Tensor Optimization with Group Lasso for Multi-agent Predictive State Representation

Biyang Ma[a], Jing Tang[a], Bilian Chen[b,c], Yinghui Pan[d,*], Yifeng Zeng[a,*]

[a]*Department of Computer and Information Sciences, Northumbria University, UK*
[b]*Department of Automation, Xiamen University, Xiamen 361005, China*
[c]*Xiamen Key Lab. of Big Data Intelligent Analysis and Decision, Xiamen 361005, China*
[d]*College of Computer Science and Software Engineering, Shenzhen University, China*

Abstract

Predictive state representation (PSR) is a compact model of dynamic systems that represents state as a vector of predictions about future observable events. It is an alternative to a partially observable Markov decision process (POMDP) model in dealing with a sequential decision-making problem under uncertainty. Most of the existing PSR research focuses on the model learning in a single-agent setting. In this paper, we investigate a multi-agent PSR model upon available agents interaction data. It turns out to be rather difficult to learn a multi-agent PSR model especially with limited samples and increasing number of agents. We resort to a tensor technique to better represent dynamic system characteristics and address the challenging task of learning multi-agent PSR problems based on tensor optimization. We first focus on a two-agent scenario and use a third order tensor (system dynamics tensor) to capture the system interaction data. Then, the PSR model discovery can be formulated as a tensor optimization problem with group lasso, and an alternating direction method of multipliers is called for solving the embedded subproblems. Hence, the prediction parameters and state vectors can be directly learned from the optimization solutions, and the transition parameters can be derived via a linear regression. Subsequently, we generalize the tensor learning approach in a multi(N >2)-agent PSR model, and analyze the computational complexity of the learning algorithms. Experimental results show that the tensor optimization approaches have provided promising performances on learning a multi-agent PSR model over multiple problem domains.

Keywords: Predictive state representations, Tensor optimization, Alternating direction method of multipliers, Group lasso

1. Introduction

Predictive state representation (PSR) is an effective approach for modelling dynamic systems and it represents environmental states as a vector of predictions about future observable

[*]Corresponding author

Email addresses: biyang.ma@northumbria.ac.uk (Biyang Ma), jing.tang@northumbria.ac.uk (Jing Tang), blchen@xmu.edu.cn (Bilian Chen), yinghuipan.uk@gmail.com (Yinghui Pan), yifeng.zeng@northumbria.ac.uk (Yifeng Zeng) events (tests) conditioned on observed events (histories) in the past [17]. Main interests in the PSR research lie in two topics: one is core test discovery while the other is PSR model learning. Most of current research works are devoted to solving these two problems in a single-agent setting while the algorithmic efficiency or reliability still needs to be improved. To the best of our knowledge, extending PSR to a multi-agent setting has not been explored in the research. This is partially due to the fact that it is rather difficult to obtain sufficient and noiseless data of a multi-agent system especially in a large or complex problem domain.

For learning a single agent PSR model, one popular approach is based on system dynamics matrix [27], and the expected performance has been well presented in the literatures, e.g., the search based technique [12, 29, 11], the spectral learning approach [2, 16] and the compressed sensing approach [9, 8]. However, the capabilities of the current approaches are still largely affected by the sizes of observation space and system dynamics matrix, since the approaches typically consider the number of combinations of observation sequences.

In a multi-agent system, the situation is even more serious due to the increasing number of agents and their interactions. Moreover, a system dynamics matrix may not be able to demonstrate potential relationships among the joint tests through only two dimensions, since they are manually stored in its columns according to the length-lexicographical ordering. Meanwhile, the high number of interaction data demands complex manipulation of many high dimensional matrices. Consequently, the computation is rather costly. In addition, a multi-agent planning problem often involves insufficient training data and the data is not noiseless in most of cases. It would be very difficult to obtain a satisfactory PSR model based on a system dynamics matrix. Thus, a system dynamics matrix may not be an appropriate tool for learning a multi-agent PSR model, especially when the number of agents increases.

In this paper, we investigate a dynamical system with more than one agent and construct their interaction data as a high dimensional system dynamics tensor for learning a multi-agent PSR model. We utilize the tensor structure to extract embedded key information from its own elements by viewing multiple dimensions at the same time (even though the data is noisy). Specifically, for core test discovery, we formulate the problem of discovering a core joint test set as a tensor optimization problem with a group lasso structure and adopt an *alternating direction method of multipliers* (ADMM) to solve it efficiently. It is worth to mention that we don't need to specify the size of the core joint test set in advance and it will be automatically determined when the optimization problem is solved. This is the main difference from other PSR learning techniques.

With the benefit of the sparse optimization solution, we learn state vectors and prediction parameters in a straightforward manner. Inspired by the transformed PSR model [24], we learn model transition parameters through a linear regression after establishing a set of auxiliary matrices. We also utilize a sparse representation of tensor and mapping technique in order to skip all the time-consuming and memory-expensive operations in the proposed algorithm. Therefore, the PSR model of the underlying system can be built accordingly. We test the learning of PSR models for multi-agent systems on several problem domains including one extremely large domain. The experimental results demonstrate that the new tensor approaches achieve the expected performance.

The rest of this paper is organized as follows. We introduce mathematical notations and operations in Section 2 and present technical background of PSR models in Section 3. Section 4 is devoted to a theoretical analysis of learning PSR models of dynamic systems in a two-agent setting. We then extend it to a setting of multiple agents (more than two) in Section 5. We conduct experimental study in four commonly used domains and their variants to test the new PSR models in Section 6. In Section 7, we briefly discuss related works on learning PSR models. Finally, we conclude our work and suggest a few directions in future work in Section 8.

2. Preparations

Throughout this paper, we uniformly use the lower case letters (e.g., x), the boldface lower case letters (e.g., $\boldsymbol{x} = (x_i)$), the capital letters (e.g., $X = (x_{ij})$), and the calligraphic letters (e.g., $\mathcal{X} = (x_{i_1 i_2 \ldots i_d})$) to denote scalars, vectors, matrices, and higher order (order three or more) tensors, respectively. We use subscripts to denote elements of a vector, a matrix, or a tensor, e.g., $\mathcal{X}_{ijk}$ represents the (ijk)-th entry of $\mathcal{X}$. The main notations, functions, and operations on matrices or tensors are listed in Table 1.

Table 1. Summary of mathematical notations

Notations	Descriptions
$\operatorname{sgn}(\cdot)$	The sign function
$\operatorname{vec}(\cdot)$	The vectorization of a matrix
$\operatorname{diag}(\cdot)$	The diagonal matrix
$\\|\cdot\\|_F$	The Frobenius norm of a matrix or tensor
$*, \otimes$	The Hadamard product and the Kronecker product
$\times_k$	The mode-k product of a tensor with a matrix
I	A selected subset of subscripts of a vector, matrix, or tensor
$\mathcal{I}(\mathcal{S}', \mathcal{S})$	An index set that records the indices of elements of subset $\mathcal{S}'$ in set $\mathcal{S}$
A^T, A^{-1}	The transpose and (pseudo-)inverse of a matrix A
$\\|A\\|_{p,0}$, $\\|A\\|_{p,1}$	The $L_{p,0}$-norm and $L_{p,1}$-norm of a matrix A $(1 \leq p \leq \infty)$
$A_{i:}, A_{:j}$	The i-th row vector and j-th column vector of a matrix A
$A_{\mathcal{I}}$	The submatrix of A consisting of rows indicated by set $\mathcal{I}$
$A_{I:}, A_{:I}$	The submatrix of A consisting of rows or columns indicated by set I
$\mathcal{A}_{(k)}$	The mode-k matricization of a tensor $\mathcal{A}$
$\mathcal{A}_{::k}$	The k-th frontal slice of a tensor $\mathcal{A}$

A tensor is a multidimensional array. It is a well developed method for dealing with multidimensional data, which widely appears in many applications, e.g. bioinformatics, computer vision and so on. The order of a tensor is the number of its dimensions. A vector is a first order tensor, a matrix is a second order tensor, and tensors of order three or higher are called high-order tensors. The Euclidean norm of a vector $\boldsymbol{a} \in \mathbb{R}^n$ is denoted by $\|\boldsymbol{a}\|_2$, the Frobenius norm of a matrix $A \in \mathbb{R}^{m \times n}$ is denoted by $\|A\|_F$, and the Frobenius norm of a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is denoted by $\|\mathcal{A}\|_F$ as well, i.e.,

$$\|\boldsymbol{a}\|_2 = \left(\sum_{i=1}^{n} {a_i}^2 \right)^{1/2}, \ \|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} {A_{ij}}^2 \right)^{1/2}, \ \|\mathcal{A}\|_F = \left(\sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_d=1}^{n_d} {\mathcal{A}_{i_1 i_2 \ldots i_d}}^2 \right)^{1/2}.$$

By considering each column vector $A_{:j}$ ($j = 1, 2, \dots, n$) as a group, the matrix $L_{2,1}$-norm [6] is the sum of the L_2-norms of all column vectors, defined as

$$\|A\|_{2,1} := \|(\|A_{:1}\|_2, \|A_{:2}\|_2, \dots, \|A_{:n}\|_2)\|_1 = \sum_{j=1}^{n} \|A_{:j}\|_2 = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} {A_{ij}}^2 \right)^{1/2} .$$

The matrix $L_{2,1}$-norm is commonly used to study *group sparsity* due to the fact that $\|A_{:j}\|_2 = 0$ implies the vector $A_{:j} = \mathbf{0}$. If a matrix has a natural grouping of its elements and the elements within a group are likely to be either all zeros or all nonzeros, then $L_{2,1}$-norm is normally adopted in some convex optimization models to facilitate group sparsity replacing the $L_{2,0}$-norm of the matrix. Here, the matrix $L_{2,0}$-norm is defined to be the number of nonzero column vectors, i.e.,

$$\|A\|_{2,0} := \|(\|A_{:1}\|_2, \|A_{:2}\|_2, \dots, \|A_{:n}\|_2)\|_0.$$

One can naturally extend the $L_{2,0}$ and $L_{2,1}$ norms of a matrix to the $L_{r,0}$ and $L_{r,1}$ norms of a matrix for any $1 \le r \le \infty$, respectively, i.e.,

$$\|A\|_{r,0} := \|(\|A_{:1}\|_r, \|A_{:2}\|_r, \dots, \|A_{:n}\|_r)\|_0 ,$$

and

$$\|A\|_{r,1} := \|(\|A_{:1}\|_r, \|A_{:2}\|_r, \dots, \|A_{:n}\|_r)\|_1 .$$

It is easy to see that for any $1 \le r \le \infty$, $\|A\|_{r,0} = \|A\|_{2,0}$, i.e., all $\|A\|_{r,0}$'s are the same. However, $\|A\|_{r,1} \ge \|A\|_{r',1}$ if $1 \le r \le r' \le \infty$.

One important tensor operation is to represent its elements into a matrix called *matricization*, also known as *matrix unfolding* or *matrix flattening*. There are d modes for tensor $\mathcal{A}$, namely, mode-1, mode-2, ..., mode-d. Denote the *mode-k matricization* of $\mathcal{A}$ to be $A_{(k)}$. Then, the $(i_1 i_2 \dots i_d)$-th entry of the tensor $\mathcal{A}$ is mapped to the $(i_k j)$-th entry of the matrix $A_{(k)} \in \mathbb{R}^{n_k \times \prod_{\ell \neq k} n_\ell}$, where

$$j = 1 + \sum_{1 \le \ell \le d, \ell \neq k} (i_\ell - 1) \prod_{1 \le t \le \ell - 1, t \neq k} n_t., \tag{1}$$

The other important tensor operation in this paper is the mode-k product of a tensor $\mathcal{A}$ by a matrix $B \in \mathbb{R}^{m \times n_k}$. The multiplication returns a new tensor in $\mathbb{R}^{n_1 \times n_2 \times \dots \times n_{k-1} \times m \times n_{k+1} \times \dots \times n_d}$, denoted as $\mathcal{A} \times_k B$, whose $(i_1 i_2 \dots i_{k-1} \ell i_{k+1} \dots i_d)$-th entry is computed as

$$(\mathcal{A} \times_k B)_{i_1 i_2 \dots i_{k-1} \ell \, i_{k+1} \dots i_d} = \sum_{i_k = 1}^{n_k} \mathcal{A}_{i_1 i_2 \dots i_{k-1} i_k i_{k+1} \dots i_d} B_{\ell i_k}.$$

The mode-k product can be rewritten in terms of matricizations, i.e.,

$$\mathcal{Y} = \mathcal{A} \times_k B \Longleftrightarrow Y_{(k)} = B A_{(k)}., \tag{2}$$

3. Technical Background of PSR Models

Linear PSR models are a well-studied version of PSR models for modelling dynamic systems [17]. The dynamic system considered in this paper is a discrete-time and controllable dynamic system with some agents who perform a sequence of actions and receive a sequence of observations from the environment with one action and one observation per time step. In order to learn a multi-agent PSR model, we should extend all necessary definitions of a single-agent PSR model. Basically, we use non-bold letters to denote single-agent related notations, and bold letters for the multi-agent case, e.g., a represents an action in the single-agent system and $\mathbf{a}$ represents a joint action in the multi-agent case. The main PSR-related notations are summarized in Table 2.

Table 2. Summary of PSR-related notations

Descriptions	Notations	
	Single-agent PSR	Multi-agent PSR
(Joint) Action	$a \leftarrow a^i$	$\mathbf{a} \leftarrow a^{i_1}a^{i_2}\dots a^{i_N}$
(Joint) Observation	$o \leftarrow o^{i'}$	$\mathbf{o} \leftarrow o^{i'_1}o^{i'_2}\dots o^{i'_N}$
(Joint) Action-observation	$ao \leftarrow a^i o^{i'}$	$\mathbf{ao} \leftarrow a^{i_1}o^{i'_1}\dots a^{i_N}o^{i'_N}$
(Joint) Test	$t \leftarrow a_1 o_1 \dots a_l o_l$	$\mathbf{t} \leftarrow t^{(1)}_{i_1}\dots t^{(N)}_{i_N} \leftarrow \mathbf{a}_1\mathbf{o}_1\dots\mathbf{a}_l\mathbf{o}_l$
(Joint) History	$h \leftarrow a_1 o_1 \dots a_{\lvert h\rvert} o_{\lvert h\rvert}$	$\mathbf{h} \leftarrow \mathbf{a}_1\mathbf{o}_1\dots\mathbf{a}_{\lvert h\rvert}\mathbf{o}_{\lvert h\rvert}$
Null (joint) history	ϕ	ϕ
Core (joint) test	$q \leftarrow a_1 o_1 \dots a_n o_n$	$\mathbf{q} \leftarrow q^{(1)}_{i_1}\dots q^{(N)}_{i_N} \leftarrow \mathbf{a}_1\mathbf{o}_1\dots\mathbf{a}_n\mathbf{o}_n$
Set of (joint) actions	$A = \{a^1, a^2, \dots, a^{\lvert A\rvert}\}$	$\mathcal{A} = \{\mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^{\lvert\mathcal{A}\rvert}\}$
Set of (joint) observations	$O = \{o^1, o^2, \dots, o^{\lvert O\rvert}\}$	$\mathcal{O} = \{\mathbf{o}^1, \mathbf{o}^2, \dots, \mathbf{o}^{\lvert\mathcal{O}\rvert}\}$
Set of (joint) tests	$T = \{t_1, t_2, \dots, t_{\lvert T\rvert}\}$	$\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_{\lvert\mathcal{T}\rvert}\}$
Set of (joint) histories	$H = \{h_1(\phi), h_2, \dots, h_{\lvert H\rvert}\}$	$\mathcal{H} = \{\mathbf{h}_1(\phi), \mathbf{h}_2, \dots, \mathbf{h}_{\lvert\mathcal{H}\rvert}\}$
Set of core (joint) tests	$\mathrm{Q} = \{q_1, q_2, \dots, q_{\lvert\mathrm{Q}\rvert}\}$	$\mathbf{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_{\lvert\mathbf{Q}\rvert}\}$
Set of core (joint) histories	$\mathrm{H} = \{h_1, h_2, \dots, h_{\lvert\mathrm{H}\rvert}\}$	$\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_{\lvert\mathbf{H}\rvert}\}$
Set of (joint) histories with length s	$H_s = \{h \mid h \in \mathrm{H}, \lvert h\rvert = s\}$	$\mathcal{H}_s = \{\mathbf{h} \mid \mathbf{h} \in \mathcal{H}, \lvert\mathbf{h}\rvert = s\}$
Time step	s	s
State vector	$p(\mathrm{Q}\mid h_s)$	$p(\mathbf{Q}\mid\mathbf{h}_s)$
Projection vector	m_t	$\boldsymbol{m}_{\mathbf{t}}$
One-step projection vector	m_{ao}	$\boldsymbol{m}_{\mathbf{ao}}$
Transition matrix	M_{ao}	$M_{\mathbf{ao}}$
State matrix	$Q \leftarrow p(\mathrm{Q}\mid H)$	$Q \leftarrow p(\mathbf{Q}\mid\mathbf{H})$
Core test tensor	-	$\mathcal{Q} \leftarrow \mathcal{D}_{\mathbf{Q},\mathcal{H}}$
System dynamics tensor	-	$\mathcal{D}$

3.1. A Single-agent PSR Model

In a single-agent PSR setting, an agent interacts with a system by performing an action a and receiving an observation o step by step. Every possible action is selected from its action set $A = \{a^1, a^2, \dots, a^{|A|}\}$ and every possible observation must be chosen from its observation set $O = \{o^1, o^2, \ldots, o^{|O|}\}$. At a given time step s, the agent has already experienced a *history* h of action-observation sequence with length s, i.e., $h_s = a_1 o_1 \ldots a_s o_s$. After the agent applies ao at this time, we update its current history h_s to a new history $h_{s+1} = h_s ao$. An l steps of action-observation pairs performed by the agent after time step s is called as the *test* t, i.e., $t = a_1 o_1 \ldots a_l o_l$. All possible histories that have been memorized form a *history set* $H = \{h_1, h_2, \ldots, h_{|H|}\}$ and all possible tests in the future form a *test set* $T = \{t_1, t_2, \ldots, t_{|T|}\}$. The conditional probability of a test t given the history $h_s = \{h | h \in \mathrm{H}, |h| = s\}$, say $p(t|h_s)$, can be computed by a *prediction equation*, as follows:

$$p(t|h_s) = f(p(\mathrm{Q}|h_s)) = p(\mathrm{Q}|h_s) m_t,$$

where f is the project function, $p(\mathrm{Q}|h_s)$ is the state vector given history h_s, and m_t is the projection vector of test t. Given a new history $h_{s+1} = h_s ao$, the new state vector can be calculated by an *updating equation*, i.e.,

$$p(\mathrm{Q}|h_{s+1}) = \frac{p(\mathrm{Q}|h_s) M_{ao}}{p(\mathrm{Q}|h_s) m_{ao}},$$

where m_{ao} is the specific projection vector for test $t = ao$ and M_{ao} is the transition matrix consisting of one-step extension projection vectors m_{aoq_i} $(\forall\, q_i \in \mathrm{Q})$.

