System Level Synthesis via Dynamic Programming

Shih-Hao Tseng, Carmen Amo Alonso, and SooJean Han

***Abstract*— System Level Synthesis (SLS) parametrization facilitates controller synthesis for large, complex, and distributed systems by incorporating system level constraints (SLCs) into a convex SLS problem and mapping its solution to stable controller design. Solving the SLS problem at scale efficiently is challenging, and current attempts take advantage of special system or controller structures to speed up the computation in parallel. However, those methods do not generalize as they rely on the specific system/controller properties.**

We argue that it is possible to solve general SLS problems more efficiently by exploiting the structure of SLS constraints. In particular, we derive dynamic programming (DP) algorithms to solve SLS problems. In addition to the plain SLS without any SLCs, we extend DP to tackle infinite horizon SLS approximation and entrywise linear constraints, which form a superclass of the locality constraints. Comparing to convex program solver and naive analytical derivation, DP solves SLS 4 to $12\times$ faster and scales with little computation overhead. We also quantize the cost of synthesizing a controller that stabilizes the system in a finite horizon through simulations.

I. Introduction

System Level Synthesis (SLS) facilitates the incorporation of system level constraints (SLCs) by its parametrization of closed-loop system responses [1], [2]. Subsequently, SLS greatly simplifies the controller synthesis problem of large-scale, complex, distributed systems into a convex program. Given matrices A and B depending on the system dynamic, SLS formulates the following convex program

$$\min \quad g(\mathbf{\Phi_x}, \mathbf{\Phi_u}) \tag{1}$$

$$\text{s.t.} \quad \begin{bmatrix} zI - A & -B \end{bmatrix} \begin{bmatrix} \mathbf{\Phi_x} \\ \mathbf{\Phi_u} \end{bmatrix} = I, \tag{1a}$$

$$\mathbf{\Phi_x}, \mathbf{\Phi_u} \in z^{-1}\mathcal{RH}_\infty,$$

$$\begin{bmatrix} \mathbf{\Phi_x} \\ \mathbf{\Phi_u} \end{bmatrix} \in \mathcal{S}, \tag{1b}$$

where g is the objective, $\mathcal{S}$ the set of SLCs, and $z^{-1}\mathcal{RH}_\infty$ a subset of strictly proper stable transfer functions. The system responses $\{\mathbf{\Phi_x}, \mathbf{\Phi_u}\}$ are transfer functions given by

$$\mathbf{\Phi_x} = \sum_{\tau=1}^{T} z^{-\tau}\Phi_x[\tau], \qquad \mathbf{\Phi_u} = \sum_{\tau=1}^{T} z^{-\tau}\Phi_u[\tau]$$

with horizon T and spectral components $\Phi_x[\tau]$ and $\Phi_u[\tau]$. In what follows, we assume that the objective g can be decomposed into a sum of per-step costs

$$g(\mathbf{\Phi_x}, \mathbf{\Phi_u}) = \sum_{\tau=1}^{T} g_\tau(\Phi_x[\tau], \Phi_u[\tau]).$$

Shih-Hao Tseng, Carmen Amo Alonso, and SooJean Han are with the Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125, USA. Emails: {shtseng,camoalon,soojean}@caltech.edu

Once (1) is solved, SLS gives a state-feedback controller $\mathbf{\Phi_u}\mathbf{\Phi_x}^{-1}$ that can stabilize the systems in horizon T. The straightforward SLC incorporation makes SLS more powerful for often-constrained practical control tasks than legacy optimal control methods (such as LQG or PID control). Meanwhile, model predictive control (MPC) can also benefit from the SLS parametrization in dealing with online constraints [3].

Despite the advantageous SLS parametrization that transforms a complex controller synthesis problem into a tractable convex program, in practice, solving (1) is still computationally demanding, especially when dealing with large-scale systems. To accelerate the solving process, existing proposals, such as [3]–[6], focus on specially structured systems and controllers (e.g., localizable systems) which allow the decomposition of (1) for parallel processing. However, for general systems without the desired structures, those methods are no longer applicable, and we resort to the heuristics or programming techniques in convex program solvers to speed up the solving process.

We then ask the question: *Is it possible to expedite the solving process without special structural assumptions?* Our answer is affirmative. So far the existing proposals essentially impose various structural constraints through (1b), and we have not yet fully exploited the structure of the SLS constraint (1a). By treating the system responses as state and control of a system, one can solve the SLS problem by dynamic programming (DP) [7], which is highly efficient. This opportunity is also noticed in [4], where the authors applied the DP principle for SLS with a quadratic cost. But to do so, [4] requires additional input coupling assumption: The boundary states must be directly controlled by the corresponding boundary actuators. A fully general DP algorithm remains open.

A. Contributions and Organization

In Section II, we derive the DP algorithms to solve the SLS problem (1). Starting with plain SLS without SLCs (1b), we then provide the approximation to infinite horizon SLS and incorporate a class of SLCs, the entrywise linear constraints, into the DP process. The entrywise linear constraints are generalized versions of the sparsity/locality constraints in the literature [1]. To demonstrate the usage of the DP algorithms, we adopt them for the SLS problems with a $\mathcal{H}_2$ objective in Section III. Through extensive simulations in Section IV, we show that DP algorithms are more scalable and can outperform existing convex program solver by 4 to $12\times$ and naive Lagrange multiplier method by 10 to $38\times$. We also quantify the synthesis overhead for stabilizing a system in a finite horizon by comparing DP with DP approximation to

infinite horizon SLS. Finally, we conclude in Section V with future research directions.

B. Notation

We use lower- and upper-case letters (such as x and A) to denote vectors and matrices respectively, while bold lower- and upper-case characters and symbols (such as $\mathbf{u}$ and $\mathbf{\Phi_u}$) are reserved for signals and transfer matrices. Let A^{ij} be the entry of A at the i^{th} row and j^{th} column. We denote by A^+ the pseudo inverse (Moore-Penrose inverse). We vectorize a matrix A to be the vector $\overrightarrow{A}$ by stacking its columns. Inversely, we rebuild the matrix $\overleftarrow{x}$ from a vector x by realigning the elements. The null space of a matrix Ψ is written as $\text{null}\,(\Psi) = \{v : \Psi v = 0\}$, where 0 is an all-zero vector. We slightly abuse the notation to write $A \in \text{null}\,(\Psi)$ if all columns in A are in $\text{null}\,(\Psi)$. Let $\|\mathbf{\Phi_u}\|_{\mathcal{H}_2}^2$ be the $\mathcal{H}_2$ norm of a transfer function $\mathbf{\Phi_u}$, which is given by $\sum_{t=0}^{\infty} \|\Phi_u[t]\|_F^2$ with $\|\cdot\|_F$ the Frobenius norm.

