cuADMM: GPU-Accelerated First-Order Optimization for Large-Scale Multi-Block Semidefinite Programs

Antoine Groudiev*†, Shucheng Kang*, Heng Yang*
*School of Engineering and Applied Sciences, Harvard University
†Computer Science Department, École Normale Supérieure, PSL University

***Abstract*—We present cuADMM, a GPU-accelerated implementation of the Alternating Direction Method of Multipliers (ADMM) for solving large-scale semidefinite programs (SDPs). cuADMM is designed to efficiently handle multiple blocks of varying sizes, allowing its application to a wide range of problems, including contact-rich trajectory optimization. We demonstrate the performance of cuADMM on two types of problems: (1) sparse SDPs generated from trajectory optimization problems, and (2) generic multi-block SDPs. Our results show that cuADMM outperforms existing solvers, such as MOSEK and ADMM+, particularly for large or many blocks, making it a valuable tool for efficiently solving large-scale SDP problems.**

I. Introduction

Motion planning plays a crucial role in robotics, especially for contact-rich tasks.

A. Problem statement

Let $N \in \mathbb{N}$ be the planning horizon, $\{x_k\}_{k=0}^{N} \subset \mathbb{R}^{n_x}$ the state trajectory, and $\{u_k\}_{k=0}^{N-1} \subset \mathbb{R}^{n_u}$ the control inputs. We want to find the trajectory that minimizes a certain cost function over the states, control inputs, as well as *contact variables* $\{\lambda_k\}_{k=0}^{N-1} \subset \mathbb{R}^{n_\lambda}$, which are used to model the contact forces. Formally, we want to solve the following optimization problem:

$$\min_{\{x_k\}_{k=0}^{N},\{u_k\}_{k=0}^{N-1},\{\lambda_k\}_{k=0}^{N-1}} \ell_N(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k, \lambda_k) \quad (1)$$

$$\text{s.t.} \quad \begin{cases} x_0 = x_{\text{init}} \\ F_k(x_{k-1}, u_{k-1}, \lambda_{k-1}, x_k) = 0, \quad k \in [N] \\ (u_{k-1}, \lambda_{k-1}, x_k) \in \mathcal{C}_k, \quad k \in [N] \end{cases} \quad (2)$$

where ℓ_k for $k < N$ are the instantaneous costs, ℓ_N is the terminal cost, F_k are the discretized dynamics constraints, and $\mathcal{C}_k$ represent other types of constraints, such as control limits, collision avoidance, etc.

B. Sparse Moment-SOS Hierarchy

Assuming that ℓ_k and F_k are polynomial functions and that $\mathcal{C}_k$ is basic semi-algebraic (that is, described by polynomial constraints), (1) becomes a Polynomial Optimization Problem (POP). Such problems can be relaxed as convex problems, using the Moment-SOS hierarchy [6], which generates a sequence of convex semidefinite programs (SDPs) that provide lower bounds converging to the optimal value of the original POP.

Furthermore, multiple sparsity patterns can be exploited in the POPs, including correlative sparsity [7], term sparsity [13], as well as robotics-specific sparsity from kinematics chains and separable contact modes [5]. Packages such as TSSOS [8] in Julia or SPOT in C++ [5] automatically generate SDPs leveraging these sparsity patterns. However, the resulting SDPs can be very large, and solving them is often a bottleneck in the planning process.

Due to the rich sparsity structure present in large-scale POPs and effectively exploited by the sparse Moment-SOS hierarchy, X can be partitioned into blocks, where each block corresponds to a single symmetric variable.

An approximate solution to the original POP can therefore be obtained by solving a multi-block SDP. Formally, if we denote by Ω the Cartesian product of the symmetric blocks, and $\Omega_+ \subset \Omega$ the subset of Ω for which all the blocks are PSD, the multi-block SDP can be written as:

$$\min_X \langle C, X \rangle \quad \text{s.t.} \quad \begin{cases} \langle A_i, X \rangle = b_i, \quad i \in [m] \\ X \in \Omega_+ \end{cases} \quad (3)$$

for some constraints matrices $A_i \in \Omega$, a constraint vector $b \in \mathbb{R}^m$ and a cost matrix $C \in \Omega$. The blocks of X are denoted by $X_b \in \mathbb{R}^{n_b \times n_b}$ for $b \in [B]$, where n_b is the size of the b-th block, and B the number of blocks. The vectorization of X is denoted by $\text{svec}(X) \in \mathbb{R}^n$, where $n = \sum_{b=1}^{B} \frac{n_b(n_b+1)}{2}$, and is obtained by stacking the upper triangular part of each block X_b in a vector, where non-diagonal elements are multiplied by $\sqrt{2}$.