To summarize, a linear PSR model for a controlled partially observable system with a single-agent setting has the parameters $\{A, O, \mathrm{Q}, \{m_{ao}\}, \{M_{ao}\}, p(\mathrm{Q}|\phi)\}$: the set of actions A, the set of observations O, the set of core tests Q, the model parameters m_{ao} and M_{ao} $(\forall\, a \in A, o \in O)$, and an initial prediction vector $p(\mathrm{Q}|\phi)$ with ϕ being the null history at initial time step $s = 0$.

3.2. A multi-agent PSR Model

In this paper, we extend a single-agent PSR model to a multi-agent setting. We consider a centralised multiagent system, which is similar to the setting of multi-agent Markov decision process [25], where a number of agents share common knowledge of a physical environment and their actions have joint impact on the environment. In this work, we do not consider their interactions as a specialization of stochastic games and will not involve recursive modelling of other agents in the multi-agent setting such as interactive partially observable Markov decision process [7] or interactive dynamic influence diagrams [31].

For a dynamic system with $N(\geq 2)$ agents, say agent $(1), (2), \ldots, (N)$, they operate a *joint action* $\mathbf{a} = a^{i_1} a^{i_2} \ldots a^{i_N}$ from a set of executable joint actions $\mathcal{A} = \{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^{|\mathcal{A}|}\}$ and sense a *joint observation* $\mathbf{o} = o^{i_1} o^{i_2} \ldots o^{i_N}$ from a set of observable joint observations $\mathcal{O} = \{\mathbf{o}^1, \mathbf{o}^2, \ldots, \mathbf{o}^{|\mathcal{O}|}\}$. In this progress, each agent (i) can only executes one *action* a from a set of executable actions $\mathcal{A}^{(i)}$ by using a stochastic policy $\pi^{(i)} : \mathcal{O}^{(i)} \times \mathcal{A}^{(i)} \rightarrow [0, 1]$, which cause the system transfers to the next state according to the state transition function $\mathbf{T} : \mathcal{S} \times \mathcal{A}^{(1)} \cdots \times \mathcal{A}^{(N)} \rightarrow \mathcal{S}$, where $\mathcal{S}$ is the state space of the system. Each agent (i) has a independent belief about the current state of the dynamic system. Each of them privately senses one *observation* o from a set of observable observation $\mathcal{O}^{(i)}$, which is correlated with current state of the system and generated by an observation function $\mathbf{O} : \mathcal{S} \rightarrow \mathcal{O}^{(i)}$, and obtains rewards as a function of the state and agents action. And this paper doesn't specify the communication mechanism or the interaction rule between the agents. When this is processed for s steps, a *joint history* $\mathbf{h}$ is used to describe the entire sequence of past action-observation pairs with length s, i.e., $\mathbf{h}_s = \mathbf{a}_1\mathbf{o}_1\mathbf{a}_2\mathbf{o}_2 \ldots \mathbf{a}_s\mathbf{o}_s$. The joint history is updated to $\mathbf{h}_{s+1} = \mathbf{h}_s\mathbf{ao}$ after agents taking a new joint action $\mathbf{a}$ and sensing a new joint observation $\mathbf{o}$. All previous action-observations pairs form a *joint history set* $\mathcal{H} = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{|\mathcal{H}|}\}$. At this time step s, agents may expect to further experience a sequence of joint action-observation pairs with length l, which is called a *joint test* $\mathbf{t}$, i.e., $\mathbf{t} = \mathbf{a}_1\mathbf{o}_1\mathbf{a}_2\mathbf{o}_2 \ldots \mathbf{a}_l\mathbf{o}_l$. All the joint tests in the future together constitute a *joint test set* $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{|\mathcal{T}|}\}$.

Analogous to the linear single-agent PSR model, we use *prediction equation* to predict the conditional probability of a joint test $\mathbf{t}$ given an arbitrary history $\mathbf{h}_s \in \mathcal{H}_s = \{\mathbf{h}|\mathbf{h} \in \mathcal{H}, |\mathbf{h}| = s\}$ at time step s, i.e.,

$$p(\mathbf{t}|\mathbf{h}_s) = f(p(\mathbf{Q}|\mathbf{h}_s)) = p(\mathbf{Q}|\mathbf{h}_s)\boldsymbol{m}_{\mathbf{t}}, \tag{3}$$

where the set of joint test $\mathbf{Q} = \{\mathbf{q}_i|\mathbf{q}_i \in \mathcal{T}, i \in \{1, 2, \ldots, n\}\}$ is called the *core joint test set*, the vector $p(\mathbf{Q}|\mathbf{h}_s)$ forms a *sufficient statistic* for the state of dynamic system at any time step s and $\boldsymbol{m}_{\mathbf{t}}$ is called the *projection vector* of test $\mathbf{t}$. We will denote $p(\mathbf{Q}|\mathbf{h}_s \leftarrow \mathbf{h}_k)$ as $p(\mathbf{Q}|\ \mathbf{h}_s)$ for any given joint history $\mathbf{h}_k \in \mathcal{H}$ ($k \in [1, |\mathcal{H}|\,]$) at a given time step $s = |\mathbf{h}_k|$ for simplicity. For $\mathbf{t}_i = \mathbf{t}_{i_1 \ldots i_N} = t^{(1)}_{i_1} \ldots t^{(N)}_{i_N}$, Eq. (3) becomes

$$p(t^{(1)}_{i_1} \ldots t^{(N)}_{i_N}|\mathbf{h}_s) = p(\mathbf{Q}|\mathbf{h}_s)\boldsymbol{m}_{t_{i_1},\ldots,t_{i_N}}. \tag{4}$$

Moreover, we use $\boldsymbol{m}_{\mathbf{ao}}$ to denote the projection vector $\boldsymbol{m}_{\mathbf{t}}$ for the one-step joint test $\mathbf{t} = \mathbf{ao}$.

The *updating equation* predicts a new state $p(\mathbf{Q}|\mathbf{h}_{s+1})$ from the old state $p(\mathbf{Q}|\mathbf{h}_s)$ after agents took a joint action $\mathbf{a} \in \mathcal{A}$ and sensed an observation $\mathbf{o} \in \mathcal{O}$ at any time step s. Given a new history $\mathbf{h}_{s+1} = \mathbf{h}_s\mathbf{ao} \in \mathcal{H}_{s+1}$ where $\mathcal{H}_{s+1} = \{\mathbf{h}\,|\,\mathbf{h} \in \mathcal{H}, |\mathbf{h}| = s + 1\}$, the update is computed as follows:

$$p(\mathbf{q}_i|\mathbf{h}_{s+1}) = \frac{p(\mathbf{aoq}_i|\mathbf{h}_s)}{p(\mathbf{ao}|\mathbf{h}_s)} = \frac{p(\mathbf{Q}|\mathbf{h}_s)\boldsymbol{m}_{\mathbf{aoq}_i}}{p(\mathbf{Q}|\mathbf{h}_s)\boldsymbol{m}_{\mathbf{ao}}}, \tag{5}$$

where the vector $\boldsymbol{m}_{\mathbf{aoq}_i}$ is $\boldsymbol{m}_{\mathbf{t}}$ for each one-step extension ($\mathbf{aoq}_i$) for all $\mathbf{a} \in \mathcal{A}, \mathbf{o} \in \mathcal{O}, \mathbf{q}_i \in \mathbf{Q}$. By arranging each projection vector $\boldsymbol{m}_{\mathbf{aoq}_i}$ according to the index of $\mathbf{q}_i$ in core joint test set $\mathbf{Q}$ into a *transition matrix* $M_{\mathbf{ao}}$, we have

$$M_{\mathbf{ao}} = [\boldsymbol{m}_{\mathbf{aoq}_1} \quad \boldsymbol{m}_{\mathbf{aoq}_2} \quad \cdots \quad \boldsymbol{m}_{\mathbf{aoq}_{|\mathbf{Q}|}}]. \tag{6}$$

Then, Eq. (5) can be rewritten as:

$$p(\mathbf{Q}|\mathbf{h}_{s+1}) = \frac{p(\mathbf{Q}|\mathbf{h}_s)M_{\mathbf{ao}}}{p(\mathbf{Q}|\mathbf{h}_s)\boldsymbol{m}_{\mathbf{ao}}}, \tag{7}$$

where the vectors $\{\boldsymbol{m}_{\mathbf{ao}}\}$ and matrices $\{M_{\mathbf{ao}}\}$ ($\forall\, \mathbf{a} \in \mathcal{A}, \mathbf{o} \in \mathcal{O}$) are called *prediction parameters* and *transition parameters* of a linear multi-agent PSR model, respectively. In general, given an initial prediction vector $p(\mathbf{Q}|\phi)$ for a null joint history $\phi \in \mathcal{H}$, the prediction vector $p(\mathbf{Q}|\mathbf{h}_s)$ can be computed step by step for any time s. On the other hand, based on the discussion above, given the core joint tests $\mathbf{Q}$ and *core joint history set* $\mathbf{H}$, we can compute the following parameters:

$$\begin{cases} \boldsymbol{m}_{\mathbf{ao}} = p(\mathbf{Q}|\mathbf{H})^{-1}p(\mathbf{ao}|\mathbf{H}) \\ \boldsymbol{m}_{\mathbf{aoq}_i} = p(\mathbf{Q}|\mathbf{H})^{-1}p(\mathbf{aoq}_i|\mathbf{H}), \end{cases} \tag{8}$$

where $p(\mathbf{Q}|\mathbf{H})$, $p(\mathbf{ao}|\mathbf{H})$ and $p(\mathbf{aoq}_i|\mathbf{H})$ can be estimated through available training data.

To conclude, a linear multi-agent PSR model in a controllable partially observable dynamic system has the parameters $\{\mathcal{A}, \mathcal{O}, \mathbf{Q}, \{\boldsymbol{m}_{\mathbf{ao}}\}, \{M_{\mathbf{ao}}\}, p(\mathbf{Q}|\phi)\}$: the joint action set $\mathcal{A}$, the joint observation set $\mathcal{O}$, the core joint test set $\mathbf{Q}$, the model parameters $\boldsymbol{m}_{\mathbf{ao}}$ and $M_{\mathbf{ao}}$ ($\forall\ \mathbf{a} \in \mathcal{A}, \mathbf{o} \in \mathcal{O}$), and an initial prediction vector $p(\mathbf{Q}|\phi)$. The process of finding $\mathbf{Q}$ is called the *discovery problem* and computing the projection vectors is called the *learning problem*.

For learning a multi-agent PSR model based on a tensor approach, a *system dynamics tensor* $\mathcal{D} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times \cdots \times n_{N+1}}$ is used to store interaction data of N agents, whose element is $\mathcal{D}_{i_1 i_2 \ldots i_N k} = p(\mathbf{t}|\mathbf{h}_k) = p(t^{(1)}_{i_1} \ldots t^{(N)}_{i_N}|\mathbf{h}_k)$ ($\forall\, \mathbf{t} \in \mathcal{T}, \mathbf{h}_k \in \mathcal{H}$). It can be estimated by the reset algorithm [12]. The *discovery problem* is then transferred into finding a minimal linearly independent set (i.e., core joint test set $\mathbf{Q}$) among mode-$(N+1)$ fibers of system dynamics tensor $\mathcal{D}$, so that the whole fibers listed in the set $\mathbf{Q}$ form a basis of the space spanned by the mode-$(N+1)$ fibers of tensor $\mathcal{D}$, and these fibers together form a subtensor of $\mathcal{D}$, namely *core test tensor* $\mathcal{Q}$. Meantime, the *system dynamics matrix* [17] can be employed as an alternative way for learning a multi-agent PSR model through the traditional matrix-based single-agent PSR learning algorithms (see Section 6). Each element of the matrix is also estimated by the reset algorithm [12], and its two dimensions represent joint histories and joint tests, respectively. The main difference between system dynamics matrix and system dynamics tensor is that we put all the joint tests of the multiple agents in only one dimension when constructing the former one. One is referred to [5] for more detailed discussions on system dynamics tensors.

4. Learning Two-Agent PSR Based on Tensor Optimization

In this section, we propose a new framework for learning two-agent PSR based on tensor optimization. In particular, we establish an optimization model to find the key ingredient of the PSR model, i.e., the core joint test set $\mathbf{Q}$ that solves the discovery problem. From the solutions of the optimization problem, we will further learn the model parameters (including prediction parameters and transition parameters) of the PSR model.

4.1. Discover a Core Joint Test Set

After constructing the interaction data of a two-agent dynamic system as a high dimensional system dynamic tensor, we solve the discovery problem for learning a multi-agent PSR model through tensor optimization. More specifically, we formulate the progress of finding a minimal linearly independent set (i.e., core joint test set) among mode-$(N+1)$ fibers of system dynamic tensor as an optimization model. Subsequently, we transform the optimization model into a group LASSO problem and solve it efficiently with the benefit of ADMM method and mapping technique. Then, we extract a core joint test set and construct the core test tensor of the PSR model with the fibers listed in the core joint test set.

4.1.1. Formulation of Optimization Model and Its Relaxation

For a two-agent dynamic system, we obtain its interaction data samples Φ_d and construct the system dynamics tensor $\mathcal{D} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$. Suppose that $t_i^{(1)}$ is the i-th test sequence of the first agent's test set $\mathcal{T}^{(1)}$, $t_j^{(2)}$ is the j-th test sequence of the second agent's test set $\mathcal{T}^{(2)}$, and $\mathbf{h}_k \in \mathcal{H}$ is the agents' joint history at time step $s = |\mathbf{h}_k|$. The element of the tensor can be represented as $\mathcal{D}_{ijk} = p(t_i^{(1)} t_j^{(2)} | \mathbf{h}_k)$. As discussed in Section 3, if the core test tensor $\mathcal{Q}$ has been extracted from the system dynamics tensor $\mathcal{D}$, then all the mode-3 fibers of $\mathcal{D}$ can be linearly represented by itself. That is, there exist two coefficient matrices $X_1 \in \mathbb{R}^{n_1 \times n_1}$ and $X_2 \in \mathbb{R}^{n_2 \times n_2}$, such that

$$\mathcal{D} = \mathcal{D} \times_1 X_1 \times_2 X_2. \tag{9}$$

In general, the number of solutions to the problem (9) may be infinite. We are interested in finding a sparse solution (X_1, X_2) that has the minimum number of nonzero columns. Let α_1 and α_2 be positive constant parameters, for any $1 \le r \le \infty$, we propose the following tensor optimization model:

$$\begin{array}{ll} \min\limits_{X_1, X_2} & \alpha_1 \left\| X_1 \right\|_{r,0} + \alpha_2 \left\| X_2 \right\|_{r,0} \\ \text{s.t.} & \mathcal{D} = \mathcal{D} \times_1 X_1 \times_2 X_2. \end{array} \tag{10}$$

We remark that for any r with $1 \le r \le \infty$ in the above model, its solutions must be the same since $\|A\|_{r,0} = \|A\|_{2,0}$ holds for any matrix A as discussed in Section 2. The way that we kept r in (10) without a specific value is for the flexibility in designing its solution methods.

