Batch-iFDD for Representation Expansion in Large MDPs

UAI 2013

Authors: Alborz Geramifard, Tom Walsh, Nicholas Roy, Jonathan P. How
Laboratory for Information and Decision Systems,
Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology

Presented by Junkyu Lee

Original slides: http://alborz-geramifard.com/Files/13UAI-BatchiFDD-slides.pdf

Motivation of this work

- How to describe Large state space in MDP?

- RL/Planning problems
  - MDP based formalism
    - State, Action, Reward
  - Flat state
  - Function approximation in RL using a set of features
    - How to construct such features?

Cont’

- In many cases, features are coming from…
  - Logical description of a problem (relational logic / propositional logic)
- Issues with features from symbolic logics
  - Requires large number of state variables (propositions)
  - Many clauses (function or combination of state variables) can be ignored
    - Usually specified by a global constraints
- Goal
  - Find a good abstraction/aggregation/coarser representation from the most fine grained features

Batch-iFDD – incremental feature dependency discovery

- Features (specific for Batch-iFDD)
  - This paper assumes that the original feature space has a description in propositional logic.
  - A feature is a conjunction of propositions
    - Note that full conjunction of propositions (and its negation) is equivalent to the original state space representation
  - Coarse feature ~= a conjunction with lower number of propositions

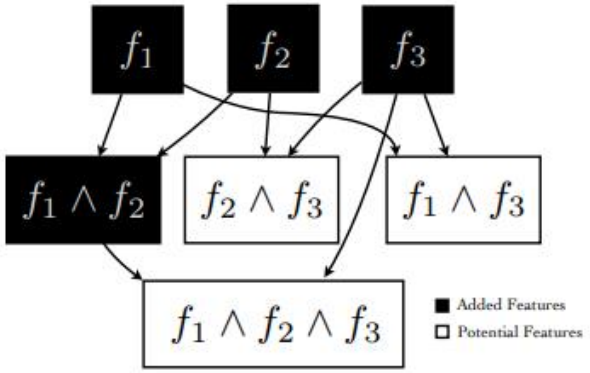
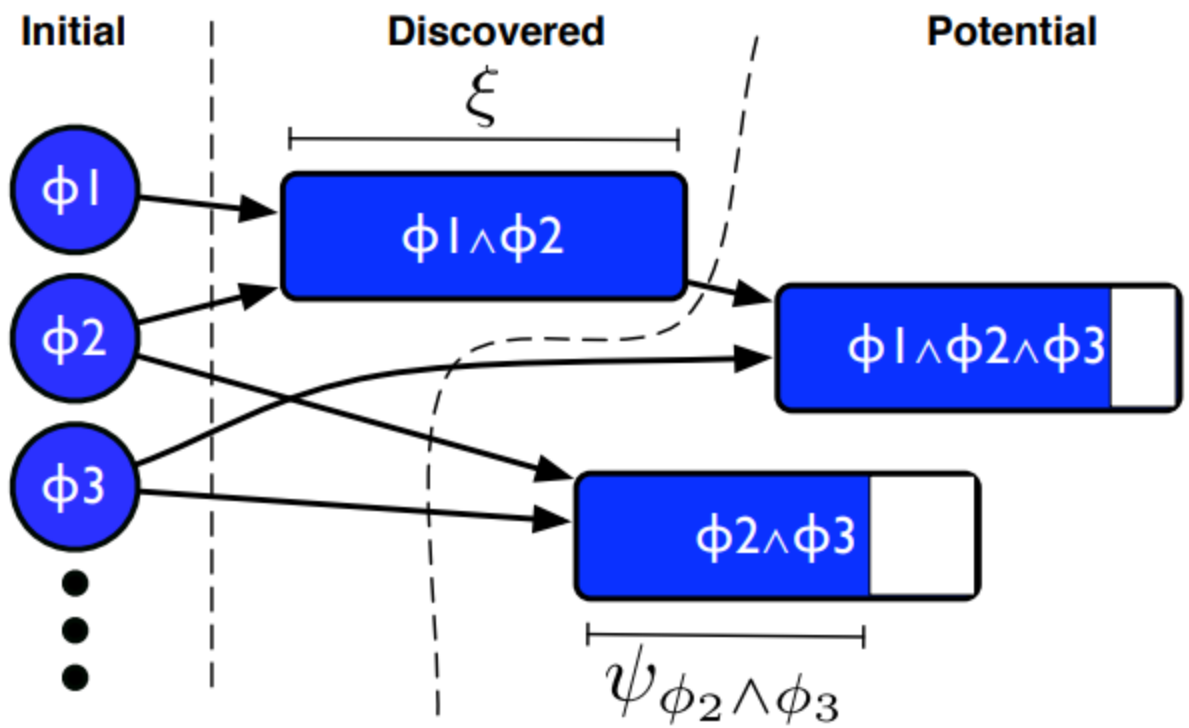


Figure 1: A partial concept lattice showing potential features as identified by iFDD. With methods like OMP-TD, all nodes are considered as potential features, despite their location in the lattice.