II. Dynamic Programming Algorithms

We illustrate the dynamic programming (DP) algorithms for state-feedback SLS problems. We begin with DP for plain SLS and reduce it as an approximation to infinite horizon SLS problems. We then extend the DP algorithm to handle entrywise linear constraints.

Before our DP derivations, we introduce two lemmas here to use later. The proofs are trivial and omitted.

Lemma 1. *Given a matrix Ψ, $\text{null}\,(\Psi)$ is a subspace and there exists some matrix Ξ such that $\text{null}\,(\Psi) = \{v : v = \Xi\theta\}$ where θ is an arbitrary vector.*

Lemma 2. *The intersection of $\text{null}\,(\Psi_a)$ and $\text{null}\,(\Psi_b)$ is $\text{null}\left(\begin{bmatrix}\Psi_a\\ \Psi_b\end{bmatrix}\right)$.*

A. Plain SLS

We first derive the DP algorithm for plain SLS without constraint (1b). Notice that we can rewrite the SLS constraints (1a) in the following form with spectral components:

$$\Phi_x[\tau+1] = A\Phi_x[\tau] + B\Phi_u[\tau], \quad \forall \tau = 1,\dots,T-1, \tag{2a}$$
$$\Phi_x[1] = I, \tag{2b}$$
$$A\Phi_x[T] + B\Phi_u[T] = 0. \tag{2c}$$

Treating $\Phi_x[\tau]$ as state variable $X[\tau]$ and $\Phi_u[\tau]$ as control $U[\tau]$ at each time τ, the SLS problem is equivalent to the following discrete time control problem with state dynamics (2a), initial condition (2b), and boundary condition (2c):

$$\min \quad \sum_{\tau=1}^{T} g_\tau(X[\tau], U[\tau]) \tag{3}$$
$$\text{s.t.} \quad X[\tau+1] = AX[\tau] + BU[\tau], \quad \forall \tau = 1,\dots,T-1, \tag{3a}$$
$$X[1] = I, \tag{3b}$$
$$AX[T] + BU[T] = 0. \tag{3c}$$

To solve the above problem by DP, we have to address the following issues:

- Compute the cost-to-go function $V_\tau(X[\tau])$ recursively backwards in time according to the state dynamic (3a).
- Ensure the boundary condition (3c) can be satisfied at each step of the backward recursion.

The form of the cost-to-go function V_τ depends on the objective g; we will explicitly derive V_τ for the specific $\mathcal{H}_2$ objective in Section III. For now, we derive $V_\tau(X[\tau])$ implicitly by definition via the following recursive relationship:

$$V_\tau(X[\tau]) = \min_{\hat{U}\in\mathcal{A}_U[\tau]} g_\tau(X[\tau],\hat{U}) + V_{\tau+1}(AX[\tau]+B\hat{U}), \quad \forall \tau = 1,\dots,T \tag{4}$$

where $\mathcal{A}_U[\tau]$ denotes the admissible set of U at time τ and $V_{T+1}(\cdot) = 0$.

Accordingly, the control $U[\tau]$ at each time τ is given by

$$\begin{aligned} U[\tau] &= \operatorname*{argmin}_{\hat{U}\in\mathcal{A}_U[\tau]} g_\tau(X[\tau],\hat{U}) + V_{\tau+1}(AX[\tau]+B\hat{U}) \\ &= K_\tau(X[\tau]), \end{aligned} \tag{5}$$

and the state $X[\tau]$ follows (3a) and (3b).

We assign $X[T+1]$ to be an all-zero matrix, so that $X[T+1] \in \text{null}\,(I) = \text{null}\,(\Psi_x[T+1])$ and (3c) is met in the form of (3a). Given $X[\tau+1] \in \text{null}\,(\Psi_x[\tau+1])$, the following theorem then constructs $\mathcal{A}_U[\tau]$ and asserts that $X[\tau] \in \text{null}\,(\Psi_x[\tau])$ to satisfy the boundary condition (3c) at the end of backward recursion.

Theorem 1. *Suppose $X[\tau+1] \in \text{null}\,(\Psi_x[\tau+1])$ for some given matrix $\Psi_x[\tau+1]$. We have*

$$\mathcal{A}_U[\tau] = \{\hat{U} : \hat{U} = Q_X X[\tau] + Q_\Lambda \Lambda\}$$

where

$$\begin{aligned} \Gamma[\tau] &= \begin{bmatrix} -B & \Xi_x[\tau+1]\end{bmatrix}, \\ Q_X &= \begin{bmatrix} I & 0\end{bmatrix}\Gamma^+[\tau]A, \\ Q_\Lambda &= \begin{bmatrix} I & 0\end{bmatrix}(I - \Gamma^+[\tau]\Gamma[\tau]), \end{aligned}$$

and $\Xi_x[\tau+1]$ is given by Lemma 1 for $\text{null}\,(\Psi_x[\tau+1])$.

Also, $X[\tau] \in \text{null}\,(\Psi_x[\tau])$ where

$$\Psi_x[\tau] = (\Gamma[\tau]\Gamma^+[\tau] - I)A.$$

Proof. Since $X[\tau+1] \in \text{null}\,(\Psi_x[\tau+1])$, by Lemma 1, there exists a matrix $\Xi_x[\tau]$ such that we can express each column in $X[\tau+1]$ as $\Xi_x[\tau]\theta$ for some vector θ. In other words, there exists a matrix Θ such that

$$\Xi_x[\tau+1]\Theta = X[\tau+1] = AX[\tau] + BU[\tau].$$

Rearranging the terms and applying the definition of $\Gamma[\tau]$ yield

$$\begin{bmatrix} -B & \Xi_x[\tau+1]\end{bmatrix}\begin{bmatrix} U[\tau]\\ \Theta\end{bmatrix} = \Gamma[\tau]\begin{bmatrix} U[\tau]\\ \Theta\end{bmatrix} = AX[\tau].$$

Therefore, we must have

$$\begin{bmatrix} U[\tau]\\ \Theta\end{bmatrix} = \Gamma^+[\tau]AX[\tau] + (I - \Gamma^+[\tau]\Gamma[\tau])\Lambda, \tag{6}$$

Algorithm 1: DP for plain SLS.