In the following discussion, we consider the case $r = 2$. The difficulty to solve (10) lies in the L_0-norm part of the objective function. Hence, recall that many sparse optimization problems can be efficiently obtained via the L_1-norm relaxation, we now convert the model (10) to a variant model with the L_1-norm relaxation, i.e.,

$$\begin{array}{ll} \min\limits_{X_1, X_2} & \alpha_1 \left\| X_1 \right\|_{2,1} + \alpha_2 \left\| X_2 \right\|_{2,1} \\ \text{s.t.} & \mathcal{D} = \mathcal{D} \times_1 X_1 \times_2 X_2. \end{array} \tag{11}$$

The model (11) is actually a convex optimization problem and can be solved efficiently.

4.1.2. Group LASSO and ADMM Method

As discussed above, for a given tensor $\mathcal{D}$, there are sparse coefficient matrices X_1 and X_2 such that $\mathcal{D} = \mathcal{D} \times_1 X_1 \times_2 X_2$. In fact, the interaction data samples of agents in a dynamic environment may have various noises, and so it is necessary to approximately relax Eq. (9) so that the gap $\|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2\|_F$ is as small as possible. This becomes a well-known *least absolute shrinkage and selection operator* (LASSO) problem [23]. A basic LASSO method selects only one variable from a group of highly correlated variables. In this article, we use group LASSO [30] as a novel version of the Lasso method for addressing the issue of approximating the gap.

For our problem, we use a generalized form of the L_1-regularization LASSO, i.e., applying $L_{2,1}$-regularization (group LASSO) on groups of variables having some sparse characteristics.Hence, the relaxation model (11) can be transformed into a group LASSO problem as follows:

$$\min_{X_1, X_2} \ \frac{1}{2} \|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2\|_F^2 + \alpha_1 \left\| X_1 \right\|_{2,1} + \alpha_2 \left\| X_2 \right\|_{2,1}, \tag{12}$$

which is an unconstrained optimization problem.

We observe that X_1 and X_2 are not related in the model. It is a common approach to consider alternating optimization over the variable matrices X_1 and X_2, i.e., starting any two matrices X_1 and X_2, and then

(i) Fixing X_2, optimizing X_1:

$$\min_{X_1} \ \frac{1}{2}\|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2\|_F^2 + \alpha_1 \|X_1\|_{2,1} . \tag{13}$$

(ii) Fixing X_1, optimizing X_2:

$$\min_{X_2} \ \frac{1}{2}\|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2\|_F^2 + \alpha_2 \|X_2\|_{2,1} . \tag{14}$$

For the two subproblems (13) and (14), we construct two auxiliary tensors $\mathcal{A} = \mathcal{D} \times_2 X_2$ and $\mathcal{B} = \mathcal{D} \times_1 X_1$, respectively. Then the optimization model (13) is equivalent to

$$\min_{X_1} \ \frac{1}{2}\|X_1 A_{(1)} - D_{(1)}\|_F^2 + \alpha_1 \|X_1\|_{2,1} .$$

Similarly, the model (14) is equivalent to

$$\min_{X_2} \ \frac{1}{2}\|X_2 B_{(2)} - D_{(2)}\|_F^2 + \alpha_2 \|X_2\|_{2,1} .$$

Therefore, it can be seen that both models are in the following form:

$$\min_{X} \ \frac{1}{2}\|XA - D\|_F^2 + \lambda \|X\|_{2,1} . \tag{15}$$

where λ, A and D are given parameters.

To solve the subproblem (15), we adopt an alternating direction method of multipliers (ADMM) which is a commonly accepted efficient method; see [4] for more details. The ADMM method is a variable optimization performed by the variable segmentation technique and the augmented Lagrangian framework. Specifically, by introducing a new auxiliary variable Y for the optimization problem (15), we obtain

$$\begin{array}{ll} \min\limits_{X,Y} & \frac{1}{2}\|XA - D\|_F^2 + \lambda \|Y\|_{2,1} \\ \text{s.t.} & X = Y. \end{array}$$

The corresponding augmented Lagrangian function is

$$L_\mu (X, Y, Z) = \frac{1}{2} \|XA - D\|_F^2 + \lambda \|Y\|_{2,1} + \operatorname{tr}\left(Z^T (X - Y)\right) + \frac{\mu}{2} \|X - Y\|_F^2 ,$$

where $\operatorname{tr}(\cdot)$ is the trace operator, Z is the *Lagrange* multiplier, and $\mu > 0$ is a penalty parameter.

For $k = 0, 1, 2, \dots$, the ADMM method consists of the following iterations:

$$\text{fix } Y^k \text{ and } Z^k \text{ and update } X^{k+1}: \qquad X^{k+1} = \arg\min_X L_\mu(X), \tag{16}$$

$$\text{fix } X^{k+1} \text{ and } Z^k \text{ and update } Y^{k+1}: \qquad Y^{k+1} = \arg\min_Y L_\mu(Y), \tag{17}$$

$$\text{fix } X^{k+1} \text{ and } Y^{k+1} \text{ and update } Z^{k+1}: \qquad Z^{k+1} = Z^k + \mu\left(X^{k+1} - Y^{k+1}\right). \tag{18}$$

The objective function $L_\mu(X)$ in problem (16) is equal to

$$L_\mu(X) = \frac{1}{2}\|XA - D\|_F^2 + \operatorname{tr}\left(Z^T X\right) + \frac{\mu}{2}\|X - Y\|_F^2 + c_0,$$

where c_0 is a constant. The optimal solution of (16) admits an analytical formula, i.e.,

$$X^* = (DA^T - Z + \mu Y)\left(AA^T + \mu E\right)^{-1},$$

where E is an identity matrix. Moreover, the objective function $L_\mu(Y)$ in problem (17) is equal to

$$\begin{aligned} L_\mu(Y) &= \lambda\|Y\|_{2,1} - \operatorname{tr}\left(Z^T Y\right) + \frac{\mu}{2}\|Y - X\|_F^2 \\ &= \lambda\|Y\|_{2,1} + \frac{\mu}{2}\left\|Y - \left(X + \frac{1}{\mu}Z\right)\right\|_F^2. \end{aligned}$$

Therefore, (17) can be separated into some subproblems, whose total number is the number of columns of X. By letting $G = X + \frac{1}{\mu}Z$, each optimal column vector of the variable Y can be calculated from the column vector of G, i.e.,

$$Y_{:i}^* = \max\left\{\|G_{:i}\|_2 - \frac{\lambda}{\mu}, 0\right\}\frac{G_{:i}}{\|G_{:i}\|_2}.$$

In implementing the ADMM, the stopping criterion of the above algorithm can be set as

$$\|X^{k+1} - Y^{k+1}\|_F \le \epsilon \quad \text{or} \quad \max\left\{\|X^{k+1} - X^k\|_F, \|Y^{k+1} - Y^k\|_F\right\} \le \epsilon.$$

The former one is called the primal residual and the latter is the difference between successive iterations.

We present the ADMM method to solve the optimization problem (15) in Algorithm 1. We first initialize the auxiliary variable Y^0 and the *Lagrange* multiplier Z^0 with identity matrix (line 1). Then, we proceed with a sub-progress, which begins with fixing Y^k and Z^k and updating X^{k+1} (line 3), follows by fixing X^{k+1} and Z^k and updating Y^{k+1} (lines 4-7). We subsequently fix X^{k+1} and Y^{k+1} and update Z^{k+1} (line 8). The procedure is iterated until the stopping criterion is satisfied (lines 2-9). Finally, the coefficient matrix X^{k+1} is returned (line 10).

The global convergence of Algorithm 1 can be easily established, similar to that of [4, Section 3]. It is worth mentioning that the convergent property satisfies for any parameter $\mu > 0$. The iteration complexity of ADMM has been studied in the literature (see [4] and the references therein). It is shown that the ADMM algorithm has $\mathbf{O}(1/\epsilon)$ convergence rate, under some mild conditions on the problem.

```
Data: matrices $A$ and $D$, positive scalars $\lambda$ and $\mu$, and a termination tolerance $\epsilon$.
Result: an optimal $X$.
1  Initialize $Y^0$ and $Z^0$ ;
2  while $\|X^{k+1} - Y^{k+1}\|_F > \epsilon \quad or \quad \max\left\{\|X^{k+1} - X^k\|_F,\ \|Y^{k+1} - Y^k\|_F\right\} > \epsilon$ do
3      $X^{k+1} = (DA^T - Z^k + \mu Y^k)\left(AA^T + \mu\, E\right)^{-1}$;
4      $G^{k+1} = X^{k+1} + \frac{1}{\mu} Z^k$;
5      for each column of $Y^{k+1}$ do
6          $Y^{k+1}_{:i} = \max\left\{\|G^{k+1}_{:i}\|_2 - \frac{\lambda}{\mu}, 0\right\} \frac{G^{k+1}_{:i}}{\|G^{k+1}_{:i}\|}$;
7      end
8      $Z^{k+1} = Z^k + \mu\,(X^{k+1} - Y^{k+1})$;
9  end
10 $X \leftarrow X^{k+1}$.
```

Algorithm 1: ADMM$(A, D, \lambda, \mu, \epsilon)$ for the optimization problem (15)

Theorem 4.1. *The sequence $\{(X^k, Y^k, Z^k)\}$ generated by Algorithm 1 converges to an optimal solution of (15) for any starting point.*

Therefore, by applying the ADMM procedure (Algorithm 1) on the two sub-problems (13) and (14), we can further solve problem (12) by alternating optimization. The procedure is presented in Algorithm 2, which terminates when it meets the stopping criterion, i.e., the objective function of problem (12) is small enough or could not be minimized any more. We will use the grid search method to choose suitable parameters; see Section 6.2.

In Algorithm 2, we first initialize the coefficient matrix X_2^0 with an identity matrix, generate the sparse tables I, I_1 and I_2 and compute matrices $D_{(1)}$ and $D_{(2)}$ (lines 1-3). We compute the auxiliary matrix $A^k_{(1)}$ via a mapping technique (lines 5-6) and update X_1^{k+1} via applying Algorithm 1 (line 7). We then calculate the auxiliary matrix $B^k_{(2)}$ via a mapping technique (lines 8-9) and update X_2^{k+1} by applying Algorithm 1 (line 10). Finally, we obtain the desired coefficient matrices X_1 and X_2, and their corresponding non-zero column index sets I and J (lines 12-13).

When implementing Algorithm 2, we use a sparse representation of tensor and mapping technique for skipping all the time-consuming and memory-expensive operations. The details are shown in Fig. 1, where we describe the technique in five steps. Specifically, we compute a number of index tables for saving computer memory resources, define an operator $\{\}$ for quickly indexing the value given the index tables, and further define a mapping operator $\mapsto$ for mapping the elements between different matrices or tensors. Indeed, we only map the non-zero elements of the system dynamics tensor to some matrices, which means that we care more about the occurred sequences in the agents' past experiences. We explain each step of Fig. 1 as follows:

Step 1: Store all the non-zero elements of system dynamics tensor $\mathcal{D}$ together to form a sparse table with two columns: one for the value and the other for the indexes;

```
Data: A tensor $\mathcal{D}$, positive scalars $\alpha_1$, $\alpha_2$ and $\mu$.
Result: An optimal solution $(X_1, X_2)$ and corresponding nonzero column index sets $I$ and
        $J$.
1  Initialize $X_2^0$;
2  Generate sparse tables I, $\mathrm{I}_1$ and $\mathrm{I}_2$;
3  Compute matrices $D_{(1)}$ and $D_{(2)}$: $\mathcal{D}\{\mathrm{I}\} \mapsto D_{(n)}\{\mathrm{I}_n\}$;
4  while the stopping criterion is not satisfied do
5      Compute matrix $A_{(2)}^k \leftarrow X_2^k D_{(2)}$;
6      Compute matrix $A_{(1)}^k$: $A_{(2)}^k\{\mathrm{I}_2\} \mapsto A_{(1)}^k\{\mathrm{I}_1\}$ ;
7      $X_1^{k+1} \leftarrow$ ADMM($A_{(1)}^k$,$D_{(1)}$,$\alpha_1, \mu, \epsilon$);
8      Compute matrix $B_{(1)}^{k+1} \leftarrow X_1^{k+1} A_{(1)}^k$;
9      Compute matrix $B_{(2)}^{k+1}$: $B_{(1)}^{k+1}\{\mathrm{I}_1\} \mapsto B_{(2)}^{k+1}\{\mathrm{I}_2\}$ ;
10     $X_2^{k+1} \leftarrow$ ADMM($B_{(2)}^{k+1}$,$D_{(2)}$, $\alpha_2, \mu, \epsilon$);
11 end
12 $X_1 \leftarrow X_1^{k+1}$ and $X_2 \leftarrow X_2^{k+1}$;
13 $I \leftarrow \{i \,|\, \mathrm{sgn}(\|(X_1)_{:i}\|_2) \neq 0\}$ and $J \leftarrow \{j \,|\, \mathrm{sgn}(\|(X_2)_{:j}\|_2) \neq 0\}$.
```

Algorithm 2: Alternating optimization for solving problem (12)

Step 2: Generate four index tables for matricization, i.e., index tables $\dot{\mathrm{I}}_1$, $\dot{\mathrm{I}}_2$, I_1 and I_2.

The former two tables are computed from the index table I using Eq. (1). The latter two tables recode column indices of index tables $\dot{\mathrm{I}}_1$ and $\dot{\mathrm{I}}_2$ by dropping all the zero columns in the corresponding matrices, thus the resulting column dimensions satisfying $\overline{n_2 n_3} \ll n_2 n_3$ and $\overline{n_1 n_3} \ll n_1 n_3$;

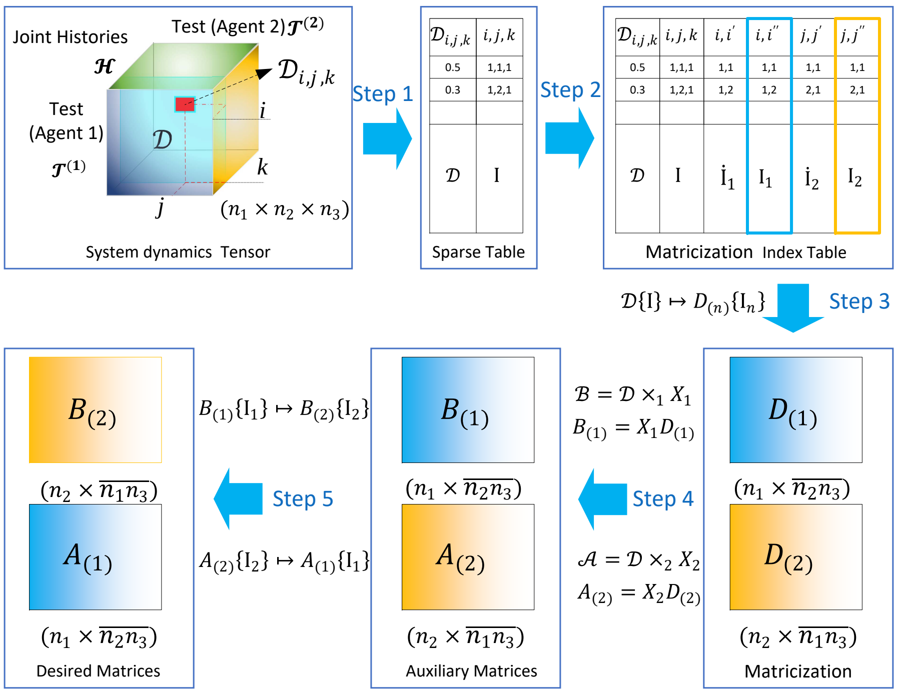
Step 3: Create the reduced mode-i unfolding matrix $D_{(i)}$ using index table I_i $(i = 1, 2)$ by mapping the non-zero elements (indexed by index table I) in tensor $\mathcal{D}$ to their positions (indexed by index table I_i) in matrix $D_{(i)}$, i.e. $\mathcal{D}\{\mathrm{I}\} \mapsto D_{(i)}\{\mathrm{I}_i\}$;

Step 4: Compute the auxiliary matrices in Algorithm 2 using these two index tables I_1 and I_2, and Eq. (2);

Step 5: Rearrange the elements of auxiliary matrices into the desired arrangement via the mapping operator without the help of auxiliary tensors, i.e. $A_{(2)}\{\mathrm{I}_2\} \mapsto A_{(1)}\{\mathrm{I}_1\}$ and $B_{(1)}\{\mathrm{I}_1\} \mapsto B_{(2)}\{\mathrm{I}_2\}$.