C. Contributions

In this paper, we present cuADMM, a GPU-accelerated generic first-order solver for large-scale multi-block semidefinite programs, designed to efficiently solve the SDPs generated by the Moment-SOS hierarchy. cuADMM is based on the Alternating Direction Method of Multipliers (ADMM) algorithm, which is well-suited for large-scale optimization problems [10]. While interior-point methods, implemented in solvers such as SDPT3 [12] and MOSEK [1], are widely used for solving SDPs, they can take hours and run out of memory for large problems. In contrast, cuADMM is designed to be fast and memory-efficient, and leverages the parallel computing capabilities of GPUs to accelerate the optimization process.

II. Algorithm

A. sGS-ADMM

cuADMM uses the sGS-ADMM variant of the ADMM algorithm [2]. Considering the multi-block SDP (3), its La-

grangian dual reads:

$$\max_{y\in\mathbb{R}^m, S\in\Omega} \langle b, y\rangle \quad \text{s.t.} \quad \begin{cases} A^\mathsf{T} y + S = C \\ S \in \Omega_+ \end{cases}. \tag{4}$$

Denoting Π_{Ω_+} the projection operator on the PSD cone, the sGS-ADMM can be summarized in Algorithm 1. Note that along with the traditional ADMM steps (steps 1, 2, and 4), an additional step specific to sGS-ADMM (step 3) is performed. While this allows for faster convergence for easier problems, it can cause convergence issues for harder ones. In practice, if the algorithm has not converged after a fixed number of iterations, we switch back to the standard ADMM algorithm by skipping step 3.

Algorithm 1: sGS-ADMM for solving (4).

Input: Initial points $X^0, S^0 \in \Omega$, $\tau \in (0, 2)$, $\sigma > 0$.
For $k = 0, 1, 2, \ldots$:

Step 1. Compute

$$r_s^{k+\frac{1}{2}} := \frac{1}{\sigma} b - A\left(\frac{1}{\sigma} X^k + S^k - C\right), \tag{5}$$

$$y^{k+\frac{1}{2}} = \left(AA^\mathsf{T}\right)^{-1} r_s^{k+\frac{1}{2}} \tag{6}$$

Step 2. Compute

$$X_b^{k+1} := X^k + \sigma(A^\mathsf{T} y^{k+\frac{1}{2}} - C), \tag{7}$$

$$S_b^{k+1} = \frac{1}{\sigma}\left(\Pi_{\Omega_+}\left(X_b^{k+1}\right) - X_b^{k+1}\right) \tag{8}$$

Step 3. Compute

$$r_s^{k+1} := \frac{1}{\sigma} b - A\left(\frac{1}{\sigma} X^k + S^{k+1} - C\right), \tag{9}$$

$$y^{k+1} = \left(AA^\mathsf{T}\right)^{-1} r_s^{k+1} \tag{10}$$

Step 4. Compute

$$X^{k+1} = X^k + \tau\sigma\left(S^{k+1} + A^\mathsf{T} y^{k+1} - C\right) \tag{11}$$

Until terminal conditions hold.
Output: $\left(X^{k+1}, y^{k+1}, S^{k+1}\right)$.

B. Termination conditions

We use the standard maximum Karush-Kuhn-Tucker (KKT) residual as a condition for termination. We define the primal and dual residuals η_p and η_d as:

$$\eta_p = \frac{\|AX - b\|_2}{1 + \|b\|_2}, \quad \eta_d = \frac{\left\|A^\mathsf{T} y + S - C\right\|_2}{1 + \|C\|_2}, \tag{12}$$

and the relative duality gap η_g as:

$$\eta_g = \frac{|\langle C, X\rangle - \langle b, y\rangle|}{1 + |\langle C, X\rangle| + |\langle b, y\rangle|}. \tag{13}$$

If the maximum KKT residual $\eta = \max(\eta_p, \eta_d, \eta_g)$ is smaller than a certain threshold, the algorithm is considered to have converged. In practice, first-order methods can only achieve moderate accuracy (up to $\eta = 10^{-3}$ or 10^{-5}), while interior-point methods can solve the problem to much higher accuracy. We show in the experiments that this is sufficient to obtain good solutions for the trajectory optimization problems.

III. Implementation

We implement Algorithm 1 on GPU using C++, CUDA, and the `cuSOLVER` and `cuBLAS` libraries, using double precision (FP64). Unlike previous work [4], we do not restrict ourselves to only two blocks sizes, but allow for any number of blocks of any size. An illustration of the GPU implementation is shown in Figure 1.