Input: A, B and objective $g(\mathbf{\Phi_x}, \mathbf{\Phi_u})$.
Output: $\Phi_x[\tau], \Phi_u[\tau]$ for all $\tau = 1, \dots, T$.
1: null $(\Psi_x[T+1]) =$ null (I).
2: $V_{T+1}(X[\tau]) = 0$.
3: **for** $\tau = T, \dots, 1$ **do**
4: Derive $\mathcal{A}_U[\tau]$ and null $(\Psi_x[\tau])$ from Theorem 1.
5: Compute $K_\tau(X[\tau])$ by (5).
6: Derive $V_\tau(X[\tau])$ from (4).
7: **end for**
8: $\Phi_x[1] = I$.
9: **for** $\tau = 1, \dots, T$ **do**
10: $\Phi_u[\tau] = K_\tau(\Phi_x[\tau])$.
11: $\Phi_x[\tau+1] = A\Phi_x[\tau] + B\Phi_u[\tau]$.
12: **end for**

for some matrix Λ, or

$$U[\tau] \in \{\hat{U} : \hat{U} = Q_X X[\tau] + Q_\Lambda \Lambda\} = \mathcal{A}_U[\tau].$$

To ensure that a solution exists for (6), [8, Theorem 1] asserts that

$$\Gamma[\tau]\Gamma^+[\tau]AX[\tau] = AX[\tau].$$

As such, we have

$$X[\tau] \in \text{null}\left((\Gamma[\tau]\Gamma^+[\tau] - I)A\right) = \text{null}\left(\Psi_x[\tau]\right). \qquad \square$$

Together, we summarize the derivation in Algorithm 1.

B. Approximation to Infinite Horizon SLS

For infinite horizon SLS, the constraints (1a) become:

$$\begin{aligned} &\Phi_x[\tau+1] = A\Phi_x[\tau] + B\Phi_u[\tau], \quad \forall \tau = 1, \dots, \infty, \\ &\Phi_x[1] = I \end{aligned}$$

where the key difference from its finite counterpart (2) arises due to the absence of the boundary condition (2c).

A naive approximation to the infinite horizon SLS is to solve (3) without the condition (3c), which leads to

$$\begin{aligned} \min \quad & \sum_{\tau=1}^{T} g_\tau(X[\tau], U[\tau]) \\ \text{s.t.} \quad & X[\tau+1] = AX[\tau] + BU[\tau], \quad \forall \tau = 1, \dots, T-1, \\ & X[1] = I. \end{aligned}$$

The above optimization problem can also be solved by dynamic programming. We can obtain the corresponding DP algorithm by relaxing the feasible set $\mathcal{A}_U[\tau]$ to be the whole space and removing line 1 and 4 from Algorithm 1.

C. SLS with Entrywise Linear Constraints

We now elaborate on how to incorporate the system level constraints (1b) into DP. Especially, we consider the *entrywise linear constraints* as follows

$$\begin{aligned} \mathcal{S} = \{\mathbf{\Phi_x}, \mathbf{\Phi_u} : \ & \Phi_x[\tau] \in \mathcal{S}_x[\tau], \\ & \Phi_u[\tau] \in \mathcal{S}_u[\tau], \text{ for all } \tau = 1, \dots, T\}, \end{aligned}$$

where

$$\mathcal{S}_x[\tau] = \{\Phi : \overrightarrow{\Phi} = S_x[\tau]\overrightarrow{\Phi}\}, \tag{7a}$$

$$\mathcal{S}_u[\tau] = \{\Phi : \overrightarrow{\Phi} = S_u[\tau]\overrightarrow{\Phi}\}. \tag{7b}$$

The entrywise linear constraint (7) allows the entries in each spectral component to depend linearly on one another. It generalizes the sparsity constraints in the literature [1], which confines the non-zero entries of each spectral component.

Example 1 (Sparsity as Entrywise Linear Constraints)**.** Let $\Phi_x[\tau]$ be a 2×2 matrix. Consider the sparsity constraint which dictates $\Phi_x^{12}[\tau] = 0$. We can express the sparsity constraint as a entrywise linear constraint by enforcing a binary diagonal $S_x[\tau]$ with 0 at the corresponding entries:

$$\begin{aligned} &\overrightarrow{\Phi_x[\tau]} = S_x[\tau]\overrightarrow{\Phi_x[\tau]} \quad \Leftrightarrow \\ &\begin{bmatrix} \Phi_x^{11}[\tau] \\ \Phi_x^{12}[\tau] \\ \Phi_x^{21}[\tau] \\ \Phi_x^{22}[\tau] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Phi_x^{11}[\tau] \\ \Phi_x^{12}[\tau] \\ \Phi_x^{21}[\tau] \\ \Phi_x^{22}[\tau] \end{bmatrix} = \begin{bmatrix} \Phi_x^{11}[\tau] \\ 0 \\ \Phi_x^{21}[\tau] \\ \Phi_x^{22}[\tau] \end{bmatrix}. \end{aligned}$$

Entrywise linear constraints are more general than sparsity or locality constraints in that entries can exist beyond the diagonal or binary values.

To incorporate entrywise linear constraints into the DP algorithm, we consider the following vectorized version of (3):

$$\begin{aligned} \min \quad & \sum_{\tau=1}^{T} g_\tau(x[\tau], u[\tau]) \\ \text{s.t.} \quad & x[\tau+1] = \widetilde{A}x[\tau] + \widetilde{B}u[\tau], \quad \forall \tau = 1, \dots, T-1, \\ & x[1] = \overrightarrow{I}, \\ & \widetilde{A}x[T] + \widetilde{B}u[T] = 0, \end{aligned}$$

where $x[\tau] = \overrightarrow{\Phi_x[\tau]}$ and $u[\tau] = \overrightarrow{\Phi_u[\tau]}$. $\widetilde{A}$ and $\widetilde{B}$ are defined such that $\widetilde{A}x[\tau] = A\Phi_x[\tau]$ and $\widetilde{B}u[\tau] = B\Phi_u[\tau]$.