Actually we compute the mode-1 unfolding of tensor $\mathcal{D} \times_2 X_2$ and the mode-2 unfolding of tensor $\mathcal{D} \times_1 X_1$, i.e., matrices $A_{(1)}$ and $B_{(2)}$ respectively, in steps 4 and 5. We optimize the computations of these two matrices to save memory and time, since they are updated frequently in Algorithm 2 (lines 5-6 and lines 8-9, respectively). When Algorithm 2 stops, we shall obtain the sparse coefficient matrices X_1 and X_2 for problem (12) and their corresponding non-zero column index sets, namely

$$\begin{cases} I = \{i \mid \mathrm{sgn}(\|(X_1)_{:i}\|_2) \neq 0,\ 1 \leq i \leq n_1\}, \\ J = \{j \mid \mathrm{sgn}(\|(X_2)_{:j}\|_2) \neq 0,\ 1 \leq j \leq n_2\}. \end{cases}$$


Joint Histories
$\mathcal{H}$
Test (Agent 2) $\mathcal{T}^{(2)}$
$\mathcal{D}_{i,j,k}$
Test (Agent 1)
$\mathcal{T}^{(1)}$
$\mathcal{D}$
i
j
k
$(n_1 \times n_2 \times n_3)$
System dynamics Tensor
Step 1
Sparse Table
Step 2
Matricization Index Table
$\mathcal{D}\{\mathrm{I}\} \mapsto D_{(n)}\{\mathrm{I}_n\}$
Step 3
$\mathcal{B} = \mathcal{D} \times_1 X_1$
$B_{(1)} = X_1 D_{(1)}$
Step 4
$\mathcal{A} = \mathcal{D} \times_2 X_2$
$A_{(2)} = X_2 D_{(2)}$
$B_{(1)}\{\mathrm{I}_1\} \mapsto B_{(2)}\{\mathrm{I}_2\}$
Step 5
$A_{(2)}\{\mathrm{I}_2\} \mapsto A_{(1)}\{\mathrm{I}_1\}$
$D_{(1)}$ $(n_1 \times \overline{n_2 n_3})$
$D_{(2)}$ $(n_2 \times \overline{n_1 n_3})$
Matricization
$B_{(1)}$ $(n_1 \times \overline{n_2 n_3})$
$A_{(2)}$ $(n_2 \times \overline{n_1 n_3})$
Auxiliary Matrices
$B_{(2)}$ $(n_2 \times \overline{n_1 n_3})$
$A_{(1)}$ $(n_1 \times \overline{n_2 n_3})$
Desired Matrices

$\mathcal{D}_{i,j,k}$	i,j,k
0.5	1,1,1
0.3	1,2,1
$\mathcal{D}$	I

$\mathcal{D}_{i,j,k}$	i,j,k	i,i'	i,i''	j,j'	j,j''
0.5	1,1,1	1,1	1,1	1,1	1,1
0.3	1,2,1	1,2	1,2	2,1	2,1
$\mathcal{D}$	I	$\dot{\mathrm{I}}_1$	I_1	$\dot{\mathrm{I}}_2$	I_2

Fig. 1. A sparse representation of tensor and mapping operations when implementing Algorithm 2

Here, we assume that the size of the index sets are $|I| = m_1$ and $|J| = m_2$ where $m_1 < n_1$ and $m_2 < n_2$.

To analyze the computational complexity of Algorithm 2, we first investigate the ADMM steps. In Algorithm 1, the computational complexity mainly lies in updating the matrix X, which needs roughly $\mathbf{O}(n^3)$ operations (n is the row dimension of X). Hence, Algorithm 1 requires $\mathbf{O}(n^3/\epsilon)$ operations in order to achieve an ϵ-accuracy solution. Therefore, by introducing mapping technique, Algorithm 2 totally has the computational complexity $\mathbf{O}(n_1 m_1 \overline{n_2 n_3} + n_2 m_2 \overline{n_1 n_3} + n_1^3/\epsilon + n_2^3/\epsilon)$, where n_i is each dimension of system dynamics tensor $\mathcal{D}$, $\overline{n_2 n_3}$ is the column dimension of mode-1 unfolding $D_{(1)}$, $\overline{n_1 n_3}$ is the column dimension of mode-2 unfolding $D_{(2)}$, $m_1 = |I|$ and $m_2 = |J|$.

4.1.3. Extract a Core Joint Test Set

From the solution discussed in Section 4.1.2, we can extract all nonzero columns from matrix X_1 according to the index set I and construct a new matrix, denoted by $\bar{X}_1 \in \mathbb{R}^{n_1 \times m_1}$, whose column indices are renumbered as $1, 2, \ldots, m_1$. Similarly, we obtain the other new matrix $\bar{X}_2 \in \mathbb{R}^{n_2 \times m_2}$ by selecting all nonzero columns of matrix X_2 according to the index set J, and the column indices of $\bar{X}_2$ are renumbered as $1, 2, \ldots, m_2$. The two new matrices are called *reduced coefficient matrices*. The entry $\mathcal{D}_{ijk}$ of the tensor $\mathcal{D}$ can now be expressed by

$$\begin{aligned}\mathcal{D}_{ijk} &= \sum_{u=1}^{n_1}\sum_{v=1}^{n_2} \mathcal{D}_{uvk}(X_1)_{iu}(X_2)_{jv} \\ &= \sum_{u\in I}\sum_{v\in J} \mathcal{D}_{uvk}(X_1)_{iu}(X_2)_{jv} \\ &= \sum_{l=1}^{m_1}\sum_{w=1}^{m_2} \mathcal{Q}_{lwk}(\bar{X}_1)_{il}(\bar{X}_2)_{jw}, \end{aligned} \tag{19}$$

where $\mathcal{Q} \in \mathbb{R}^{m_1\times m_2\times n_3}$. The tensor $\mathcal{Q}$ is actually the core test tensor, whose elements can be completely extracted from the system dynamics tensor $\mathcal{D}$ according to the indices I and J, i.e., $\mathcal{Q} = \mathcal{D}_{IJ:}$. We don't need to specify the dimensions of the core test tensor in advance as it will be automatically determined after we solve the optimization problem (12). This differs from the previous PSR learning approaches.

We rewrite Eq. (19) in the tensor mode product form, i.e.,

$$\mathcal{D} = \mathcal{Q} \times_1 \bar{X}_1 \times_2 \bar{X}_2.$$

Fig. 2 shows the framework of the tensor optimization for obtaining the core test tensor. Accordingly, we construct the core joint test set $\mathbf{Q}$ from the joint test set $\mathcal{T}$ by using the

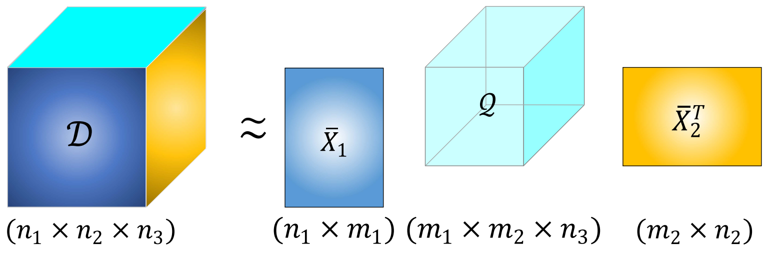


Fig. 2. A diagram of tensor optimization framework to obtain the core test tensor

nonzero column index sets I and J, i.e.,

$$\mathbf{Q} = \{\mathbf{t}_{i_1 i_2} \mid \mathbf{t}_{i_1 i_2} \in \mathcal{T}, i_1 \in I, i_2 \in J\},$$

which solves the discovery problem of a two-agent PSR model.

4.2. Learning Prediction Parameters and State Vectors

Let us denote $Q = Q_{(3)} \in \mathbb{R}^{n_3\times(m_1 m_2)}$ as the mode-3 unfolding of the core test tensor $\mathcal{Q}$ and $Q_{k:}$ to be its k-th row vector for $k = 1, 2, \ldots, n_3$. By the physical meaning of the core test tensor, we know that Q is the *state matrix* of the dynamic system and $Q_{k:}$ is the *state vector* at time step $s = |\mathbf{h}_k|$ with joint history $\mathbf{h}_k \in \mathcal{H}$, i.e., $Q_{k:} = p(\mathbf{Q}|\mathbf{h}_k)$. Moreover, we have from Eq. (19) that

$$\begin{aligned}\mathcal{D}_{ijk} &= \sum_{l=1}^{m_1}\sum_{w=1}^{m_2} \mathcal{Q}_{lwk}(\bar{X}_1)_{il}(\bar{X}_2)_{jw} \\ &= (\mathrm{vec}(\mathcal{Q}_{::k}))^T \ \left((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:}\right)^T \\ &= Q_{k:} \left((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:}\right)^T. \end{aligned}$$

From Eq. (4), we have $\mathcal{D}_{ijk} = p(t_i^{(1)} t_j^{(2)} | \mathbf{h}_k) = p(\mathbf{Q}|\mathbf{h}_k)\boldsymbol{m}_{t_i t_j}$, and so

$$\boldsymbol{m}_{\mathbf{t}} = \boldsymbol{m}_{t_i t_j} = \left((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:}\right)^T, \tag{20}$$

which is the prediction parameter that we need to learn.

Fig. 3 shows the concrete process of learning the prediction parameter $\boldsymbol{m}_{t_i t_j}$ and the state vector $Q_{k:}$ by the tensor optimization method. To be specific,

- **Step 1-3** describe how to find the state vector $Q_{k:}$ from the core test tensor $\mathcal{Q}$. We extract the k-th frontal slice matrix from the tensor $\mathcal{Q}$ in Step 1, and (column) vectorize and transpose it, i.e., $Q_{k:} = (\text{vec}(\mathcal{Q}_{::k}))^T$ in Steps 2-3.
- **Step 4-6** show the detailed process of obtaining the prediction parameter $\boldsymbol{m}_{t_i t_j}$. We obtain the two row vectors $(\bar{X}_1)_{i:}$ and $(\bar{X}_2)_{j:}$ of the corresponding reduced coefficient matrix in Steps 4 and 5, respectively, and calculate the prediction parameter $\boldsymbol{m}_{t_i t_j}$ by the Kronecker product of the two vectors in Step 6.

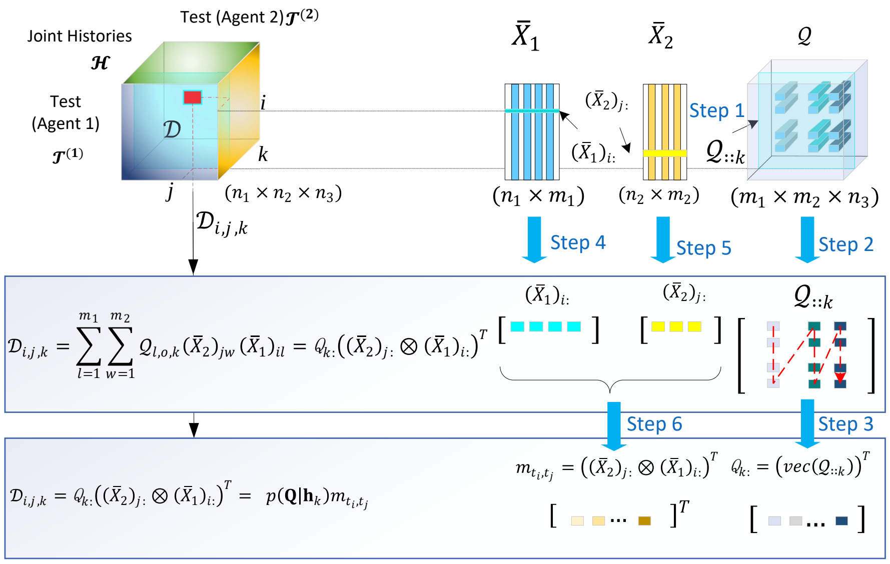

Test (Agent 2) $\mathcal{T}^{(2)}$
Joint Histories $\mathcal{H}$
Test (Agent 1) $\mathcal{T}^{(1)}$
$\mathcal{D}$
i
k
j
$(n_1 \times n_2 \times n_3)$
$\bar{X}_1$
$\bar{X}_2$
$\mathcal{Q}$
$(\bar{X}_2)_{j:}$
$(\bar{X}_1)_{i:}$
Step 1
$\mathcal{Q}_{::k}$
$(n_1 \times m_1)$
$(n_2 \times m_2)$
$(m_1 \times m_2 \times n_3)$
$\mathcal{D}_{i,j,k}$
Step 4
Step 5
Step 2
$\mathcal{D}_{i,j,k} = \sum_{l=1}^{m_1}\sum_{w=1}^{m_2} \mathcal{Q}_{l,o,k}(\bar{X}_2)_{jw}(\bar{X}_1)_{il} = \mathcal{Q}_{k:}((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:})^T$
$(\bar{X}_1)_{i:}$
$(\bar{X}_2)_{j:}$
$\mathcal{Q}_{::k}$
Step 6
Step 3
$m_{t_i,t_j} = ((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:})^T$
$\mathcal{Q}_{k:} = (vec(\mathcal{Q}_{::k}))^T$
$\mathcal{D}_{i,j,k} = \mathcal{Q}_{k:}((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:})^T = p(\mathbf{Q}|\mathbf{h}_k)m_{t_i,t_j}$

Fig. 3. Flowchart of learning the prediction parameter and the state vector

4.3. Learning Transition Parameters

From the analysis in Section 3 and 4.2, we can find one-step projection vectors $\{\boldsymbol{m}_{\mathbf{ao}}\}$ from the prediction parameter set $\{\boldsymbol{m}_{t_i t_j}\}$ for each $\mathbf{ao} \in \mathcal{A} \times \mathcal{O}$. The projection vectors serve as a key ingredient for learning the model transition parameters $\{M_{\mathbf{ao}}\}$. There are two methods to compute them.

One method is to directly compute the updated state vector $\boldsymbol{m}_{\mathbf{aoq}_i}$ for each one-step extension $(\mathbf{aoq}_i)$ for all $\mathbf{a} \in \mathcal{A}, \mathbf{o} \in \mathcal{O}, \mathbf{q}_i \in \mathbf{Q}$. We first extract a subtensor $\mathcal{Q}_{\mathbf{ao}}$ from the original tensor $\mathcal{D}$ according to the following index sets

$$\bar{I} = \{i \,|\, i = \mathcal{I}(\mathbf{ao}q_{i'}^{(1)} q_j^{(2)}, \mathcal{T}^{(1)}), i' \in I, j \in J\},$$
$$\bar{J} = \{j \,|\, j = \mathcal{I}(\mathbf{ao}q_i^{(1)} q_{j'}^{(2)}, \mathcal{T}^{(2)}), i \in I, j' \in J\},$$

i.e., $\mathcal{Q}_{\mathbf{ao}} = \mathcal{D}_{\bar{I}\bar{J}:}$, and then construct a matrix $Q_{\mathbf{ao}}$ from the mode-3 unfolding of the tensor $\mathcal{Q}_{\mathbf{ao}}$. According to Eq. (8) and $\mathbf{H} \subseteq \mathcal{H}$, we compute

$$\boldsymbol{m}_{\mathbf{aoq}_i} = p(\mathbf{Q}|\mathcal{H})^{-1} p(\mathbf{aoq}_i|\mathcal{H}) \tag{21}$$

where $p(\mathbf{Q}|\mathcal{H})$ is the system state matrix Q and $p(\mathbf{aoq}_i|\mathcal{H})$ is the i-th column vector of matrix $Q_{\mathbf{ao}}$. Hence, from Eq. (6), we obtain the transition matrix

$$M_{\mathbf{ao}} = [\boldsymbol{m}_{\mathbf{aoq}_1} \quad \boldsymbol{m}_{\mathbf{aoq}_2} \quad \dots \quad \boldsymbol{m}_{\mathbf{aoq}_{m_1 m_2}}].$$

As this method needs to construct a subtensor $\mathcal{Q}_{\mathbf{ao}}$ in order to obtain $p(\mathbf{aoq}_i|\mathcal{H})$, it consumes a large amount of memory and computational resources. Thus, it is not applicable to a large size.