A. PSD cones projection

The projection of X on the Cartesian product of multiple PSD cones is done by performing an eigenvalue decomposition of each block in X, and putting to zero the negative eigenvalues.

To do so, the primal variable X in svec form is converted to sequences of matrices $\{M_i\}$ grouped by matrix size, using fast parallel mappings. Each block is then decomposed as $M_i = Q_i W_i Q_i^\mathsf{T}$ using the `cuSOLVER` library. We then obtain the projected block $\Pi_{\mathbb{S}^{|M_i|}}(M_i) = Q_i \max(0, W_i) Q_i^\mathsf{T}$ by performing the matrix multiplications in parallel. Finally, we convert the projected blocks back to svec form and concatenate them to obtain the projected primal variable $\Pi_{\Omega_+}(X)$.

`cuSOLVER` offers two methods for computing the eigenvalue decomposition: a batched interface using the Jacobi method, and a single matrix interface using the QR method, which can be parallelized using multiple CUDA streams. While the former is more efficient for many small matrices of the same size, the latter is faster for larger matrices. Therefore, we use a heuristic to choose, for each block, the method that minimizes the time to compute the eigenvalue decomposition, depending on the block size and the number of blocks.

B. Linear system solving

Solving the linear system $AA^\mathsf{T} y = r_s$ is done using a hybrid CPU-GPU approach similar to Kang et al. [4]. After precomputing the Cholesky factorization $AA\mathsf{T} = PLDL^\mathsf{T} P^\mathsf{T}$, the permutation matrix multiplications are handled on the GPU, while LDL^T system solving is done on CPU using `CHOLMOD` [3].

IV. Experiments

In this section, we evaluate the performance of cuADMM on two types of problems: (1) sparse semidefinite programs (SDPs) generated from contact-rich trajectory optimization problems (IV-B), and (2) generic multi-block SDPs (IV-C).

A. Setup

We compare the performance of three solvers: MOSEK [1], ADMM+ [11], and cuADMM. Default cuADMM uses the standard ADMM algorithm, while cuADMM (sGS) uses the semi-implicit Gauss-Seidel method.

To help the reader estimate the size of the SDP instances, we present in the Appendix some properties of the problems. For the two types of problems, we report the running time of the solvers, the maximum KKT residual η, as well as the distance of the primal objective to the optimal value, defined as $\left|\frac{p-p^*}{1+|p|+|p^*|}\right|$, where p is the primal objective value, d is the dual objective value, and p^* is the primal optimal value found

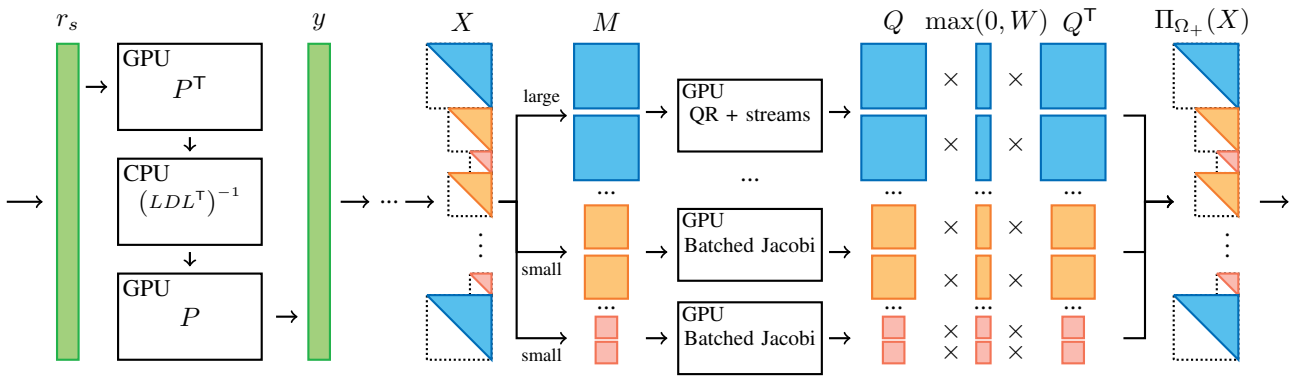

Fig. 1. Illustration of the GPU implementation for a problem with three block sizes (one large and two small). On the left, we use a hybrid CPU-GPU approach for solving the linear system $AA^\mathsf{T}y = r_s$. On the right, we use the eigenvalue decomposition to project the blocks of X on the PSD cones.

by MOSEK after convergence. Solvers are stopped when η is less than 10^{-3} and distance of the primal objective to the optimal value is less than 2×10^{-3}, or after 10 hours of computation time. For large problems where MOSEK takes more than 10 hours to solve up to 10^{-8} precision, we drop the primal distance metric condition.