Overall framework

- Take a MDP/planning problem
- Enumerate all propositions
- Initialize feature pool with a conjunction up to 2 propositions
  - E.g. if there are propositions A,B, C, D [should include negations too] init pool is {empty, A, B, C, D, AB, AC, AD, BC, BD, CD}
- Iteration
  - At each iteration,
    enumerate a conjunction of existing features that are not present in the current pool, e.g. ABC or ABD
    find the one that contributes the most on the error in the value function approximation and put it to the pool

iFDD [Geramifard et al. 2011]



Batch-iFDD – incremental feature dependency discovery

- Key techniques
  - Matching pursuit (MP)
    - Finding a new feature that contributes the most on value function approximation
  - Projection operators [to do MP]
    - Actual computation of Bellman Error

Matching pursuit (MP) is a sparse approximation algorithm which involves finding the "best matching" projections of multidimensional data onto the span of an over-complete (i.e., redundant) dictionary D. The basic idea is to approximately represent a signal f from Hilbert space H as a weighted sum of finitely many functions g_{γ_n} (called atoms) taken from D. An approximation with N atoms has the form

$$f(t) \approx \hat{f}_N(t) := \sum_{n=1}^{N} a_n g_{\gamma_n}(t)$$

where a_n is the scalar weighting factor (amplitude) for the atom $g_{\gamma_n} \in D$. Normally, not every atom in D will be used in this sum. Instead, matching pursuit chooses the atoms one at a time in order to maximally (greedily) reduce the approximation error. This is achieved by finding the atom that has the biggest inner product with the signal (assuming the atoms are normalized), subtracting from the signal an approximation that uses only that one atom, and repeating the process until the signal is satisfactorily decomposed, i.e., the norm of the residual is small, where the residual after calculating γ_N and a_N is denoted by

$$R_{N+1} = f - \hat{f}_N.$$

Algorithm Matching Pursuit

Input: Signal: $f(t)$, dictionary D.

Output: List of coefficients $(a_n)_{n=1}^N$ and indices for corresponding atoms $(\gamma_n)_{n=1}^N$.

Initialization:

$R_1 \leftarrow f(t)$;

$n \leftarrow 1$;

Repeat:

Find $g_{\gamma_n} \in D$ with maximum inner product $|\langle R_n, g_{\gamma_n} \rangle|$;

$a_n \leftarrow \langle R_n, g_{\gamma_n} \rangle / \|g_{\gamma_n}\|^2$;

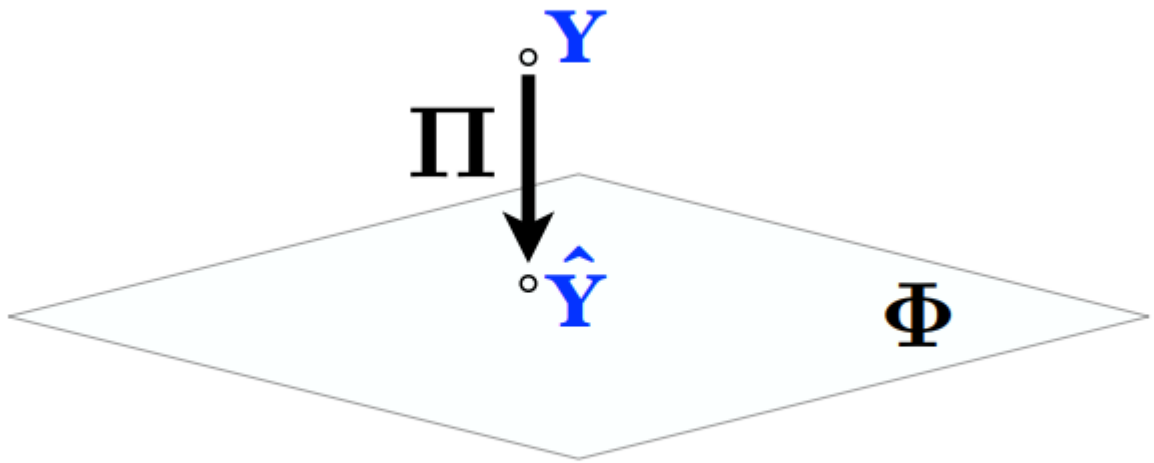
$R_{n+1} \leftarrow R_n - a_n g_{\gamma_n}$;

