:
    - Implicit trade-off: cost of hitting obstacles and reward for goal
    - In Robotics, a successful path often more important than a short path:
  - Make the trade-off explicit:
    - What is the lowest cost path with success probability of at least $p_{\text{min}}$?
Robust Combination of Local Controllers

- Lowest Cost Path with success probability of at least $p_{\text{min}}$
- Dynamic programming algorithm:
  - Discretize $[p_{\text{min}}, 1]$ into $S+1$ values;
    - $q(s) = (p_{\text{min}})s/S$, $s = 0, \ldots, S$
    - $V(s, x_i)$: minimum cost-to-go starting at $x_i$, reaching goal with success probability at least $q(s)$.

High Success Probability  Low Success Probability
Robust Combination of Local Controllers

- Similar notion of local controllers than our approach
  - Local controllers are assumed to be known as prior knowledge
  - Also assume knowledge of the environment
- Only used in an offline formulation
  - How do I explore the graph?
  - How do I sample the nodes efficiently?
  - What do I do if transition is not successful?
- Stochastic Environments, high probability paths
  - We want to concentrate on (quasi)-deterministic environments, and find optimal paths
Graph Based RL

- Our Approach:
  - Used for quasi-deterministic continuous control problem
  - Uses continuous state and action spaces and continuous time
  - Tackles 2 questions:
    - How do I explore the graph?
    - How do I sample the nodes efficiently?
  - Main Question: How do I explore the state space most efficiently to find good solutions from the start node $x_S$ to the goal node $x_G$?
Graph Based RL: Successor States

- Sampling the nodes
  - The graph is built in an online process
  - 2 different goals:
    - Extend the graph in unknown regions which are likely to lead to the goal
    - Increase the density of nodes regions which are likely to contain the optimal path
  - At each step, we sample potential successor states $s_k$
    - Applying random actions for a random amount of time
    - From the current node $x_t$
    - From random nodes $x_i$
  - Successor States are included as terminal node with an estimated value $\hat{V}$ in the graph
    - Enforces Directed Exploration
Graph Based RL: Successor States

- Estimating the Value of Successor States
  - Additional prior knowledge can be used:
    - Optimistically estimating the Value at successor state $x$
      - Costs of direct path to goal
    - If goal is not known:
      - Use constant, maximal possible value of the MDP.
  - Additionally use knowledge of neighbored nodes
    - Nodes far away / High uncertainty: use optimistic value
    - Many nodes in neighborhood: use weighted sum of values
Graph Based RL: Successor States

- Gaussian Process
  - Gives us a mean estimate and a variance estimate
    - Variance estimate ~ node density
  - Use optimistic Value as prior mean
  - Posterior: Use knowledge of neighbored nodes

- Squared Exponential Kernel: \( k(x, x') = \exp \left( -\frac{d(x, x')^2}{2\sigma^2} \right) \)
- Sigma... Kernel Width: Specified Sampling Resolution
Graph Based RL: Successor States

- **Gaussian Process:**
  - Use mean as estimated Value
  - Variance/Uncertainty of estimate tells us whether we can produce new information with successor node $s_k$
  - Only use successor states where the variance is higher than $\theta_{\text{min}}^{\text{exp}}$
    - $\theta_{\text{min}}^{\text{exp}}$ should be lowered over time

- **Successor state** is added to the graph if the agent visits the corresponding terminal node
  - We have to delete all remaining successor states in the neighborhood because their value and variance estimates are not valid any more…

- We calculate all local connections from the new node and add them to the graph

- Cost estimate: Time needed to accomplish connection
  - Without knowledge of any obstacles
  - Real costs are not known until we visit the edge
Creating/Exploring the Graph

- We have to explore:
  - Successor Nodes
    - Creating new nodes is very expensive in the sense of exploration
    - Each node adds several new edges we have to explore…
  - Unvisited edges
    - We can estimate costs coming from the time needed for the transition from the local controllers
    - Real costs are unknown.
Creating/Exploring the Graph