Experiments were conducted on a high-performance workstation equipped with a 2.7 GHz AMD 64-Core sWRX8 Processor and 1 TB of RAM, and an NVIDIA RTX 6000 Ada Generation.

B. SPOT SDPs for motion planning

We start by evaluating the performance of CUADMM on SDP instances generated from contact-rich trajectory optimization problems by the SPOT library [5]. To study the impact of the problem size on the comparative performance of the solvers, we consider two sets of problems: one with short planning horizons ($N = 1$) and one with long planning horizons (N between 10 and 30).

The properties of the problems are shown in Tables IV and V of the Appendix. We report SPOT-specific parameters used to generate the trajectory optimization problem. N is the planning horizon, "Term sparsity" indicates which algorithm is used to compute the term sparsity graph [4], and the vector length denotes the length of the primal vector X in svec form. Non-specified parameters are set to default values of SPOT.

For both short and long planning horizon, MOSEK outperforms CUADMM on easy problems (i.e., Push Box and Push T), while CUADMM scales much better on hard problems. ADMM+ is faster than CUADMM on small problems, but is outperformed by other methods when the number of blocks increases (see for instance Push T).

While we only solve the problems up to medium precision ($\eta < 10^{-3}$), the visualization of the trajectories (Figures 2 and 3) shows that the trajectories found with low precision are very close to the ones found with high precision.

C. Generic multi-block SDPs

To demonstrate the effectiveness of CUADMM beyond trajectory optimization problems, we compare the performance of the same solvers on multi-block SDP instances collected by Mittelmann [9]. The properties of the problems are shown in Table VI, and the results in Table III.

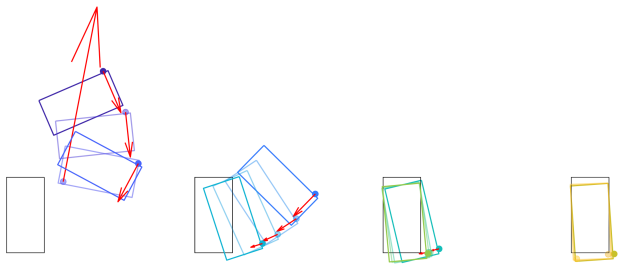

Fig. 2. Push Box trajectory solved with MOSEK up to precision $\eta < 10^{-8}$.

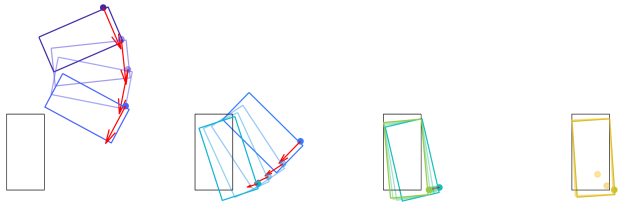

Fig. 3. Push Box trajectory solved with CUADMM up to $\eta = 9.97 \times 10^{-4}$.

While interior-point methods are much faster than first-order methods for problems exhibiting blocks of small size (as seen for the `ros_2000` problem), GPU-accelerated first-order methods outperform interior-point methods for problem with large or many blocks, such as `chs_500` and `tahala`, despite the CPU-GPU transfer overhead. Furthermore, CUADMM is consistently faster than ADMM+ on all instances, the speed improvement being more pronounced for larger problems.

V. CONCLUSION

We presented CUADMM, a GPU-based first-order solver for large-scale multi-block semidefinite programs, designed to efficiently solve the SDPs generated by the Moment-SOS hierarchy. CUADMM is based on the Alternating Direction Method of Multipliers (ADMM) algorithm, along with an implementation of the symmetric Gauss-Seidel variant, making it more scalable than interior-point methods. Our experimental results show that CUADMM outperforms existing solvers for large-scale SDPs, especially those arising from contact-rich trajectory optimization problems, while still being applicable to generic multi-block SDPs.

TABLE I
SOLVER RESULTS FOR THE SPOT PROBLEMS WITH **SHORT** PLANNING HORIZONS.