$n \leftarrow n + 1$;

Until stop condition (for example: $\|R_n\| < \text{threshold}$)

return

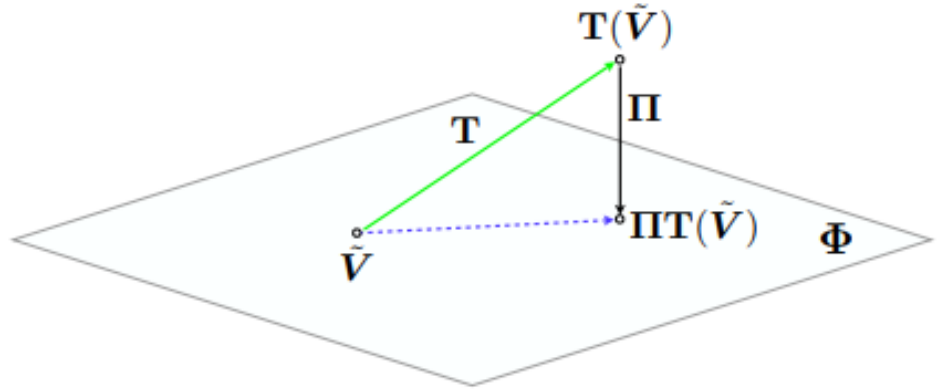
Matching Pursuit



Given a set of features at each step, add the feature which **reduces the error the most**.

$$||\mathbf{Y} - \hat{\mathbf{Y}}||$$

Feature Selection in RL



LSTD : $||\tilde{\mathbf{V}} - \mathbf{\Pi T}(\tilde{\mathbf{V}})||$

Batch-iFDD

- Run iFDD in Batch: Add new feature (conjunction) with highest error reduction (akin to OMP-TD).
- *Theorem*: iFDD in batch approximately finds the feature with the best guaranteed error reduction.

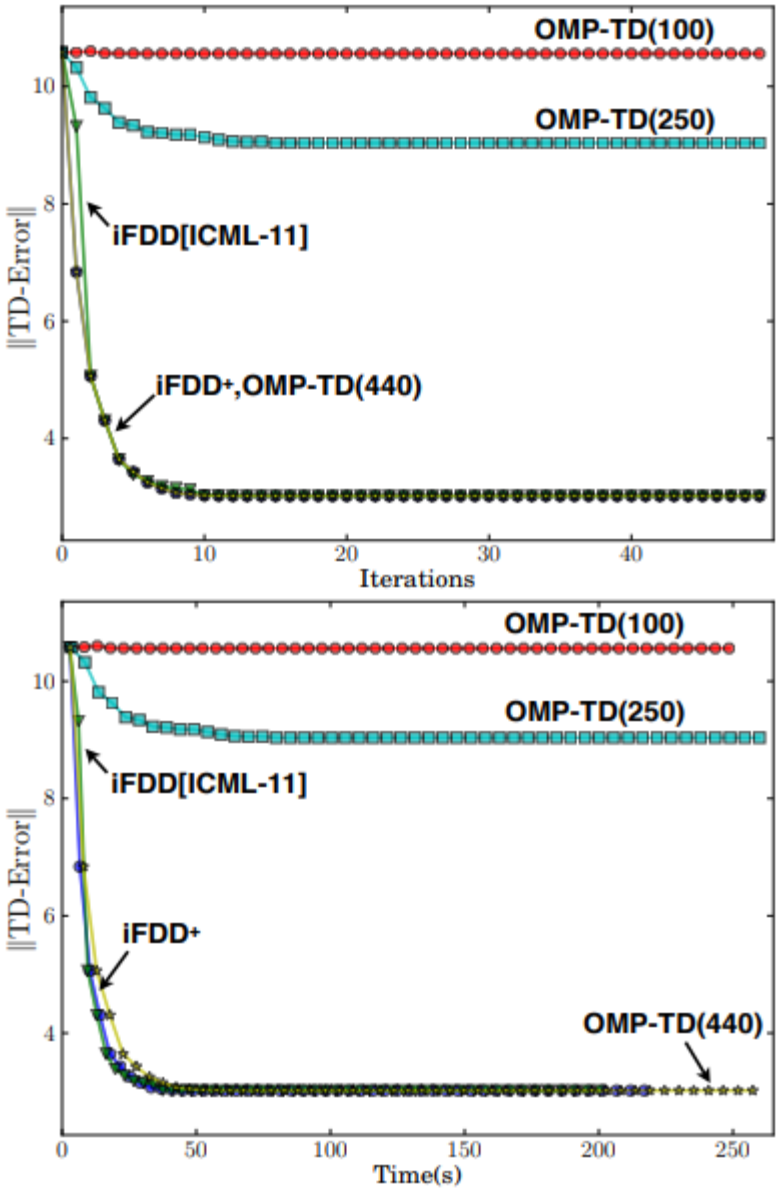
$$||\tilde{\mathbf{V}} - \mathbf{\Pi T}(\tilde{\mathbf{V}})||$$