To find an alternative way to obtaining the transition matrix $M_{\mathbf{ao}}$, we observe that $p(\mathbf{aoq}_i|\mathbf{h}_k)$ is the element of the vector $p(\mathbf{aoq}_i|\mathcal{H})$ for all $\mathbf{h}_k \in \mathcal{H}$ and $\mathbf{ao} \in \mathcal{A} \times \mathcal{O}$. According to the probability chain rule, we have at any time step $s = |\mathbf{h}_k|$,

$$p(\mathbf{aoq}_i|\mathbf{h}_s = \mathbf{h}_k) = p(\mathbf{ao}|\mathbf{h}_s = \mathbf{h}_k) \, * \, p(\mathbf{q}_i|\mathbf{h}_{s+1} = \mathbf{h}_k\mathbf{ao}).$$

Therefore,

$$p(\mathbf{aoq}_i|\mathcal{H}') = p(\mathbf{ao}|\mathcal{H}') \, * \, p(\mathbf{q}_i|\mathcal{H}'_{\mathbf{ao}}), \tag{22}$$

where $\mathcal{H}'$ and $\mathcal{H}'_{\mathbf{ao}}$ are the subsets of the joint history set $\mathcal{H}$. Specifically,

$$\mathcal{H}'_{\mathbf{ao}} = \{\mathbf{h} \,|\, \mathbf{h} = \mathbf{h}'\mathbf{ao}, \mathbf{h} \in \mathcal{H}, \mathbf{h}' \in \mathcal{H}\} \text{ and } \mathcal{H}' = \{\mathbf{h} \,|\, \mathbf{h} = \mathbf{h}' \setminus \mathbf{ao}, \mathbf{h} \in \mathcal{H}, \mathbf{h}' \in \mathcal{H}'_{\mathbf{ao}}\}.$$

The former set consists of all the joint histories $\mathbf{h} \in \mathcal{H}$ ending with action-observation $\mathbf{ao}$ and the latter one is obtained by truncating the terminal action-observation sequence $\mathbf{ao}$ for every joint history $\mathbf{h}' \in \mathcal{H}'_{\mathbf{ao}}$. We observe that $|\mathcal{H}'| = |\mathcal{H}'_{\mathbf{ao}}|$. Let $p(\mathbf{Q}|\mathcal{H}')$ be the matrix extracted and constructed from the system state matrix Q according to the row index set $\mathcal{I}(\mathcal{H}', \mathcal{H})$, i.e., $p(\mathbf{Q}|\mathcal{H}') \leftarrow Q_{\mathcal{I}(\mathcal{H}',\mathcal{H})}$. Similarly, we have $p(\mathbf{Q}|\mathcal{H}'_{\mathbf{ao}}) \leftarrow Q_{\mathcal{I}(\mathcal{H}'_{\mathbf{ao}},\mathcal{H})}$. For simplicity, we use $Q_{\mathcal{H}'}$ and $Q_{\mathcal{H}'_{\mathbf{ao}}}$ to denote $Q_{\mathcal{I}(\mathcal{H}',\mathcal{H})}$ and $Q_{\mathcal{I}(\mathcal{H}'_{\mathbf{ao}},\mathcal{H})}$, respectively. Therefore, we obtain

$$\begin{aligned} \boldsymbol{m}_{\mathbf{aoq}_i} &= p(\mathbf{Q}|\mathcal{H}')^{-1}\big(p(\mathbf{ao}|\mathcal{H}') \, * \, p(\mathbf{q}_i|\mathcal{H}'_{\mathbf{ao}})\big) \\ &= p(\mathbf{Q}|\mathcal{H}')^{-1}\Bigg(\big(p(\mathbf{Q}|\mathcal{H}')\boldsymbol{m}_{\mathbf{ao}}\big) \, * \, p(\mathbf{q}_i|\mathcal{H}'_{\mathbf{ao}})\Bigg), \end{aligned}$$

where the first equation is obtained from Eqs. (21) and (22), and the second one is obtained from Eq. (3). Moreover, from Eq. (6), we have

$$\begin{aligned} M_{\mathbf{ao}} &= p(\mathbf{Q}|\mathcal{H}')^{-1}\mathrm{diag}(p(\mathbf{Q}|\mathcal{H}')\boldsymbol{m}_{\mathbf{ao}})p(\mathbf{Q}|\mathcal{H}'_{\mathbf{ao}}) \\ &= Q_{\mathcal{H}'}^{-1}\mathrm{diag}(Q_{\mathcal{H}'}\boldsymbol{m}_{\mathbf{ao}})Q_{\mathcal{H}'_{\mathbf{ao}}}. \end{aligned} \tag{23}$$

If there are enough training data, we can get an precise estimation of $Q_{\mathcal{H}'}$, $Q_{\mathcal{H}'_{\mathbf{ao}}}$ and $\boldsymbol{m}_{\mathbf{ao}}$. The error of calculating $M_{\mathbf{ao}}$ will be sufficiently minimal, which implies that Eq. (23) is true in general. However, due to the limited training data, it is generally impossible to obtain an exact $M_{\mathbf{ao}}$ to maintain the equality in Eq. (23).

Let the vector $\boldsymbol{d}_{\mathbf{ao}} = Q_{\mathcal{H}'}\boldsymbol{m}_{\mathbf{ao}}$ whose elements are $d^1_{\mathbf{ao}}, \ldots, d^{|\mathcal{H}'|}_{\mathbf{ao}}$. We construct a diagonal matrix $D_{\mathbf{ao}} = \text{diag}(d^1_{\mathbf{ao}}, \ldots, d^{|\mathcal{H}'|}_{\mathbf{ao}})$ and transform Eq. (23) into the following optimization problem

$$\min_{M} \ \frac{1}{2}\|D_{\mathbf{ao}}Q_{\mathcal{H}'_{\mathbf{ao}}} - Q_{\mathcal{H}'}M\|_F^2,$$

whose corresponding optimal solution is

$$M^*_{\mathbf{ao}} = (Q^T_{\mathcal{H}'}Q_{\mathcal{H}'})^{-1}(Q^T_{\mathcal{H}'}D_{\mathbf{ao}}Q_{\mathcal{H}'_{\mathbf{ao}}}).$$

To conclude this section, we provide the overall process of discovering the core joint test set and learning the model parameters of a two-agent PSR model; see Fig. 4 and Algorithm 3. Fig. 4 consists of 8 steps and the pseudo code of each step is shown in Algorithm 3. We explain each of the 8 steps as follows:

Step 1: Establish the system dynamics tensor $\mathcal{D}$ from the interaction data Φ_d (line 1);

Step 2: Construct a tensor optimization problem and solve it by Algorithm 2 (line 2);

Step 3: Obtain the sparse coefficient matrices X_1 and X_2 and their corresponding indices (line 2);

Step 4: Construct the reduced coefficient matrices $\bar{X}_1$ and $\bar{X}_2$, the core test set $\mathbf{Q}$, and the core test tensor $\mathcal{Q}$ (lines 3-5);

Step 5: Construct the system state matrix Q and calculate the projection vector $\boldsymbol{m}_{\mathbf{t}}$ (lines 6-11);

Step 6: Obtain the PSR model prediction parameter $\boldsymbol{m}_{\mathbf{ao}}$ from $\boldsymbol{m}_{\mathbf{t}}$ and construct two joint history sets $\mathcal{H}'_{\mathbf{ao}}$ and $\mathcal{H}'$ and matrices $Q_{\mathcal{H}'_{\mathbf{ao}}}$ and $Q_{\mathcal{H}'}$ (lines 13-18);

Step 7: Compute the vector $\boldsymbol{d}_{\mathbf{ao}}$ and the matrix $D_{\mathbf{ao}}$ (lines 19-20);

Step 8: Compute the model parameter $M_{\mathbf{ao}}$ (line 21).

5. Extension to Learning a (N >2)-Agent PSR Model

We proceed to extend the single-agent PSR learning to a multi-agent case. As the learning procedures are quite similar, we briefly discuss the core test discovery and model parameter learning in this section.

```
Data: Interaction data $\Phi_d$ of the two agents.
Result: Core joint test set $\mathbf{Q}$, state matrix $p(\mathbf{Q}|\mathcal{H})$, and model parameters $\{\boldsymbol{m}_{\mathbf{ao}}\}$ and
        $\{M_{\mathbf{ao}}\}$.
1  Construct system dynamics tensor $\mathcal{D}$ from data $\Phi_d$;
2  Run Algorithm 2 to obtain $[X_1, I]$ and $[X_2, J]$;
3  $\bar{X}_1 \leftarrow (X_1)_{:I}$ and $\bar{X}_2 \leftarrow (X_2)_{:J}$;
4  $\mathcal{Q} \leftarrow \mathcal{D}_{IJ:}$;
5  $\mathbf{Q} \leftarrow \{\mathbf{t}_{i_1 i_2} | \mathbf{t}_{i_1 i_2} \in \mathcal{T}, i_1 \in I, i_2 \in J\}$;
6  $Q \leftarrow \mathcal{Q}_{(3)}$;
7  $p(\mathbf{Q}|\mathcal{H}) \leftarrow Q$;
8  foreach $i \in \{1, 2, \dots n_1\}, j \in \{1, 2, \dots n_2\}, \boldsymbol{t} \in \mathcal{T}$ do
9      $\boldsymbol{m}_{t_i t_j} \leftarrow \big((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:}\big)^T$;
10     $\boldsymbol{m}_{\mathbf{t}} \leftarrow \boldsymbol{m}_{t_i t_j}$;
11 end
12 foreach $\boldsymbol{a} \in \mathcal{A}, \boldsymbol{o} \in \mathcal{O}$ do
13     $\mathbf{t} \leftarrow \mathbf{ao}$;
14     $\boldsymbol{m}_{\mathbf{ao}} \leftarrow \boldsymbol{m}_{\mathbf{t}}$;
15     $\mathcal{H}'_{\mathbf{ao}} \leftarrow \{\mathbf{h} \,|\, \mathbf{h} = \mathbf{h}'\mathbf{ao}, \mathbf{h} \in \mathcal{H}, \mathbf{h}' \in \mathcal{H}\}$;
16     $\mathcal{H}' \leftarrow \{\mathbf{h} \,|\, \mathbf{h} = \mathbf{h}' \setminus \mathbf{ao}, \mathbf{h} \in \mathcal{H}, \mathbf{h}' \in \mathcal{H}'_{\mathbf{ao}}\}$;
17     $Q_{\mathcal{H}'_{\mathbf{ao}}} \leftarrow Q_{\mathcal{I}(\mathcal{H}'_{\mathbf{ao}}, \mathcal{H})}$;
18     $Q_{\mathcal{H}'} \leftarrow Q_{\mathcal{I}(\mathcal{H}', \mathcal{H})}$;
19     $\boldsymbol{d}_{\mathbf{ao}} \leftarrow Q_{\mathcal{H}'} \boldsymbol{m}_{\mathbf{ao}}$;
20     $D_{\mathbf{ao}} \leftarrow \mathrm{diag}(d^1_{\mathbf{ao}}, \dots, d^{|\mathcal{H}'|}_{\mathbf{ao}})$;
21     $M_{\mathbf{ao}} \leftarrow (Q^T_{\mathcal{H}'} Q_{\mathcal{H}'})^{-1} (Q^T_{\mathcal{H}'} D_{\mathbf{ao}} Q_{\mathcal{H}'_{\mathbf{ao}}})$ ;
22 end
```

Algorithm 3: Tensor-based ADMM for learning two-agent PSR model

5.1. *Tensor Optimization for Discovering a Core Joint Test Set*

For a dynamic system with N agents, suppose that the system dynamics tensor is $\mathcal{D} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times \cdots \times n_{N+1}}$ with the element $\mathcal{D}_{i_1 i_2 \dots i_N k} = p(t^{(1)}_{i_1} \dots t^{(N)}_{i_N} | \mathbf{h}_k)$. Analogous to Section 4.1, we seek to find a number of sparse coefficient matrices $X_1, X_2, \cdots, X_N$ such that

$$\mathcal{D} = \mathcal{D} \times_1 X_1 \times_2 X_2 \times_3 \cdots \times_N X_N. \tag{24}$$

In practice, we relax Eq. (24) and search for a solution such that the gap $\|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2 \cdots \times_N X_N\|$ is minimized. Thus, we formulate it as a group LASSO problem by using the $L_{2,1}$-regularization, i.e.,

$$\min_{X_1, X_2, \dots, X_N} \frac{1}{2} \|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2 \times_3 \cdots \times_N X_N\|_F^2 + \alpha_1 \|X_1\|_{2,1} + \alpha_2 \|X_2\|_{2,1} + \cdots + \alpha_N \|X_N\|_{2,1}.$$

This is an unconstrained optimization problem and can be solved by alternately minimizing, i.e., optimize one X_i while other $N-1$ X_i's are fixed until the procedures converge.

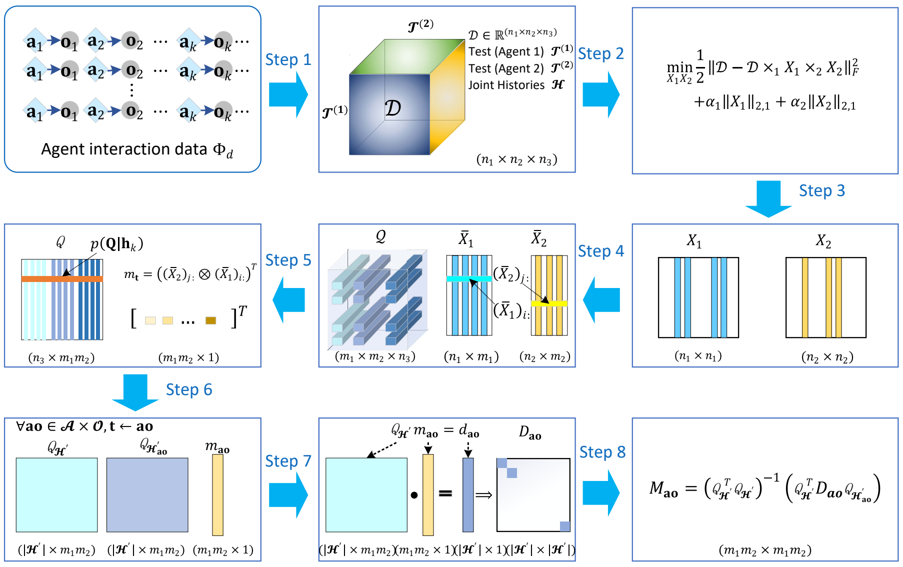

Agent interaction data Φ_d
Step 1
$\mathcal{D} \in \mathbb{R}^{(n_1 \times n_2 \times n_3)}$
Test (Agent 1) $\mathcal{T}^{(1)}$
Test (Agent 2) $\mathcal{T}^{(2)}$
Joint Histories $\mathcal{H}$
$(n_1 \times n_2 \times n_3)$
Step 2
$\min_{X_1 X_2} \frac{1}{2}\|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2\|_F^2 + \alpha_1\|X_1\|_{2,1} + \alpha_2\|X_2\|_{2,1}$
Step 3
$(n_1 \times n_1)$
$(n_2 \times n_2)$
Step 4
$(m_1 \times m_2 \times n_3)$
$(n_1 \times m_1)$
$(n_2 \times m_2)$
Step 5
$p(\mathbf{Q}|\mathbf{h}_k)$
$m_t = ((\bar{X}_2)_{j:} \otimes (\bar{X}_1)_{i:})^T$
$(n_3 \times m_1 m_2)$
$(m_1 m_2 \times 1)$
Step 6
$\forall \mathbf{ao} \in \mathcal{A} \times \mathcal{O}, \mathbf{t} \leftarrow \mathbf{ao}$
Step 7
$\mathcal{Q}_{\mathcal{H}'} m_{\mathbf{ao}} = d_{\mathbf{ao}}$
$D_{\mathbf{ao}}$
Step 8
$M_{\mathbf{ao}} = (\mathcal{Q}_{\mathcal{H}'}^T \mathcal{Q}_{\mathcal{H}'})^{-1} (\mathcal{Q}_{\mathcal{H}'}^T D_{\mathbf{ao}} \mathcal{Q}_{\mathcal{H}'_{\mathbf{ao}}})$
$(m_1 m_2 \times m_1 m_2)$

Fig. 4. An overall framework of learning a two-agent PSR model

Without loss of generality, let us consider the case of optimizing the matrix X_1, i.e., we fix variables $X_2, X_3, \cdots, X_N$ and optimize the following problem

$$\min_{X_1} \quad \frac{1}{2}\|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2 \times \cdots \times_N X_N\|_F^2 + \alpha_1 \|X_1\|_{2,1} . \tag{25}$$

Let $\mathcal{A} = \mathcal{D} \times_2 X_2 \times_3 \cdots \times_N X_N$ and (25) is then equivalent to

$$\min_{X_1} \quad \frac{1}{2}\|X_1 A_{(1)} - D_{(1)}\|_F^2 + \alpha_1 \|X_1\|_{2,1} .$$

We find that all the N subproblems again have the same unified form as the optimization problem (15), which can be efficiently solved by Algorithm 1.

When the solution coefficient matrices $X_1, X_2, \cdots, X_N$ are available, we extract reduced coefficient matrix $\bar{X}_i$ from the matrix X_i according to its nonzero column index set I_i where $|I_i| = m_i$ for each $i \in \{1, 2, \ldots, N\}$. Hence, the dimension of each $\bar{X}_i$ is $n_i \times m_i$. Each element of tensor $\mathcal{D}$ can be written as

$$\begin{aligned} \mathcal{D}_{i_1 i_2 \ldots i_N k} &= \sum_{i'_1 \in I_1} \sum_{i'_2 \in I_2} \cdots \sum_{i'_N \in I_N} \mathcal{D}_{i'_1 i'_2 \ldots i'_N k} (X_1)_{i_1 i'_1} (X_2)_{i_2 i'_2} \ldots (X_N)_{i_N i'_N} \\ &= \sum_{i'_1=1}^{m_1} \sum_{i'_2=1}^{m_2} \cdots \sum_{i'_N=1}^{m_N} \mathcal{Q}_{i'_1 i'_2 \ldots i'_N k} (\bar{X}_1)_{i_1 i'_1} (\bar{X}_2)_{i_2 i'_2} \ldots (\bar{X}_N)_{i_N i'_N} . \end{aligned} \tag{26}$$

The above can also be written as the tensor mode product form

$$\mathcal{D} = \mathcal{Q} \times_1 \bar{X}_1 \times_2 \bar{X}_2 \cdots \times_N \bar{X}_N ,$$

where the core tensor $\mathcal{Q} \in \mathbb{R}^{m_1 \times m_2 \times \cdots \times m_N \times n_{N+1}}$ is extracted from the tensor $\mathcal{D}$ according to the nonzero column index sets $I_1, I_2, \ldots, I_N$, i.e., $\mathcal{Q} = \mathcal{D}_{I_1 I_2 \ldots I_N :}$.