We first incorporate the condition (7b). By Lemma 1, we can express:

$$\begin{aligned} \mathcal{S}_u[\tau] &= \{\Phi : \overrightarrow{\Phi} \in \text{null}\left(S_u[\tau] - I\right)\} \\ &= \{\Phi : \overrightarrow{\Phi} = \Xi_{S_u}[\tau]\hat{v}\}. \end{aligned} \tag{8}$$

Therefore, we can enforce (7b) by requiring $u[\tau] = \Xi_{S_u}[\tau]v[\tau]$. Similar to the plain SLS case, we can derive $v[\tau]$, $u[\tau]$, and the cost-to-go function $V_\tau(x[\tau])$ by

$$v[\tau] = \underset{\hat{v} \in \mathcal{A}_v[\tau]}{\text{argmin}}\, g_\tau(x[\tau], \Xi_{S_u}[\tau]\hat{v}) + V_{\tau+1}(\widetilde{A}x[\tau] + \widetilde{B}\Xi_{S_u}[\tau]\hat{v})$$

$$u[\tau] = \Xi_{S_u}[\tau]v[\tau] = K_\tau(x[\tau]), \tag{9}$$

$$V_\tau(x[\tau]) = g_\tau(x[\tau], u[\tau]) + V_{\tau+1}(\widetilde{A}x[\tau] + \widetilde{B}u[\tau]), \tag{10}$$

for all $\tau = T, \dots, 1$ and $V_{T+1}(\cdot) = 0$.

Again, let $x[T+1] = 0 \in$ null (I) to enforce the boundary condition, we derive the admissible set $\mathcal{A}_v[\tau]$ and some condition for $x[\tau]$ from a corollary to Theorem 1 below.

Algorithm 2: DP for SLS with entrywise linear constraints.

Input: A, B, objective $g(\mathbf{\Phi_x}, \mathbf{\Phi_u})$, and entrywise linear constraints $\mathcal{S}$.

Output: $\Phi_x[\tau], \Phi_u[\tau]$ for all $\tau = 1, \dots, T$.

null $(\Psi_x[T+1]) =$ null (I).
$V_{T+1}(x[\tau]) = 0$.
for $\tau = T, \dots, 1$ **do**
  Derive $\Xi_{S_u}[\tau]$ from (8).
  Derive $\mathcal{A}_v[\tau]$ and null $(\Omega[\tau])$ from Corollary 1.
  Compute $K_\tau(x[\tau])$ by (9).
  Derive $V_\tau(x[\tau])$ from (10).
  Obtain null $(\Psi_x[\tau])$ by (11).
end for
$\Phi_x[1] = I$.
for $\tau = 1, \dots, T$ **do**
  $\overrightarrow{\Phi_u[\tau]} = K_\tau(\overrightarrow{\Phi_x[\tau]})$.
  $\Phi_x[\tau+1] = A\Phi_x[\tau] + B\Phi_u[\tau]$.
end for

Corollary 1. *Suppose $x[\tau+1] \in$ null $(\Psi_x[\tau+1])$ for some given matrix $\Psi_x[\tau+1]$, we have*

$$\mathcal{A}_v[\tau] = \{\hat{v} : \hat{v} = Q_x x[\tau] + Q_\lambda \lambda\}$$

where

$$\begin{aligned}\Gamma[\tau] &= \begin{bmatrix} -\widetilde{B}\Xi_{S_u}[\tau] & \Xi_x[\tau+1] \end{bmatrix}, \\ Q_x &= \begin{bmatrix} I & 0 \end{bmatrix} \Gamma^+[\tau]\widetilde{A}, \\ Q_\lambda &= \begin{bmatrix} I & 0 \end{bmatrix} (I - \Gamma^+[\tau]\Gamma[\tau]),\end{aligned}$$

and $\Xi_x[\tau+1]$ is given by Lemma 1 for null $(\Psi_x[\tau+1])$.

Also, $x[\tau] \in$ null $(\Omega[\tau])$ where

$$\Omega[\tau] = (\Gamma[\tau]\Gamma^+[\tau] - I)\widetilde{A}.$$

We omit the proof of Corollary 1, which is the same as the proof of Theorem 1.

We also need to enforce the condition (7a). We know

$$x[\tau] \in \mathcal{S}_x[\tau] = \{\Phi : \overrightarrow{\Phi} \in \text{null}\,(S_x[\tau] - I)\}.$$

Meanwhile, Corollary 1 suggests that $x[\tau] \in$ null $(\Omega[\tau])$. So by Lemma 2, we have $x[\tau] \in$ null $(\Psi_x[\tau])$ where

$$\text{null}\,(\Psi_x[\tau]) = \text{null}\left(\begin{bmatrix} S_x[\tau] - I \\ \Omega[\tau] \end{bmatrix}\right). \tag{11}$$

Together, we summarize the derivation in Algorithm 2.

III. Case Study: $\mathcal{H}_2$ Objective

In Section II, we derived DP algorithms for plain SLS, an approximation to infinite horizon SLS, and SLS with sparsity constraints. Here, we apply the algorithms to a specific objective function, the $\mathcal{H}_2$ objective, as an example. The $\mathcal{H}_2$ objective function is given by

$$g(\mathbf{\Phi_x}, \mathbf{\Phi_u}) = \|C\mathbf{\Phi_x} + D\mathbf{\Phi_u}\|_{\mathcal{H}_2}^2 = \sum_{\tau=1}^{T} g_\tau(\Phi_x[\tau], \Phi_u[\tau])$$

where

$$g_\tau(\Phi_x[\tau], \Phi_u[\tau]) = \|C\Phi_x[\tau] + D\Phi_u[\tau]\|_F^2$$

for some matrices C and D.

Plain SLS: We apply Algorithm 1 to plain SLS with $\mathcal{H}_2$ objective. The first step is to derive an explicit cost-to-go function $V_\tau(X[\tau])$, and we proceed by mathematical induction. We claim

$$V_\tau(X[\tau]) = \sum_{t=\tau}^{T} \|P_t[\tau]X[\tau]\|_F^2 \tag{12}$$

where $P_t[\tau]$ are some matrices.