$$\tilde{f}_1^* = \underset{f \in \texttt{pair}(\mathcal{X})}{\text{argmax}} \frac{|\sum_{i \in \{1,\cdots,m\}, \phi_f(s_i)=1} \delta_i|}{\sqrt{\sum_{i \in \{1,\cdots,m\}, \phi_f(s_i)=1} 1}} \qquad \text{iFDD}^+$$

$$\tilde{f}_2^* = \underset{f \in \texttt{pair}(\mathcal{X})}{\text{argmax}} \sum_{i \in \{1,\ldots,m\}, \phi_f(s_i)=1} |\delta_i| \qquad \text{iFDD}$$ [Geramifard et al. 2012]

previous approaches

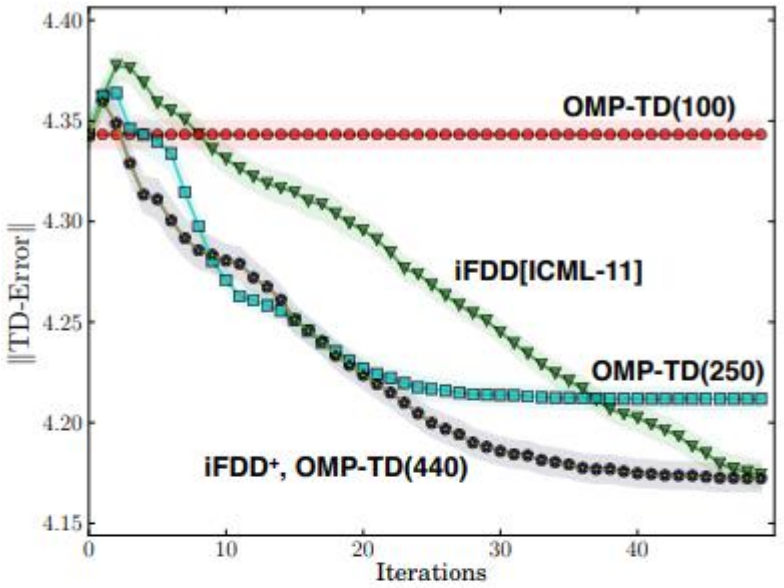
- MP is a general framework
- The difference is how to construct a pool of feature sets
  - Generate Bellman Error Basis Function on fly [Parr et al. 2007]
  - Initialize with some feature sets
    - iFDD [Geramifard et al. 2012]
    - OMP-TD [Painter-Wakefield and Parr, 2012] → orthogonal MP