- Exploration edges:
  - Transitions to virtual final states
  - Final value $\hat{V}$:
    - Successors: Estimated Values
    - Unknown Edges: Value of the end state of the transition

- Which exploration edges should we explore?
  - Edges, that are likely to be in the neighborhood of the optimal path from $x_S$ to $x_G$
  - Exploration score:
    - Costs from start state + estimated costs of edge + final value $\hat{V}$
    - $\sigma(e) = c(x^S, x') + \hat{r}(e) + \hat{V}$
    - We use only known edges for $c$ and $V$, otherwise there is always a low cost path to an exploration edges (usually not all edges have been explored)
Creating/Exploring the Graph

- Exploration Score of Successor Nodes
Creating/Exploring the Graph

- Exploration policy $\pi^{\text{exp}}$
  - Use all edges (in order to explore unknown edges)
  - Calculate the maximum exploration score $\sigma_{\text{max}}$ of all end transitions (including the transition to the goal state)
  - Only activate the final transitions which have a higher exploration score than $\sigma_{\text{max}} - \theta_\sigma$

- Interpolation between:
  - Greedy Search: $\theta_\sigma = \infty$
    - Always explore towards the goal
    - Creates a more uniform distribution of nodes
    - Wastes time to go to goal which could be used for exploration
  - A*-Search: $\theta_\sigma = 0$
    - Always go to the best looking exploration target
    - May need to traverse the state space many times
Exploring the graph

- Different $\theta_\sigma$

- Plots for summed online rewards and time needed to get to a certain optimal performance
Building/Exploring the graph

- Building process of the graph

20 Episodes

100 Episodes
Value: -13.5

500 Episodes
Value: -9.5
Results:

- PuddleWorld: Comparison to prioritized sweeping/uniform sampling of the nodes
Results:

- Armreaching Task under static stability constraints of the CoM
  - Optimistic distance to goal is not known
  - We can use a heuristic that uses
Extensions and Open Questions

- How does it scale to high-dimensional problems?
  - Results are still missing
  - Should be possible for static problems…
- Dynamic Environments / More complex controllers
  - Much more complex local controllers
    - In particular for more dimensions
    - Agent has to reach the position and velocity for each dimension in the same time. Quite restrictive…
    - Still works for 2 dimensions… more dimensions is critical
- Additional Problem:
  - Valid neighborhood of each node (where does the controller still work?)
    - We have to do additional analysis here…
  - Can we somehow decouple the dimensions, relief the restriction of reaching the target-state exactly in the same time?
- Application: Robot Balancing task with Motion Primitives
Extensions and Open Questions

- Dynamic Environments:
  - Local Controllers

- Both dimensions are controlled to reach the end point in the same amount of time
Extensions and Open Questions

- **Uncertainty:**
  - Use robust controllers (feedback)
  - Use stochastic graphs

- **Integrating sensory data**
  - Sensory data can’t be directly controlled by a local controller
  - Possible Solution: Use simple, instanced based Function approximation in the nodes

- **Pruning of the graph:**
  - Many nodes which have been created are not needed any more later in the exploration/optimization process
    - Useful in particular for high dimensional problems to tackle the curse of dimensionality
  - Use more sophisticated node optimization techniques
    - Generate gradient information at each nodes by simulation

- **Non-linear, complex dynamical models**
  - Can the local controllers also be learnt?
  - In any case... a very hard task
Extensions and Open Questions

- Reward Prediction
  - How do we decrease the number of edges we have to explore?
  - Generalize the rewards of the edges
    - If we have a more accurate estimate of the reward of an edge, less exploration edges have to be visited
  - First approach: ambiguous results, needs more investigation...

![Diagram](image)
Conclusions

- Sample-Based Methods have better convergence properties
- Graph-Based RL:
  - Extension of the [Guestrin01] approach to online learning.
  - Builds the graph online in an efficient manner
- Graph-Based RL can produce high-quality policies which handle continuous actions and continuous time steps
- Outperforms parametric RL methods in learning speed and quality of the learnt policy considerably.
- There are still a few open questions/problems …
- Computationally quite expensive…