For $\tau = T+1$, $V_{T+1}(X[\tau]) = 0$ satisfies the claim.

For $\tau < T+1$, (4) gives

$$\begin{aligned}V_\tau(X[\tau]) = \min_{\hat{U} \in \mathcal{A}_U[\tau]} \Bigg\{ &\left\| CX[\tau] + D\hat{U} \right\|_F^2 \\ &+ \sum_{t=\tau+1}^{T} \left\| P_t[\tau+1](AX[\tau] + B\hat{U}) \right\|_F^2 \Bigg\}.\end{aligned}$$

By Theorem 1, we have

$$\begin{aligned}V_\tau(X[\tau]) = \min_{\Lambda} \Bigg\{ &\|C_X X[\tau] + D_\Lambda \Lambda\|_F^2 \\ &+ \sum_{t=\tau+1}^{T} \|P_t[\tau+1](A_X X[\tau] + B_\Lambda \Lambda)\|_F^2 \Bigg\}\end{aligned} \tag{13}$$

where $A_X = A + BQ_X$, $B_\Lambda = BQ_\Lambda$, $C_X = C + DQ_X$, and $D_\Lambda = DQ_\Lambda$.

We can find the minimizer $\Lambda[\tau]$ to (13) by the first-order condition of its derivative:

$$\begin{aligned}&D_\Lambda^\top (C_X X[\tau] + D_\Lambda \Lambda[\tau]) \\ &+ B_\Lambda^\top \sum_{t=\tau+1}^{T} P_t[\tau+1]^\top P_t[\tau+1](A_X X[\tau] + B_\Lambda \Lambda[\tau]) = O\end{aligned}$$

where O is the all-zero matrix.

By defining

$$\begin{aligned}P[\tau] &= \sum_{t=\tau+1}^{T} P_t[\tau+1]^\top P_t[\tau+1], \\ L[\tau] &= -(D_\Lambda^\top D_\Lambda + B_\Lambda^\top P[\tau] B_\Lambda)^{-1}(D_\Lambda^\top C_X + B_\Lambda^\top P[\tau] A_X),\end{aligned} \tag{14}$$

we can derive

$$\begin{aligned}\Lambda[\tau] &= L[\tau]X[\tau] \\ U[\tau] &= Q_X X[\tau] + Q_\Lambda \Lambda[\tau] = (Q_X + Q_\Lambda L[\tau])X[\tau] \\ &= K_\tau(X[\tau]) = K[\tau]X[\tau]\end{aligned} \tag{15}$$

where we introduce the matrix $K[\tau]$ accordingly.

Consequently, the cost-to-go function is

$$\begin{aligned}V_\tau(X[\tau]) &= \|(C + DK[\tau])X[\tau]\|_F^2 \\ &+ \sum_{t=\tau+1}^{T} \|P_t[\tau+1](A + BK[\tau])X[\tau]\|_F^2,\end{aligned} \tag{16}$$

Algorithm 3: DP for plain SLS with $\mathcal{H}_2$ objective.

Input: A, B, C, D.
Output: $\Phi_x[\tau], \Phi_u[\tau]$ for all $\tau = 1, \dots, T$.

null $(\Psi_x[T+1]) =$ null (I).
$P = 0$.
for $\tau = T, \dots, 1$ **do**
  Derive $\mathcal{A}_U[\tau]$ and null $(\Psi_x[\tau])$ from Theorem 1.
  Compute $K[\tau]$ by (15).
  Update P by (17).
end for
$\Phi_x[1] = I$.
for $\tau = 1, \dots, T$ **do**
  $\Phi_u[\tau] = K[\tau]\Phi_x[\tau]$.
  $\Phi_x[\tau+1] = A\Phi_x[\tau] + B\Phi_u[\tau]$.
end for

of which the form matches our claim (12).

Although we need the cost-to-go function for derivation, we don't need it when deriving $U[\tau]$. As such, we only need to keep track of the quantity $P[\tau]$. Compare (12) and (16), we can update (14) by

$$P[\tau-1] = (C + DK[\tau])^\top (C + DK[\tau]) + (A + BK[\tau])^\top P[\tau](A + BK[\tau]). \quad (17)$$

In sum, we derive Algorithm 3 for plain SLS with $\mathcal{H}_2$ objective.

Approximation to Infinite Horizon SLS: We then extend the DP approximation to infinite horizon SLS to $\mathcal{H}_2$ objective. As discussed in Section II-B, we just need to relax the feasible set $\mathcal{A}_U[\tau]$ to be the whole space, and the cost-to-go function becomes

$$V_\tau(X[\tau]) = \min_{\hat{U}} \left\{ \left\| CX[\tau] + D\hat{U} \right\|_F^2 + \sum_{t=\tau+1}^{T} \left\| P_t[\tau+1](AX[\tau] + B\hat{U}) \right\|_F^2 \right\}.$$

Similarly, we obtain the first-order condition of the derivative:

$$D^\top (CX[\tau] + DU[\tau]) + B^\top P[\tau](AX[\tau] + BU[\tau]) = O$$

where $P[\tau]$ is defined in (14). Hence,

$$K[\tau] = -(D^\top D + B^\top P[\tau] B)^{-1}(D^\top C + B^\top P[\tau] A), \quad (18)$$
$$U[\tau] = K[\tau]X[\tau] = K_\tau(X[\tau])$$

and the corresponding DP Approx algorithm is summarized in Algorithm 4.

SLS with Entrywise Linear Constriants: Finally, we specialize Algorithm 2 for $\mathcal{H}_2$ objective. We first introduce $\widetilde{C}$ and $\widetilde{D}$ such that

$$C\Phi_x[\tau] = \widetilde{C}\overrightarrow{\Phi_x[\tau]}, \quad \text{and} \quad D\Phi_u[\tau] = \widetilde{D}\overrightarrow{\Phi_u[\tau]},$$

Algorithm 4: DP Approx: DP approximation for infinite horizon SLS with $\mathcal{H}_2$ objective.