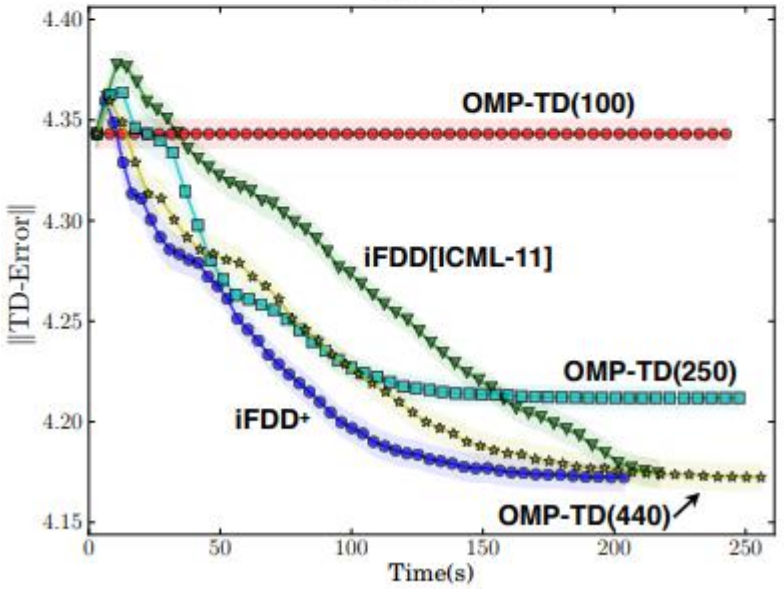


(a) MountainCar

Experiments

- Mountain car
- Full state space
  - 20 (positions) x 20 (velocities)
  - Base features : 40 = 20 + 20
- Can’t see how many features are used in FDD+

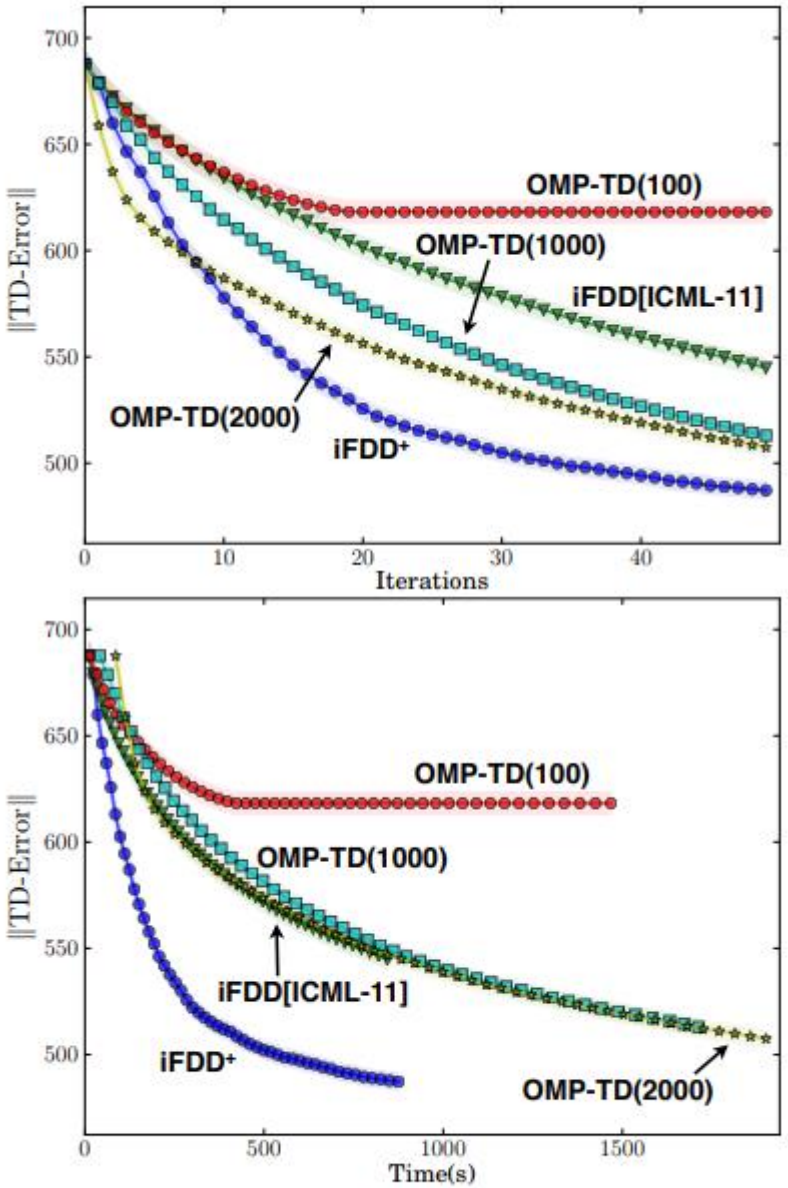




(b) Inverted Pendulum

Experiments

- Inverted Pendulum
- Full state space
  - 20 (positions) x 20 (velocities)
  - Base features : 40 = 20 + 20
- Can’t see how many features are used in FDD+



(c) System Administrator

Experiments

- Sys admin
  - 20 binary propositions (turn on/off computer) → 2^20 state space
- Full state space
  - 2^20
  - Base features : 40 [each computer on/off]
- Can’t see how many features are used in FDD+
  - iFDD+ discovered features with 8 terms