The core joint test set $\mathbf{Q}$ can be obtained correspondingly, i.e.,

$$\mathbf{Q} = \{\mathbf{t}_{i_1 i_2 \ldots i_N} \,|\, \mathbf{t}_{i_1 i_2 \ldots i_N} \in \mathcal{T}, i_1 \in I_1, i_2 \in I_2, \ldots, i_N \in I_N\}.$$

This solves the discovery problem of a multi-agent PSR model.

5.2. Learning Model Parameters

In this part, we discuss how to learn state vectors, prediction parameters, and further transition parameters of a multi-agent PSR model.

Let $Q = Q_{(N+1)} \in \mathbb{R}^{n_{N+1} \times (m_1 m_2 \ldots m_N)}$ be the mode-$(N+1)$ unfolding of the core test tensor $\mathcal{Q}$. Then, each row vector of the the state matrix Q is the state vector at the time step $s = |\mathbf{h}_k|$ with the joint history $\mathbf{h}_k \in \mathcal{H}$, i.e., $Q_{k:} = p(\mathbf{Q}|\mathbf{h}_k)$. On the other side, from Eq. (19), we derive that

$$\begin{aligned}\mathcal{D}_{i_1 i_2 \ldots i_N k} &= (\mathrm{vec}(\mathcal{Q}_{:\cdots:k}))^T \left((\bar{X}_N)_{i_N:} \otimes \cdots \otimes (\bar{X}_2)_{i_2:} \otimes (\bar{X}_1)_{i_1:}\right)^T \\ &= Q_{k:} \left((\bar{X}_N)_{i_N:} \otimes \cdots \otimes (\bar{X}_2)_{i_2:} \otimes (\bar{X}_1)_{i_1:}\right)^T,\end{aligned}$$

indicating that the prediction parameter is

$$\boldsymbol{m}_{\mathbf{t}} = \left((\bar{X}_N)_{i_N:} \otimes \cdots \otimes (\bar{X}_2)_{i_2:} \otimes (\bar{X}_1)_{i_1:}\right)^T.$$

Next, we let $\mathbf{t} = \mathbf{ao}$ and we can find the one-step projection vector $\boldsymbol{m}_{\mathbf{ao}}$ from the prediction parameter for all $\mathbf{a} \in \mathcal{A}$ and $\mathbf{o} \in \mathcal{O}$. We are now ready to present the calculation of transition parameters. We construct three matrices $Q_{\mathcal{H}'}$, $Q_{\mathcal{H}'_{\mathbf{ao}}}$ and $D_{\mathbf{ao}}$ analogous to that in Section 4.3, establish an optimization problem, and then find an optimal solution M^* to the transition matrix $M_{\mathbf{ao}}$, i.e.,

$$M^* = (Q_{\mathcal{H}'}^T Q_{\mathcal{H}'})^{-1} (Q_{\mathcal{H}'}^T D_{\mathbf{ao}} Q_{\mathcal{H}'_{\mathbf{ao}}}).$$

The overall learning process for a multi-agent PSR model is summarized in Algorithm 4. We first construct the system dynamics tensor $\mathcal{D}$ from the agents interaction data Φ_d and and sparse tables I, I_1, I_2, ..., I_N (lines 1). Then, we can use alternating minimization strategy with a mapping technology to optimize one variable at each time while keeping others fixed until the procedures converge (lines 2-11) where we adopt Algorithm 1 to solve the embedding sub-problems. Therefore, we find the nonzero column index set I_i and reduce the coefficient matrix $\bar{X}_i$ for all $i \in \{1, 2, \ldots, N\}$ (lines 12-15). The next step is to find the core test set $\mathcal{Q}$ and the system state matrix Q (lines 16-19). Furthermore, we find the prediction parameters $\{\boldsymbol{m}_{\mathbf{ao}}\}$ from $\{\boldsymbol{m}_{\mathbf{t}}\}$ (lines 20-23). Finally, we calculate the transition parameters $\{M_{\mathbf{ao}}\}$ of the multi-agent PSR model (lines 23-34).

The discovery-based learning approaches using a combinatorial search for the set of core tests have the time complexity of $\mathbf{O}((|\mathcal{A}||\mathcal{O}|)^L)$ in the worst case, where L is the max-length of a joint action-observation sequence in a training dataset. This discovery-based time complexity can be reduced to $\mathbf{O}\left(|\mathcal{H}||\mathbf{Q}||\mathcal{T}|\right)$ when the minimal size of set of core tests is provided as an input. However, this assumption is not available in practice in most cases. In contrast, Algorithm 4 without mapping technique for the discovery problem has

```
Data: Interaction data $\Phi_d$ of $N$ agents, parameters $\alpha_i\,(i = 1, 2, \dots, N)$ and $\mu$.
Result: Core joint test set $\mathbf{Q}$, state matrix $p(\mathbf{Q}|\mathcal{H})$, and model parameters $\{\boldsymbol{m}_{\mathbf{ao}}\}$ and
        $\{M_{\mathbf{ao}}\}$.
1  Construct system dynamics tensor $\mathcal{D}$ and sparse tables I, I$_1$, I$_2$, ..., I$_N$ from data $\Phi_d$;
2  while the stopping criterion is not satisfied do
3      Initialize matrices $A^{(1)}_{(1)} \leftarrow D_{(1)}$;
4      $X_1 \leftarrow \mathrm{ADMM}(A^{(1)}_{(1)}, D_{(1)}, \alpha_1, \mu, \epsilon)$;
5      foreach $i \in \{2, \dots, N\}$ do
6          $A^{(i)}_{(i-1)} \leftarrow X_{(i-1)} A^{(i-1)}_{(i-1)}$;
7          $A^{(i)}_{(i-1)}\{\mathrm{I}_{i-1}\} \mapsto A^{(i)}_{(i)}\{\mathrm{I}_i\}$ ;
8          $X_i \leftarrow \mathrm{ADMM}(A^{(i)}_{(i)}, D_{(i)}, \alpha_i, \mu, \epsilon)$;
9          Update $X_i$;
10     end
11 end
12 foreach $i \in \{1, 2, \dots, N\}$ do
13     Find $I_i$;
14     $\bar{X}_i \leftarrow (X_i)_{:I_i}$;
15 end
16 $\mathcal{Q} \leftarrow \mathcal{D}_{I_1 I_2 \dots I_N :}$;
17 $\mathbf{Q} \leftarrow \{\mathbf{t}_{i_1 i_2 \dots i_N} | \mathbf{t}_{i_1 i_2 \dots i_N} \in \mathcal{T}, i_1 \in I_1, i_2 \in I_2, \dots, i_N \in I_N\}$;
18 $Q \leftarrow Q_{(N+1)}$;
19 $p(\mathbf{Q}|\mathcal{H}) \leftarrow Q$;
20 foreach $i_1 \in \{1, 2, \dots n_1\}, i_2 \in \{1, 2, \dots n_2\}, \dots, i_N \in \{1, 2, \dots n_N\}, \boldsymbol{t} \in \mathcal{T}$ do
21     $\boldsymbol{m}_{t_{i_1} t_{i_2} \dots t_{i_N}} \leftarrow \big((\bar{X}_N)_{i_N:} \otimes \cdots \otimes (\bar{X}_2)_{i_2:} \otimes (\bar{X}_1)_{i_1:}\big)^T$;
22     $\boldsymbol{m}_{\mathbf{t}} \leftarrow \boldsymbol{m}_{t_{i_1} t_{i_2} \dots t_{i_N}}$;
23 end
24 foreach $\boldsymbol{a} \in \mathcal{A}, \boldsymbol{o} \in \mathcal{O}$ do
25     $\mathbf{t} \leftarrow \mathbf{ao}$;
26     $\boldsymbol{m}_{\mathbf{ao}} \leftarrow \boldsymbol{m}_{\mathbf{t}}$;
27     $\mathcal{H}'_{\mathbf{ao}} \leftarrow \{\mathbf{h} | \mathbf{h} = \mathbf{h}'\mathbf{ao}, \mathbf{h} \in \mathcal{H}, \mathbf{h}' \in \mathcal{H}\}$;
28     $\mathcal{H}' \leftarrow \{\mathbf{h} | \mathbf{h} = \mathbf{h}' \setminus \mathbf{ao}, \mathbf{h} \in \mathcal{H}, \mathbf{h}' \in \mathcal{H}'_{\mathbf{ao}}\}$;
29     $Q_{\mathcal{H}'_{\mathbf{ao}}} \leftarrow Q_{\mathcal{I}(\mathcal{H}'_{\mathbf{ao}}, \mathcal{H})}$;
30     $Q_{\mathcal{H}'} \leftarrow Q_{\mathcal{I}(\mathcal{H}', \mathcal{H})}$;
31     $\boldsymbol{d}_{\mathbf{ao}} \leftarrow Q_{\mathcal{H}'} \boldsymbol{m}_{\mathbf{ao}}$;
32     $D_{\mathbf{ao}} \leftarrow \mathrm{diag}(d^1_{\mathbf{ao}}, \dots, d^{|\mathcal{H}'|}_{\mathbf{ao}})$;
33     $M_{\mathbf{ao}} \leftarrow (Q^T_{\mathcal{H}'} Q_{\mathcal{H}'})^{-1} (Q^T_{\mathcal{H}'} D_{\mathbf{ao}} Q_{\mathcal{H}'_{\mathbf{ao}}})$ ;
34 end
```

Algorithm 4: Tensor-based ADMM for learning a multi-agent PSR model

the computational complexity of $\mathbf{O}(Nn^3/\epsilon + |\mathcal{H}|n^{2N-1})$, where $n = \max_{i=1}^{N}(|\mathcal{T}^{(i)}|)$. By introducing mapping technique, the computational complexity of Algorithm 4 for the discovery problem (lines 1-19) can be further reduced to $\mathbf{O}(Nn^3/\epsilon + \overline{n}mn)$, where $m = \max_{i=1}^{N}(|I_i|)$,

Table 3. Basic information of the two-agent domains

Domain	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{S}\rvert$	Relationship
Tag	5×5	$2^4 \times 2^4$	870	Competitive
Gridworld$*$	4×4	$2^4 \times 2^4$	2704	Competitive
ColoredGridworld$*$	4×4	$2^8 \times 2^8$	2704	Competitive
Poc-Man$*$	4×4	$\approx 2^9 \times 2^9$	$\approx 10^{56}$	Cooperative

$\overline{n} = \max_{i=1}^{N}(\overline{\prod_{j=1,j\neq i}^{N} n_j})$. Moreover, the time complexity of our algorithm for the learning problem (lines 20-34) is $\mathbf{O}(|\mathcal{A}||\mathcal{O}|d_{\max}^3)$, where $d_{\max} = \max(|\mathbf{Q}|, |\mathcal{H}'|)$. In summary, Algorithm 4 with mapping technique for learning a multi-agent PSR model has the computational complexity of $\mathbf{O}(Nn^3/\epsilon + \overline{n}mn + |\mathcal{A}||\mathcal{O}|d_{\max}^3)$.

6. Experimental Study

We will test the learning algorithms for a multi-agent PSR model (i.e., Algorithms 3 and 4) over four different extended versions of benchmarks that are commonly used for the PSR research, i.e., Tag [22], Gridworld$*$ [9], ColoredGridworld$*$ [8] and Poc-Man$*$ [26]. We present basic information of a two-agent setting, including the size of action space $\mathcal{A}$, observation space $\mathcal{O}$, system state space $\mathcal{S}$ and the relationship between the two agents in Table 3. The detailed descriptions of the four domains can also be seen in the reference [5]. In addition, we simply add one more agent in domains Gridworld$*$ and Tag to formulate a three-agent system for testing our algorithm.

We adapt traditional learning PSR methods that were developed for a single-agent case (i.e., transformed PSR (TPSR) and compressed PSR (CPSR) approaches [9, 8]) and empirically compare them with our algorithms in the aforementioned domains. To conduct a fair comparison, we pre-set the compressed dimension of our algorithms equal to that of TPSR and CPSR algorithms. All the prediction models are implemented in the integrated development environment of MATLAB conducted on a Windows PC with a 4-core Intel Core i7-8550U 1.99 GHz CPU and 16 GB RAM.

For evaluating the learnt PSR models, a series of action-observation sequences with length L are generated in every domain d, which are collected during agents applying random exploration strategy to continuously interact with the environment. Then, we split the sequences into two datasets, one is training dataset Φ_d and the other is test dataset Ψ_d. The training dataset Φ_d contains 2000 sequences of action-observation with a maximum length $L = 10$. While in the test dataset Ψ_d, there are 3000 sequences of action-observation with a maximum length $L = 15$. For each domain, $R_{num} = 20$ rounds are conducted to evaluate the average performance of each model. In each round r, a subset Φ_d^{r} is randomly sampled from the training dataset Φ_d with size $|\Phi_d^{\mathrm{r}}| = 500$, and a subset Ψ_d^{r} is randomly sampled from the test dataset Ψ_d with size $|\Psi_d^{\mathrm{r}}| = 1000$ for all the domains.

As we can see in Table 4, a subset Φ_d^{r} is randomly sampled from the training dataset Φ_d with size $|\Phi_d^{\mathrm{r}}| = 500$ for all domains and the joint test set is generated according to a rule, which specifies the expected maximum length of the joint tests in the joint test set ($l = max(\{|\mathbf{t}|\,|\,\forall\,\mathbf{t} \in \mathcal{T}\})$). When we let the value l increase from 1 to 10 ($L = 10$), the

Table 4. Examples of system dynamics tensor

Domain	N	l	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{D}\rvert$	Required RAM(GB)
		1	1367	$73 \times 74 \times 4721$	< 16
		2	5788	$1225 \times 1294 \times 4721$	55.8
	2	3	9776	$4199 \times 4398 \times 4721$	649.6
		⋮	⋮	⋮	⋮
Tag		10	23707	$17981 \times 18227 \times 4721$	11528.0
		1	4259	$75 \times 77 \times 71 \times 4897$	≈ 16
		2	8855	$1092 \times 1222 \times 1203 \times 4897$	58642.4
	3	3	12851	$4109 \times 4493 \times 4535 \times 4897$	—
		⋮	⋮	⋮	⋮
		10	26826	$17531 \times 17961 \times 18125 \times 4897$	—
		1	1135	$62 \times 61 \times 4713$	< 16
		2	5564	$1108 \times 1136 \times 4713$	44.2
	2	3	9564	$4293 \times 4290 \times 4713$	646.7
		⋮	⋮	⋮	⋮
GridWorld$*$		10	23589	$18271 \times 18195 \times 4713$	11710.7
		1	3816	$62 \times 58 \times 59 \times 4907$	< 16
		2	8525	$1152 \times 1132 \times 1015 \times 4907$	48391.7
	3	3	12525	$4417 \times 4332 \times 3811 \times 4907$	—
		⋮	⋮	⋮	⋮
		10	26525	$18343 \times 18251 \times 17543 \times 4907$	—
		1	1125	$74 \times 74 \times 4725$	< 16
		2	4906	$1232 \times 1214 \times 4725$	52.7
	2	3	8895	$3604 \times 3648 \times 4725$	462.8
CloloredGridWorld$*$		10	22894	$17023 \times 17145 \times 4725$	10274.6
		1	3337	$67 \times 62 \times 113 \times 4898$	17.1
		2	7272	$1200 \times 1201 \times 1141 \times 4898$	60009.3
	3	3	11272	$3557 \times 3604 \times 3422 \times 4898$	1600875.5
		⋮	⋮	⋮	⋮
		10	25272	$16980 \times 17084 \times 16779 \times 4898$	—
		1	3440	$76 \times 95 \times 4489$	< 16
		2	7011	$3669 \times 3280 \times 4489$	402.5
	2	3	10085	$6734 \times 6324 \times 4489$	1424.3
		⋮	⋮	⋮	⋮
Poc-Man$*$		10	19205	$16027 \times 13937 \times 4489$	7470.7
		1	4608	$108 \times 145 \times 117 \times 4930$	67.3
		2	9108	$2597 \times 2520 \times 2474 \times 4930$	594715.5
	3	3	13108	$6032 \times 5830 \times 5877 \times 4930$	—
		⋮	⋮	⋮	⋮
		10	27108	$19962 \times 19732 \times 19794 \times 4930$	—

constructed joint test set has an increasing number of joint tests and the size of the system dynamics tensor also increases. As a result, the RAM memory required for storing system dynamics tensor, modelling and solving the optimization becomes unbearable. Thanks to the probability chain rule, when obtaining a sufficiently long history, we can use a joint test set ($l = 1$) and a history set to construct the system dynamics tensor for modelling and prediction without losing any solution quality. Therefore, instead of creating a sparse and large tensor, wasting time to the never occurred sequences in the dataset and computing matrix unfoldings via auxiliary tensors and tensor mode product, we use a sparse representation of tensor and mapping technique for skipping all the time-consuming and memory-expensive operations in Algorithms 2, 3 and 4.