Input: A, B, C, D.
Output: $\Phi_x[\tau], \Phi_u[\tau]$ for all $\tau = 1, \dots, T$.

$P = 0$.
for $\tau = T, \dots, 1$ **do**
  Compute $K[\tau]$ by (18).
  Update P by (17).
end for
$\Phi_x[1] = I$.
for $\tau = 1, \dots, T$ **do**
  $\Phi_u[\tau] = K[\tau]\Phi_x[\tau]$.
  $\Phi_x[\tau+1] = A\Phi_x[\tau] + B\Phi_u[\tau]$.
end for

for all τ. Likewise, we assume that

$$V_\tau(x[\tau]) = \sum_{t=\tau}^{T} \|P_t[\tau]x[\tau]\|_F^2 .$$

Following the similar procedure, we obtain $\Xi_{S_u}[\tau]$ from (8) and compute

$$\begin{aligned} L_n[\tau] &= D_\lambda[\tau]^\top C_x[\tau] + B_\lambda[\tau]^\top P[\tau] A_x[\tau], \\ L_d[\tau] &= D_\lambda[\tau]^\top D_\lambda[\tau] + B_\lambda[\tau]^\top P[\tau] B_\lambda[\tau], \\ L[\tau] &= -L_d[\tau]^{-1} L_n[\tau], \end{aligned}$$

where $P[\tau]$ is defined in (14) and

$$\begin{aligned} A_x[\tau] &= \widetilde{A} + \widetilde{B}\Xi_{S_u}[\tau]Q_x, \quad & B_\lambda[\tau] &= \widetilde{B}\Xi_{S_u}[\tau]Q_\lambda, \\ C_x[\tau] &= \widetilde{C} + \widetilde{D}\Xi_{S_u}[\tau]Q_x, \quad & D_\lambda[\tau] &= \widetilde{D}\Xi_{S_u}[\tau]Q_\lambda. \end{aligned}$$

Accordingly,

$$\begin{aligned} \lambda[\tau] &= L[\tau]x[\tau], \\ v[\tau] &= Q_x[\tau]x[\tau] + Q_\lambda[\tau]\lambda[\tau] = (Q_x[\tau] + Q_\lambda[\tau]L[\tau])x[\tau], \\ u[\tau] &= \Xi_{S_u}[\tau]v[\tau] = \Xi_{S_u}[\tau](Q_x[\tau] + Q_\lambda[\tau]L[\tau])x[\tau] \\ &= K_\tau(x[\tau]) = K[\tau]x[\tau] \end{aligned} \quad (19)$$

with $K[\tau]$ defined correspondingly.

As a result,

$$V_\tau(x[\tau]) = \left\| (\widetilde{C} + \widetilde{D}K[\tau])x[\tau] \right\|_F^2 + \sum_{t=\tau+1}^{T} \left\| P_t[\tau+1](\widetilde{A} + \widetilde{B}K[\tau])x[\tau] \right\|_F^2 ,$$

which confirms our assumption above. Therefore, we can update $P[\tau]$ by

$$P[\tau-1] = (\widetilde{C} + \widetilde{D}K[\tau])^\top (\widetilde{C} + \widetilde{D}K[\tau]) + (\widetilde{A} + \widetilde{B}K[\tau])^\top P[\tau](\widetilde{A} + \widetilde{B}K[\tau]) \quad (20)$$

and we summarize the derivation in Algorithm 5.

Remarks on Scalability We remark that dynamic programming methods could suffer the *curse of dimensionality*, i.e., the storage size grows super-linearly (or exponentially)

Algorithm 5: DP for SLS with $\mathcal{H}_2$ objective subject to entrywise linear constraints.

Input: A, B, C, D and $S_x[\tau], S_u[\tau]$
Output: $\Phi_x[\tau], \Phi_u[\tau]$ for all $\tau = 1, \dots, T$.

null $(\Psi_x[T+1]) =$ null (I).
$P = 0$.
for $\tau = T, \dots, 1$ **do**
  Derive $\Xi_{S_u}[\tau]$ from (8).
  Derive $\mathcal{A}_v[\tau]$ and null $(\Omega[\tau])$ from Corollary 1.
  Compute $K[\tau]$ by (19).
  Update P by (20).
  Obtain null $(\Psi_x[\tau])$ by (11).
end for
$\Phi_x[1] = I$.
for $\tau = 1, \dots, T$ **do**
  $\overrightarrow{\Phi_u[\tau]} = K[\tau]\overrightarrow{\Phi_x[\tau]}$.
  $\Phi_x[\tau+1] = A\Phi_x[\tau] + B\Phi_u[\tau]$.
end for

in the dimension of the variables, when maintaining the cost-to-go function V_τ. Here, for a quadratic objective such as $\mathcal{H}_2$, our DP algorithm maintains the cost-to-go function by keeping track of the matrix $K[\tau]$, whose size grows linearly. As a result, our algorithms scale well and we demonstrate their scalability in Section IV-B.

IV. Evaluation

We evaluate our algorithms through simulations. We first compare the scalability of DP against existing solver CVX [9] and naive Lagrange multiplier method, which also yields the analytical solution. We then simulate DP (Algorithm 3) and DP Approx (Algorithm 4) to evaluate the cost, in terms of computation overhead, of obtaining a finite impulse response (FIR) system under feedback. We begin with our simulation setup and a brief introduction of the Lagrange multiplier method.

A. Simulation Setup and Naive Lagrange Multiplier Method

We conduct the simulations using SLSpy [10], [11]. In each simulation, we synthesize controllers for 100 random systems and collect the statistical data. Each system is a fully actuated chain with N_x nodes, where the A matrix is tridiagonal with randomly generated off-diagonal entries, and B matrix is a diagonal matrix with random diagonal entries. The SLS objective is as follows

$$g(\mathbf{\Phi_x}, \mathbf{\Phi_u}) = \left\| \begin{bmatrix} I \\ 0 \end{bmatrix} \mathbf{\Phi_x} + \begin{bmatrix} 0 \\ I \end{bmatrix} \mathbf{\Phi_u} \right\|_{\mathcal{H}_2}^2 = \left\| \begin{bmatrix} \mathbf{\Phi_x} \\ \mathbf{\Phi_u} \end{bmatrix} \right\|_{\mathcal{H}_2}^2, \tag{21}$$

where C and D matrices are defined accordingly. We consider the d-locality constraint as in [12] with actuation delay 1, communication delay 2 and $d = 3$. As a subclass of the sparsity constraints, we can also express locality constraints as entrywise linear constraints. The results are measured on a desktop with AMD Ryzen 7 3700X processor (16 logical cores) and 32 GB DDR4 memory.

TABLE I
Average synthesis time of plain SLS for random chain-like systems.

N_x	Synthesis Time (ms)		
	DP (Algorithm 3)	CVX	Lagrange Multiplier
5	5.11	72.44	54.86
10	6.60	90.55	1176.51
15	8.42	136.76	9586.75
20	11.25	244.37	45782.15

TABLE II
Average synthesis time of SLS with locality constraints for random chain-like systems.

N_x	Synthesis Time (ms)		
	DP (Algorithm 5)	CVX	Lagrange Multiplier
5	12.35	217.81	479.60
10	129.48	1411.00	78405.76
15	685.03	4890.28	1013700.11
20	1968.85	8549.75	not feasible

Since the SLS constraints (2) and the locality constraints (in the form of (7)) are all equalities, we can rewrite the SLS problem as

$$\min\ g(\mathbf{\Phi_x}, \mathbf{\Phi_u})\ \text{ s.t. } h(\mathbf{\Phi_x}, \mathbf{\Phi_u}) = 0$$

and apply the naive Lagrange multiplier method to solve

$$\nabla_{\mathbf{\Phi_x}, \mathbf{\Phi_u}, \lambda}\ g(\mathbf{\Phi_x}, \mathbf{\Phi_u}) - \lambda h(\mathbf{\Phi_x}, \mathbf{\Phi_u}) = 0.$$

Given the objective (21), we express the above condition as

$$J\overrightarrow{\Phi} - b = 0$$

for some matrix J and vector b, where $\overrightarrow{\Phi}$ is a vector of the entries in $\mathbf{\Phi_x}$ and $\mathbf{\Phi_u}$, and compute $\overrightarrow{\Phi}$ by $J^{-1}b$.

B. Scalability with System Size

To evaluate the scalability of the methods, we run the simulations with different system size N_x, measure the average synthesis time for the plain SLS and SLS with locality constraints, and summarize the results in Table I and Table II, respectively. Among the methods, DP scales the best. For plain SLS, DP is 12 to 22× faster than CVX and 10 to 4000× faster than naive Lagrange multiplier method; With locality constraints, DP is 4 to 17× faster than CVX and more than 38× faster than naive Lagrange multiplier method, which cannot even deal with $N_x = 20$. We remark that DP outperformed two other methods using only one CPU core without any optimization, while CVX parallelized its work over 16 logical cores. It is possible to improve the performance of DP by parallelizing its computation.

TABLE III

AVERAGE SYNTHESIS TIME OF PLAIN SLS FOR RANDOM CHAIN-LIKE SYSTEMS UNDER RANDOM OBJECTIVES.

N_x	Synthesis Time (ms) / Solvable Rate	
	DP (Algorithm 3)	CVX
5	3.35 / 100%	44.20 / 100%
10	4.18 / 100%	92.33 / 100%
15	5.09 / 100%	240.10 / 100%
20	6.81 / 100%	535.77 / 100%

TABLE IV

AVERAGE SYNTHESIS TIME OF SLS WITH LOCALITY CONSTRAINTS FOR RANDOM CHAIN-LIKE SYSTEMS UNDER RANDOM OBJECTIVES.

N_x	Synthesis Time (ms) / Solvable Rate	
	DP (Algorithm 3)	CVX
5	7.07 / 100%	104.21 / 100%
10	120.89 / 100%	664.09 / 91%
15	590.84 / 100%	1989.17 / 85%
20	1498.32 / 100%	4051.55 / 82%

In addition to the simple objective in (21), we also compare DP with CVX under random objectives

$$g(\mathbf{\Phi_x}, \mathbf{\Phi_u}) = \|C\mathbf{\Phi_x} + D\mathbf{\Phi_u}\|_{\mathcal{H}_2}^2$$

where C and D are both N_x by N_x matrices with elements generated from standard normal distribution. For each random system, we generate a random objective as above and solve the plain SLS and SLS with locality constraints. The results are summarized in Table III and Table IV. Besides the synthesis time, we also calculate the solvable rate, which is the ratio between the total number of solvable cases and the total number of cases. The synthesis time is computed according to the cases that are solvable for both methods. As shown in the tables, DP is always able to produce the optimal solutions while CVX fails to find the solution when we incorporate the locality constraints. Also, similar to the results in Table I and Table II, DP is 2.7 to 78.7× faster than CVX (on solvable cases).

C. Cost for FIR System

The boundary constraint is essential for the synthesized controller to stabilize a system in a finite horizon (FIR system). When the desired horizon goes to infinity, the controllers subject to the boundary condition become the ones without. Below, we examine the computation overhead for an FIR system and evaluate how close the DP controller (by Algorithm 3) is to the DP Approx controller (by Algorithm 4), which is an approximation to infinite horizon SLS.

Fig. 1 shows the computation overhead in terms of synthesis time versus the system size (N_x) and the horizon of the synthesized controllers. DP Approx is about 3 to 4× faster than DP, and both of them scales linearly with the FIR horizon as expected. In exchange, Fig. 2 shows that the DP Approx controller fails to stabilize the system within the desired FIR horizon after an impulse noise hits the center

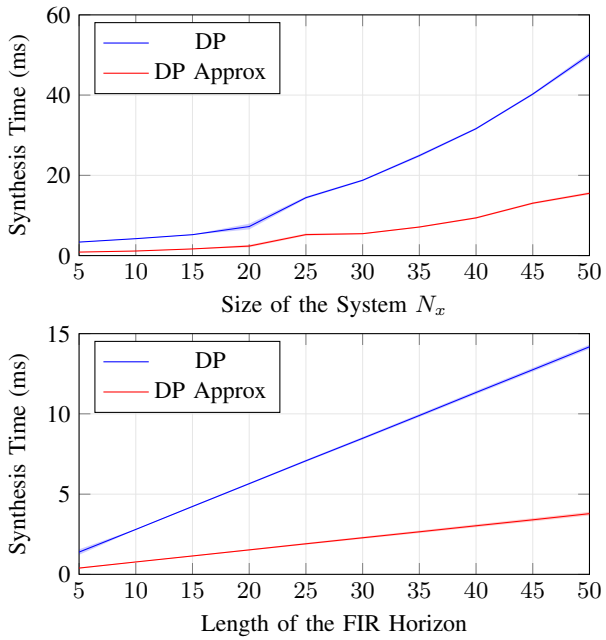

Fig. 1. Synthesis time of the controller by DP (Algorithm 3) and the infinite horizon approximation controller by DP Approx (Algorithm 4).

of the system. When we allow a longer FIR horizon, DP Approx controller performs the same as the DP controller. In sum, we pay some tens of milliseconds more to guarantee a controller stabilizing a system in a finite horizon.