6.1. Performance Measurements

We compare the models in terms of prediction accuracy, which calculates the gap between the true predictions and the estimated ones given by the learnt PSR models over the test dataset Ψ^{r}_{d}. Therefore, we use the metric *absolute error* (AE) to measure this difference, which computes the average of absolute error of one-step prediction error per time step given an arbitrary history $\mathbf{h}_k \in \mathcal{H}_s$ at time step $s = |\mathbf{h}_k|$, defined by

$$AE = \frac{1}{N_T} \sum_{t=1}^{N_T} |\hat{p}(\mathbf{o}^t_{k+1}|\mathbf{h}^t_k \mathbf{a}^t_{k+1}) - p(\mathbf{o}^t_{k+1}|\mathbf{h}^t_k \mathbf{a}^t_{k+1})|. \tag{27}$$

In Eq. (27), N_T is the total number of test sequences ($N_T = |\Psi^{\text{test}}_d| \times R_{num} = 1000 \times 20$), and k is the test length starting from 0 to $L - 1$ (where $L = 15$). The probability $p(\cdot)$ is obtained from the Monte-Carlo roll-out prediction [9] (since we do not have any POMDP files, we cannot obtain the true predictions or calculations), and the estimated probability $\hat{p}(\cdot)$ is computed by the learnt model.

6.2. Parameter Selection in ADMM

We use a grid search method to choose the suitable parameters μ and α_i ($i = 1, 2, \ldots, N$) in the ADMM procedure in Algorithm 4. The details are shown in the following.

- Let s_1 and s_2 be integer numbers in the intervals $[1, 3]$ and $[1, 8]$, respectively, and then create the search space for parameters $\mu = e^{-s_1}$ and $\alpha_i = e^{s_2}/(1 - e^{-s_2}) * \frac{N\|D_{(i)}\|_F^2}{2log(n_i)}$, where e is Euler's number;

- Run Algorithm 1 to obtain the optimal solution X and its non-zero column index set I for each parameter combination in the above space for each agent;

- Compute the value of objective function

  $$v_{obj} = \frac{1}{2}\|\mathcal{D} - \mathcal{D} \times_1 X_1 \times_2 X_2 \times_3 \cdots \times_N X_N\|_F^2 + \alpha_1 \|X_1\|_{2,1} + \alpha_2 \|X_2\|_{2,1} + \cdots + \alpha_N \|X_N\|_{2,1},$$

  and the size of core joint test set ($|\mathbf{Q}| = \prod_{i=1}^N |I_i|$), then we have the gap

  $$gap = \sqrt{v_{obj}^2 + |\mathbf{Q}|^2}$$

  for each parameter combination;

Table 5. Selected parameters in various domains with two or three agents

N	Domain	μ^*	α_i^*	$\lvert\mathbf{Q}\rvert$	$\lvert\mathcal{D}\rvert$
2	Tag	0.449329	3.15334,1.50441	$9 \times 1 = 9$	74,74,4661
	Gridworld*	0.149569	1.57363,0.750751	$8 \times 2 = 16$	59,59,4673
	ColoredGridworld*	0.049787	11.59446, 2.740408	$7 \times 6 = 42$	73,74,4725
	Poc-Man*	0.049787	9.584841,2.265424	$9 \times 5 = 45$	76,95,4441
3	Tag	0.449329	11.20784,3.751926,3.751926	$9 \times 4 \times 1 = 36$	75,77,71,4897
	Gridworld*	0.149569	7.04372, 2.35795,0.305283	$3 \times 3 \times 3 = 27$	58,61,59,4882
	ColoredGridworld*	0.449329	21.75155,21.75155,21.75155	$16 \times 14 \times 1 = 224$	59,64,107,4666
	Poc-Man*	0.449329	15.56419,15.56419,15.56419	$16 \times 5 \times 2 = 160$	113,144,113,4726

- Select the parameters μ^* and α_i^* that achieve the minimum gap.

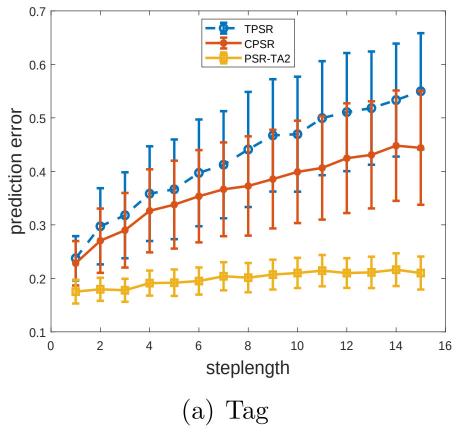
For each round of testing of each domain in a multi-round test setting, there are inaccuracies and inconsistencies in the system dynamics tensor obtained from the training data, which ultimately leads to inconsistencies and changes of the obtained parameters μ^* and α_i^* during model learning. Therefore, it is not necessary to keep every specified value of parameters μ^* and α_i^* for each domain in a multi-round test. For example, Table 5 shows some selected parameters μ^* and α_i^* of the four domains with 2 agents or 3 agents. We observe that in some domains with 3 agents, the value α_i^* is identical for every agent and the size of core joint test set ($|\mathbf{Q}|$) is very large. Therefore, we can reduce the parameter search space to two dimensions (μ^* and α_i^*), and add a constraint on the output coefficient matrices X_i ($i = 1, 2, \ldots, N$), i.e., we only select the columns of these matrices whose L_2-norm is in top-k ($k = 5$). As a result, we can reduce the computational time and required memory of data and computations in the grid search, learning model parameters and making prediction as well.

6.3. Comparative Results

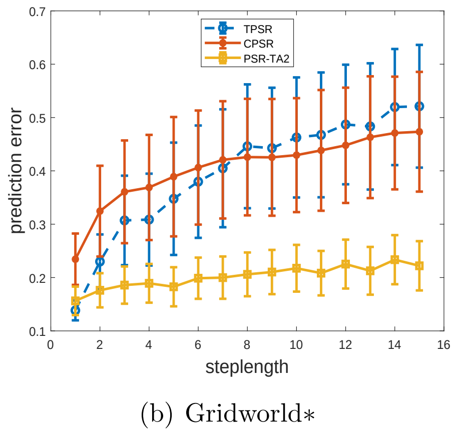
We demonstrate the comparison results of the PSR models in this section, including the one-step prediction accuracy and the running time. We denote Algorithm 3 as PSR-TA2 and Algorithm 4 for three-agent as PSR-TA3 for short. Specifically, we compare the prediction error of TPSR, CPSR and PSR-TA2 in four two-agent domains (i.e., Tag, Gridworld$*$, ColoredGridworld$*$, and Poc-Man$*$) in Fig. 5 and the corresponding running time in Fig. 6. We further compare the prediction error of TPSR, CPSR and PSR-TA3 in four three-agent domains (i.e., Tag, Gridworld$*$, ColoredGridworld$*$, and Poc-Man$*$) in Fig. 7.

In Fig. 5, the x-axis represents the step length of action-observation and the y-axis represents the prediction error of N_T trials of each model evaluated by Eq. (27). To comprehensively compare the performance of the learnt models, we present the mean value and standard deviation of the prediction errors in the four two-agent domains. We observe that the prediction error of our algorithm (PSR-TA2) increases slightly with the step-length as compared with TPSR and CPSR for almost all cases in two-agent systems except in ColoredGridworld$*$. The learnt model by our algorithm is more accurate than others, especially in Tag and the very large domain Poc-Man$*$.

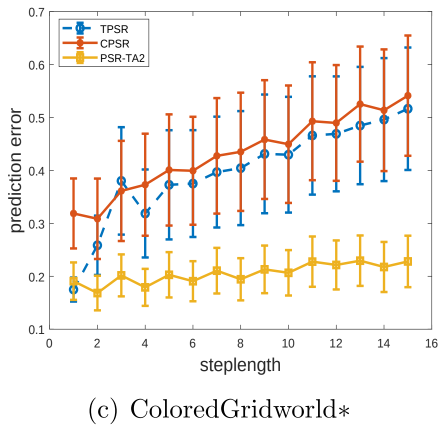
The corresponding running time of each algorithm in four domains is given in Fig. 6, including the time of three main parts: data preprocessing, modelling and making prediction.



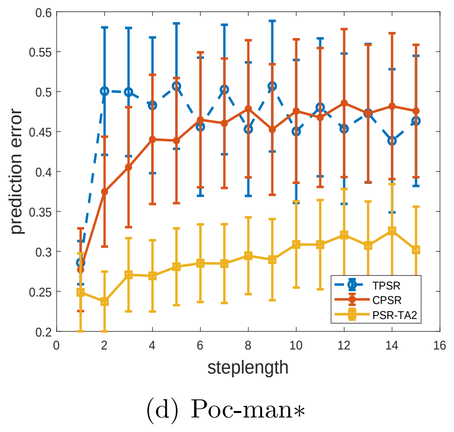
(a) Tag



(b) Gridworld∗



(c) ColoredGridworld∗



(d) Poc-man∗

Fig. 5. Prediction errors of TPSR, CPSR and PSR-TA2 for the two-agent domains Tag, Gridworld∗, GridworldColor∗ and Poc-Man∗

In the data preprocessing phase, we compute the time of establishing and normalizing the dynamics matrix (tensor) of the system and the auxiliary matrices required in an algorithm. The modelling time mainly includes finding a core test set in a PSR model and learning model parameters, that is, solving the discovery and learning problems. For the TPSR and CPSR methods, the singular value decomposition (SVD) operation on the system dynamics matrix is required, while our algorithm PSR-TA2 needs to solve the tensor optimization problem. We have the following observations:

• *Data preprocessing.* Compared to CPSR and PSR-TA2, TPSR method costs much more time in this phase since TPSR needs to construct more auxiliary matrices before applying SVD on a system dynamics matrix;

• *Modelling.* Compared to the traditional CPSR method, the computations of the auxiliary tensor $\mathcal{A}$ and matrix X required in our method may need more time due to the larger state space and observation space of domain ColoredGridworld∗. Thanks to the benefit of the sparse table, the time of PSR-TA2 is less than others for domains without a large state space and observation space. Notice that the grid search method in selecting parameters can be accelerated by parallel computing.

• *Making prediction.* Without the help of reduction of coefficient matrices X_i (i =

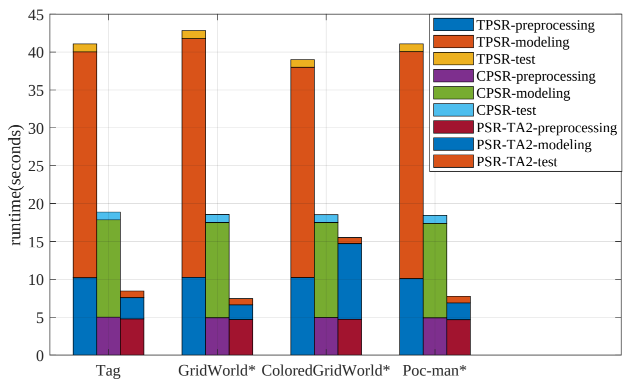
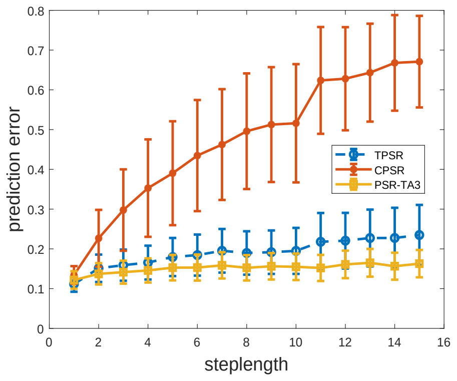


Fig. 6. Runtime in building and testing the models for four different two-agent domains

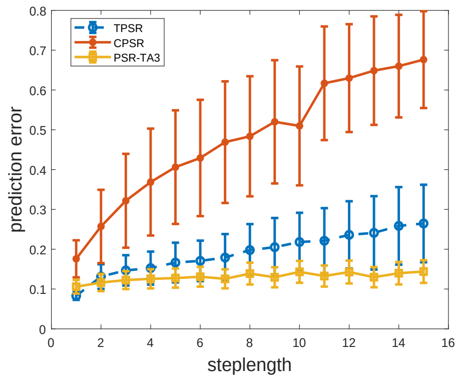
$1, 2, \ldots, N$), our algorithm takes about the same amount of time or even more than other methods in this phase because our algorithm obtains a larger set of projection vectors in all domains (see Table 5). It eventually takes more time to update the states of the PSR model during prediction execution. Therefore, it is naturally considered to perform an important feature screening on the obtained core test set, we thus add a constraint on the output coefficient matrices X_i ($i = 1, 2, \ldots, N$) to help us eliminate the less important core tests and reduce the time cost of the algorithm for predicting. Then, we preset the size of state vectors for all the three methods to the same value in each domain. Hence, the consuming time of all the three methods are almost the same.

In Figs. 7-8, we add one more agent in all the domains. For all horizons of these four domains, PSR-TA3 performs better than the other two algorithms and produces more competitive predictions. The modelling time of TPSR increases rapidly especially in Poc-man$*$ because the system dynamics matrices become huge as the number of agents increases. PSR-TA3 is not effected by the increase of the size of system dynamics tensor ascribed to the usage of sparse representation of tensor and the mapping technique. All of the compared models give more accurate prediction in these domains, and the predictive uncertainty of all models accumulates as the time step increases, leading to the oscillation of the final prediction error, which gradually expands or even diverges. For all horizons of these four domains, the prediction error curves of our method has either a slow rise, a small dynamic oscillation, or a small fluctuation (small standard deviation), which means it has higher stability and accuracy than other models.

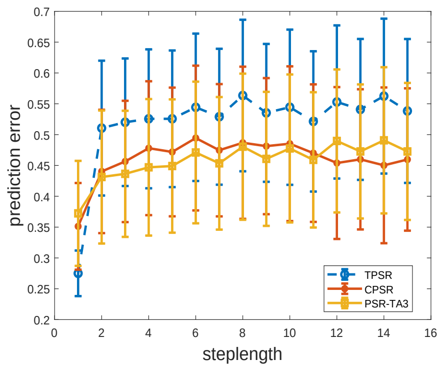
In summary, our algorithms perform better, partially because the tensor approach can exploit the embedded connections of high dimensional data and it is not significantly effected by noisy data generated in a dynamic system. As we see in Table 4, adding extra agents into the system will cause much more complicated interactions between agents. In domains with a very large set of states and action-observations, the system dynamics tensor eventually becomes a space with high dimensional and sparse data, which exceeds the maximum size of a matrix that can be constructed and calculated in Matlab. Hence, we can not further conduct the scalability tests of our algorithms in these large problem domain and will investigate the scalability in a commonly used small problem domain in Section 6.4.



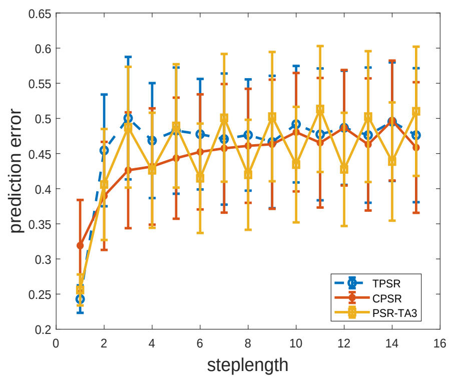
(a) Tag (N=3)



(b) Gridworld$*$ (N=3)



(c) ColoredGridworld$*$(N=3)



(d) Poc-man$*$(N=3)

Fig. 7. Prediction errors of TPSR, CPSR and PSR-TA3 for three-agent domains Tag, Gridworld$*$, GridworldColor$*$ and Poc-Man$*$

6.4. Scalability Test

We choose the well-known *Tiger* problem domain [14] - a classical benchmark to measure the performance of POMDP models. To evaluate our algorithm, we propose a multi-agent version of the *Tiger* problem as shown in Fig. 9. In the multi-agent Tiger domain, the agents are randomly assigned to a room with a door, they have to avoid being eaten by the tiger when they open the door. Of course, they can choose to listen when they are not sure about the tiger position. The set of actions for each agent is $A = \{Open,\ Listen\}$. The agent fails to execute an action with the probability 0.1. If this happens, the agent randomly choose the other action. The set of all possible observations of each agent is $O = \{Null, Tiger\}$. When the agent listens, it will receive an observation with the probability 0.85; otherwise, it opens the door and receives the observation with the probability 0.5. The door of room without an agent will be randomly opened with the probability 0.1. The tiger randomly moves to a door if the door is opened, and will stay in the same position if all doors are closed. The termination condition of the game is that only one agent is alive.