V. CONCLUSION AND FUTURE DIRECTIONS

We derived DP algorithms to solve general state-feedback SLS problems, including plain SLS, infinite horizon approximation, and sparsity constrained SLS. Sparsity constraints generalize locality constraints by allowing linear dependency among entries of spectral components. Using $\mathcal{H}_2$ objective as an example, we demonstrate how to adapt DP algorithms to a given objective. Our simulation results suggest that DP significantly outperforms CVX and naive Lagrange multiplier method. We also quantify the computation overhead of obtaining a controller for an FIR system after feedback.

Future work includes extensions of the DP algorithms for output-feedback SLS, which contains more parameters to handle. Also, it is worth covering more constraint classes, such as inequality constraints or dependencies among entries from different spectral components. Finally, one can apply DP to online synthesis settings such as model predictive control, where the computational overhead is crucial.

REFERENCES

[1] J. Anderson *et al.*, "System level synthesis," *Annual Reviews in Control*, vol. 59, no. 12, pp. 3238–3251, 2019.

[2] Y.-S. Wang, N. Matni, and J. C. Doyle, "A system level approach to controller synthesis," *IEEE Trans. Autom. Control*, vol. 34, no. 8, pp. 982–987, 2019.

[3] C. Amo Alonso and N. Matni, "Distributed and localized model predictive control via system level synthesis," *arXiv preprint arXiv:1909.10074*, 2019.

[4] J. Anderson and N. Matni, "Structured state space realizations for SLS distributed controllers," in *Proc. Allerton*, 2017, pp. 982–987.

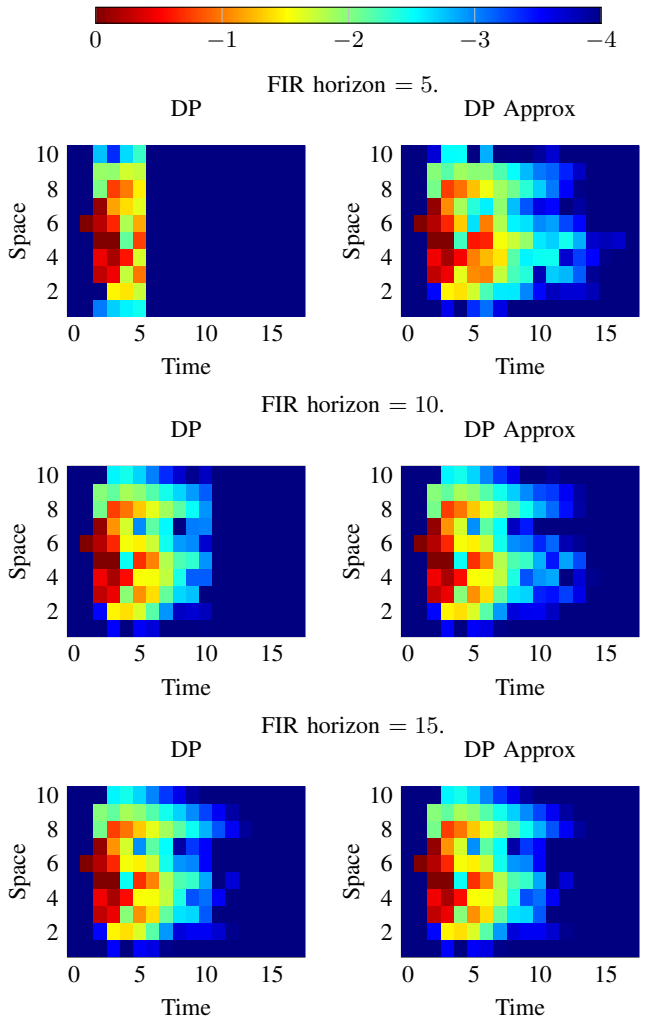

Fig. 2. Comparison of controllers computed with and without boundary constraints for different time horizons. On the left, the controllers are computed using DP, i.e., accounting for boundary constraints. On the right, the controllers are computed using DP Approx, i.e., no boundary constraints are imposed. Each row corresponds to a different FIR horizon. The figures show the space-time diagram of the state (in log scale, $\log_{10}(|x|)$) after some noise hits the center of the chain at time 0. The DP Approx controller approximates finite horizon DP controller when the horizon gets longer.

[5] N. Matni, Y.-S. Wang, and J. Anderson, "Scalable system level synthesis for virtually localizable systems," in *Proc. IEEE CDC*. IEEE, 2017, pp. 3473–3480.

[6] Y.-S. Wang, N. Matni, and J. C. Doyle, "Separable and localized system-level synthesis for large-scale systems," *IEEE Trans. Autom. Control*, vol. 63, no. 12, pp. 4234–4249, 2018.

[7] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 2005, vol. 1.

[8] M. James, "The generalised inverse," *The Mathematical Gazette*, vol. 62, no. 420, pp. 109–114, 1978.

[9] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming," http://cvxr.com/cvx, Mar. 2014.

[10] S.-H. Tseng and J. S. Li, "SLSpy: Python-based system-level controller synthesis framework," *arXiv preprint arXiv:2004.12565*, 2020.

[11] SLSpy. [Online]. Available: https://github.com/shih-hao-tseng/SLSpy

[12] Y.-S. Wang and N. Matni, "Localized LQG optimal control for large-scale systems," in *Proc. IEEE ACC*, 2016.