We set the maximum length of the joint tests in the joint test set as 1 for more than 6 agents, 2 for more than 3 agents but fewer than 6 agents, and 3 for fewer than 4 agents. Under this setting, we can obtain a more sufficient system dynamics tensor without exceeding

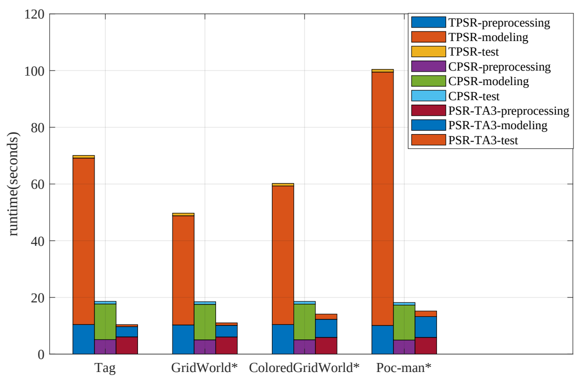
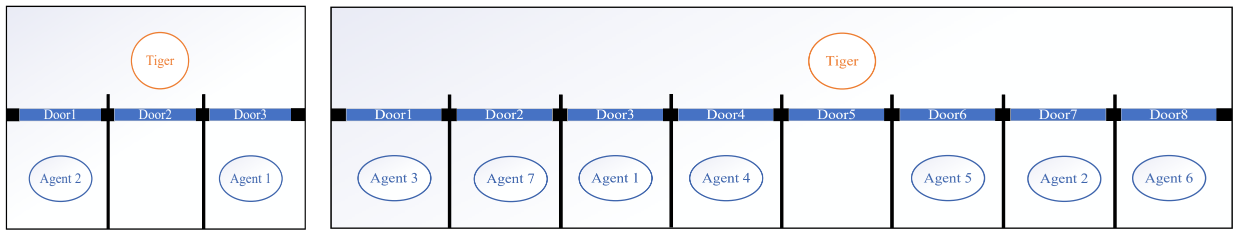


Fig. 8. Runtime in building and testing the models for four different three-agent domains



(a) Tiger@2 (b) Tiger@7

Fig. 9. The multi-agent *Tiger* domain with two agents and seven agents

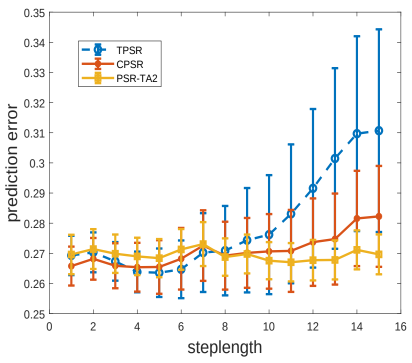
the maximum size of a matrix that can be constructed and calculated in Matlab.

In Figs. 10(a)-10(f), we add one agent at each time in the tests. For all horizons, PSR-TAM($M = 2, \dots, 7$) performs better than the other two algorithms and produces more competitive predictions. The corresponding running time of each algorithm in four domains is given in Fig. 11. The modelling time of TPSR increases rapidly especially in the domain with seven agents. This is because the system dynamics matrices become huge as the number of agents increases. PSR-TA7 is not effected by the increase of the size of system dynamics tensor ascribed to the usage of a sparse representation of tensor and the mapping technique. For all horizons of these four domains, the prediction error boxes of our model are small, which shows that the algorithm is more reliable than other models.

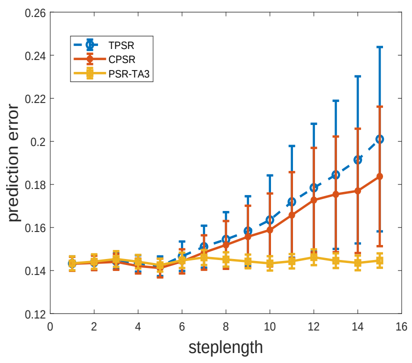
In summary, our tensor-based PSR method performs better than the state-of-art PSR methods for dealing with multi-agent PSR problems. Meanwhile, it achieves better scalability in the domain with a small set of actions and observations. Due to the limited matrix computational space in Matlab, more complicated domains can't be processed to compare the results. We will investigate more efficient matrix decomposition techniques to improve the proposed algorithm.

7. Related Works

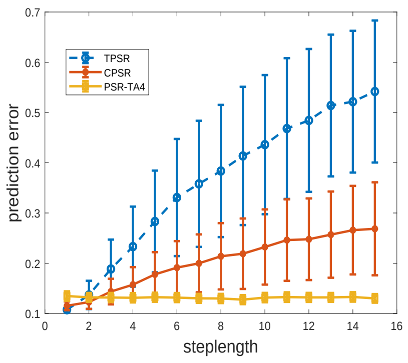
A PSR model utilizes a function of statistic vectors about future action-observation sequences to capture the state of a dynamical system, which was first proposed by Littman et



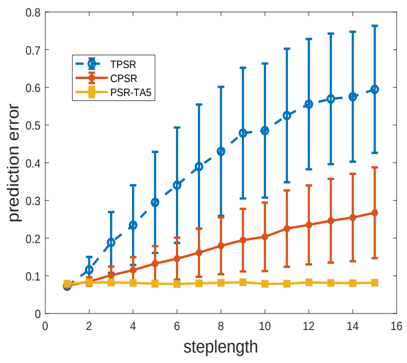
(a) Tiger@2



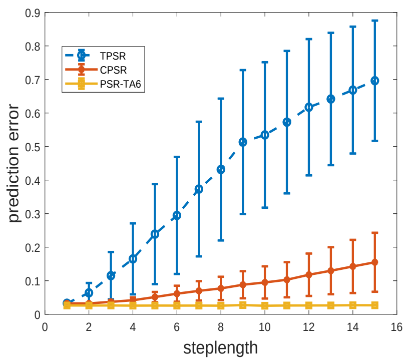
(b) Tiger@3



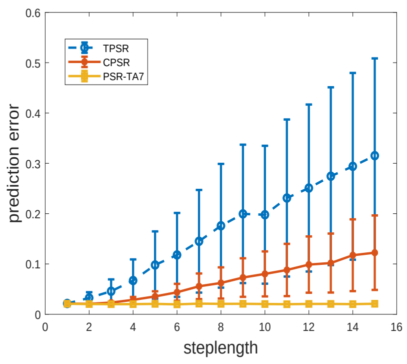
(c) Tiger@4



(d) Tiger@5



(e) Tiger@6



(f) Tiger@7

Fig. 10. Prediction errors of TPSR, CPSR and PSR-TA2 for multi-agent *Tiger* domain

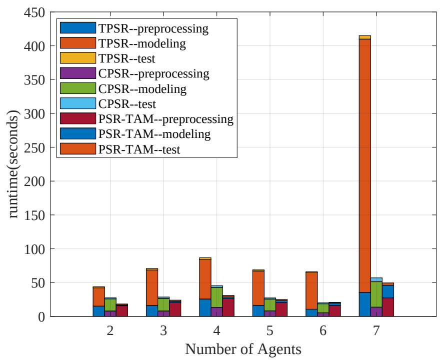


Fig. 11. Runtime in building and testing the models for multi-agent Tiger domain

al. [17] in 2001. Their work not only established theoretical foundations of PSR, but also illustrated the advantages of the PSR models compared to other common approaches, such as hidden Markov model (HMM) and partially observable Markov decision process (POMDP), and discussed the conversion relationship between the PSR models and others.

After nearly two decades of development, PSR becomes more and more powerful in modelling a dynamical system, especially when we can't explicitly enumerate the system states or the state space is very large. For example, Hefny et al. [10] proposed recurrent predictive state policy (RPSP) networks - a reinforcement learning approach with a recurrent architecture and used PSR represent a state in POMDPs environments. Liu et al. [18] introduced an approach with the benefit of the theoretical advantages of PSRs and no prior knowledge of the underlying system is required for online modelling and planning POMDPs domains. Zhang et al. [33] developed a method that extracts causal state representations from a recurrent neural network for representing states, which generalizes PSRs to non-linear predictive models. In the following, we summarize the relevant literatures of learning PSR models as three parts according to the methodology, i.e. matrix-based approaches and other techniques.

Inspired by the basic theory and powerful expression of PSR models, researchers have developed a number of mathematical methods for learning PSR. James et al. [12] studied a controllable and resettable dynamic system and first provided discovery and learning algorithm for a PSR model mainly based on the rank theory of matrices. McCracken et al. [21] proposed a constrained gradient descent method that discovers the core tests of PSR by using a very small amount of training data and provide an accurate predictions when having sufficient data, which improves the efficiency and accuracy of a PSR model. Moreover, by taking advantages of the landmarks, James et al. [13] proposed a PSR model called memory-PSRs that utilize memories of the past and predictions of the future. Some researchers were also dedicated to learning approaches for non-resettable dynamical systems, such as suffix-history algorithm [29] and learning PSR models from a single sequence (i.e., history) [28].

The matrix-based modelling approaches for learning PSR models have become very popular and been well-studied since the system dynamics matrix [27] was proposed, e.g., the spectral learning approach [2, 1, 3, 15, 16, 24, 11] and the compressed sensing approach [9, 8]. The spectral learning methods tried to solve PSR models by analysing the principal components of system dynamics matrix. Kulesza et al. [16] proposed a spectral learning algorithm that can handle insufficient data, which could reduce prediction errors compared to standard spectral learning approaches. Kulesza et al. [15] further proposed a low-rank spectral learning method with a particular weighted loss function to overcome the consequence of discarding arbitrarily small singular values of the system dynamics matrix [16]. They also gave a bound on the loss of the learned low-rank model in terms of the discarded singular values. Recently, Huang et al. [11] developed an approach for finding a limited set of columns (basis selection) in spectral learning of PSR and used a model entropy to evaluate the accuracy of the learnt PSR model. Meanwhile, TPSR method proposed by Rosencrantz et al. [24] efficiently alleviated the learning progress of PSR based on matrix SVD. Subsequently, some variants of TPSR were proposed in the recent years. For example, Boots et al. [2, 3] applied incremental SVD and random projections for scaling the traditional TPSR model to online TPSR, which suits for a complex system with an extremely large dataset. Hamilton et al. [9, 8] presented compressed TPSR for relatively large domains via employing a particularly sparse structure of a system dynamics matrix.

In parallel, Liu et al. [19] utilized a landmark technique for partitioning an entire state space into several sub-state space, learnt a local PSR model for each sub-state space, and combined all local PSR models into a whole PSR for a dynamical system. They also modelled the discovery problem as a sequential decision making problem and solved it via Monte-carlo tree search [20] . Zeng et al. [32] employed a group lasso technique and formulated the discovery problem as an optimization problem without specifying the number of core tests in advance, which had a well performance in a large domain. However, all the existing algorithms were solely proposed for learning a single-agent PSR model.

8. Conclusion and Future Work

We investigate an extension of a single-agent PSR model in a multi-agent setting. The challenge lies in learning the model parameters from the high-dimensional data that records how multiple agents interact over times. Tensors seem to be a natural way to describe high-dimensional data owing to its multidimensionality. In this paper, by utilizing the system dynamics tensor to represent the interaction data of multiple agents, we formulate the PSR discovery problem as a tensor optimization problem with group lasso, where the ADMM method is adopted to efficiently solve its subproblems. With the benefit of a group sparsity structure of the optimization solutions, we can easily get the core joint test test and learn the prediction parameters and state vectors. Moreover, we obtain the transition parameters of PSR model through linear regression. Hence, the PSR model of the underlying system can be established accordingly. We utilize a sparse representation of tensor and mapping technique for skipping all the time-consuming and memory-expensive operations when implementing our algorithms. Experimental results show that our tensor approach is capable of learning multi-agent PSR models on several domains including one large domain. We also observe that our method outperforms two other popular methods (TPSR and CPSR) in terms of prediction error, especially for the case of more agents.

This work is the first attempt at learning a multi-agent PSR model. We are planning to develop more efficient technique to solve the optimization problem, and explore other ways for learning the PSR models, e.g., incremental learning by partitioning the system dynamics tensor and taking advantage of the tensor sparsity. This will allow us to improve the scalability of our proposed methods.

Acknowledgements

Dr. Bilian Chen and Dr. Yinghui Pan were supported in part by the National Natural Science Foundation of China (Grants No. 61772442, 61806089 and 61836005). Professor Yifeng Zeng and Dr. Biyang Ma thanks the support of the EPSRC New Investigator Award in 2019.

References

[1] B. Boots and G.J. Gordon. Predictive state temporal difference learning. *Advances in Neural Information Processing Systems*, 23(0):271279, 2010.

[2] B. Boots and G.J. Gordon. An online spectral learning algorithm for partially observable dynamical systems. *AAAI'11 Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, (0):293–300, 2011.

[3] B. Boots, S. Siddiqi, and G.J. Gordon. Closing the learning-planning loop with predictive state representations. *International Journal of Robotics Research*, 30(7):954–966, 2010.

[4] S. Boyd, N. Parikh, E. Chu, Peleato B., and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. 3(1):1–122, 2011.

[5] B. Chen, B. Ma, Y. Zeng, Cao L., and Tang J. Tensor decomposition for multi-agent predictive state representation. *arXiv: Multiagent Systems*, 2020.

[6] C. Ding, D. Zhou, X. He, and H. Zha. r_1-pca: rotational invariant l1-norm principal component analysis for robust subspace factorization. In *Proceedings of the 23rd International Conference on Machine Learning. ICML '06. Pittsburgh, Pennsylvania, USA: ACM*, page 281288, 2006. 10.1145/1143844.1143880.

[7] P. J Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research (JAIR)*, 24:49–79, 2005.

[8] W.L. Hamilton, M. MilaniFard, and J. Pineau. Modelling sparse dynamical systems with compressed predictive state representations. In *Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, JMLR:W&CP*, volume 28, pages 178–186, 2013.

[9] W.L. Hamilton, Mahdi M. Fard, and J. Pineau. Efficient learning and planning with compressed predictive states. *Journal of Machine Learning Research*, 15:3395–3439, 2014.

[10] A. Hefny, Z. Marinho, W. Sun, S. Srinivasa, and G. Gordon. Recurrent predictive state policy networks. *arXiv preprint arXiv:1803.01489*, 2018.

[11] C. Huang, Y. An, Z. Sun, Z. Hong, and Y. Liu. Basis selection in spectral learning of predictive state representations. *Neurocomputing*, 310(1):183–189, 2018.

[12] M.R. James and S.P. Singh. Learning and discovery of predictive state representations in dynamical systems with reset. In *Proceedings of the 21st International Conference on Machine Learning, Banff, Alberta, Canada*, pages 695–702, 2004.

[13] M.R. James, B. Wolfe, and S. Singh. Combining memory and landmarks with predictive state representations. In *International Joint Conference on Artificial Intelligence*, pages 734–739, 2005.

[14] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99 – 134, 1998.

[15] A. Kulesza, N. Jiang, and S. Singh. Low-rank spectral learning with weighted loss functions. pages 517–525, 2015.

[16] A. Kulesza, N. Jiang, and S. Singh. Spectral learning of predictive state representations with insufficient statistics. In *AAAI'15 Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI Press*, pages 2715–2721, 2015.

[17] M.L. Littman and S. Singh. Predictive representations of state. In *International Conference on Neural Information Processing Systems: Natural and Synthetic*, pages 1555–1561, 2001.

[18] Y. Liu and J. Zheng. Online learning and planning in partially observable domains without prior knowledge. *arXiv preprint arXiv:1906.05130*, 2019.

[19] Y.L. Liu, Y. Tang, and Y. Zeng. Predictive state representations with state space partitioning. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum(eds.)*, pages 1259–1266, 2015.

[20] Y. Liu, H. Zhu, Y. Zeng, and Z. Dai. Learning predictive state representations via monte-carlo tree search. pages 3192–3198, 2016.

[21] P.N. McCracken and M.H. Bowling. Online discovery and learning of predictive state representations. In *Advances in Neural Information Processing Systems*, volume 18, pages 875–882, 2006.

[22] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International Joint Conference on Artificial Intelligence*, pages 1025–1030, 2003.

[23] Z. Qin, K. Scheinberg, and D. Goldfarb. Efficient block-coordinate descent algorithms for the group lasso. *Mathematical Programming Computation*, 5(2):143–169, 2013.

[24] M. Rosencrantz, G.J. Gordon, and S. Thrun. Learning low dimensional predictive representations. In *International Conference on Machine Learning*, page 88, 2004.

[25] S. Seuken and S. Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[26] D. Silver and J. Veness. Monte-carlo planning in large pomdps. In *Neural Information Processing Systems*, pages 2164–2172, 2010.

[27] S.P. Singh, M. James, and M. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Conference on Uncertainty in Artificial Intelligence*, pages 512–519, 2004.

[28] E. Wiewiora. Learning predictive representations from a history. In *Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany*, pages 969–976, 2005.

[29] B. Wolfe, M.R. James, and S.P. Singh. Learning predictive state representations in dynamical systems without reset. In *International Conference on Machine Learning*, pages 980–987, 2005.

[30] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[31] Y. Zeng and P. Doshi. Exploiting model equivalences for solving interactive dynamic influence diagrams. *Journal of Artificial Intelligence Research (JAIR)*, 43:211–255, 2012.

[32] Y. Zeng, B. Ma, B. Chen, J. Tang, and M. He. Group sparse optimization for learning predictive state representations. *Information Sciences*, 412:1–13, 2017.

[33] A. Zhang, Z. C Lipton, L. Pineda, K. Azizzadenesheli, A. Anandkumar, L. Itti, J. Pineau, and T. Furlanello. Learning causal state representations of partially observable environments. *arXiv preprint arXiv:1906.10437*, 2019.