Problem	Solver	Time (s)	Max. KKT residual η	Primal distance
Push Box	MOSEK	$\mathbf{1.40}$	9.87×10^{-6}	7.21×10^{-4}
	ADMM+	3.5	4.06×10^{-4}	7.40×10^{-6}
	CUADMM	6.2	8.31×10^{-4}	7.40×10^{-6}
	CUADMM (sGS)	6.2	2.28×10^{-4}	3.33×10^{-5}
Push T	MOSEK	$\mathbf{0.1}$	2.65×10^{-6}	1.62×10^{-4}
	ADMM+	7.6	5.03×10^{-4}	8.71×10^{-5}
	CUADMM	0.5	1.00×10^{-3}	1.89×10^{-3}
	CUADMM (sGS)	0.5	1.00×10^{-3}	1.89×10^{-3}
Push Bot	MOSEK	32.5	1.23×10^{-6}	3.17×10^{-4}
	ADMM+	1.9	4.02×10^{-4}	9.98×10^{-5}
	CUADMM	$\mathbf{1.0}$	2.48×10^{-4}	1.34×10^{-4}
	CUADMM (sGS)	1.2	7.15×10^{-4}	9.98×10^{-5}
Planar Hand	MOSEK	405.5	2.32×10^{-4}	1.38×10^{-3}
	ADMM+	$\mathbf{32.2}$	4.87×10^{-4}	3.26×10^{-4}
	CUADMM	54.2	9.98×10^{-4}	2.42×10^{-7}
	CUADMM (sGS)	77.0	9.99×10^{-4}	1.36×10^{-7}
Tunnel	MOSEK	$3,764.8$	1.12×10^{-6}	4.82×10^{-4}
	ADMM+	$\mathbf{339.8}$	3.14×10^{-4}	9.81×10^{-4}
	CUADMM	471.8	6.14×10^{-4}	1.57×10^{-3}
	CUADMM (sGS)	793.1	7.96×10^{-4}	1.57×10^{-3}

TABLE II
SOLVER RESULTS FOR THE SPOT PROBLEMS WITH **LONG** PLANNING HORIZONS. "—" INDICATES A TIMEOUT AFTER 10 HOURS.

Problem	Solver	Time (s)	Max. KKT residual η	Primal distance
Push Box	MOSEK	$\mathbf{213.4}$	8.47×10^{-7}	9.20×10^{-4}
	ADMM+[1]	$1,649$	2.20×10^{-3}	1.85×10^{-4}
	CUADMM	905.1	9.98×10^{-4}	2.96×10^{-4}
	CUADMM (sGS)	278.0	9.97×10^{-4}	4.97×10^{-4}
Push T	MOSEK	$\mathbf{36.6}$	2.44×10^{-7}	8.3×10^{-4}
	ADMM+	—	—	—
	CUADMM	142.0	1.61×10^{-5}	9.03×10^{-4}
	CUADMM (sGS)	186.6	8.24×10^{-4}	8.99×10^{-4}
Planar Hand	MOSEK	$12,017$	1.91×10^{-6}	—
	ADMM+	—	—	—
	CUADMM	—	—	—
	CUADMM (sGS)	$\mathbf{1,281}$	9.95×10^{-4}	—
Tunnel	MOSEK	$32,593$	7.09×10^{-6}	—
	ADMM+	—	—	—
	CUADMM	—	—	—
	CUADMM (sGS)	$\mathbf{2,491}$	9.97×10^{-4}	—

[1]ADMM+ does not support the relative gap as a stopping criterion, so we report the maximum KKT residual η at the end of the run. This is the only run where the relative gap condition was not satisfied.

TABLE III
SOLVER RESULTS FOR THE GENERIC MULTI-BLOCK SDPS.

Problem	Solver	Time (s)	Duality gap	Primal distance
chs_500	MOSEK	3.88	3.68×10^{-4}	6.96×10^{-5}
	ADMM+	91.5	1.73×10^{-4}	6.06×10^{-10}
	CUADMM	$\mathbf{0.2}$	6.22×10^{-4}	1.01×10^{-9}
	CUADMM (sGS)	0.7	8.36×10^{-4}	9.88×10^{-10}
ros_2000	MOSEK	$\mathbf{0.59}$	2.16×10^{-4}	6.93×10^{-4}
	ADMM+	257.8	6.29×10^{-4}	5.50×10^{-4}
	CUADMM	3.4	9.97×10^{-4}	1.02×10^{-3}
	CUADMM (sGS)	16.6	9.72×10^{-4}	9.49×10^{-4}
taha1a	MOSEK	3.49	1.22×10^{-4}	2.50×10^{-4}
	ADMM+	3.8	5.94×10^{-4}	1.67×10^{-4}
	CUADMM	2.3	5.47×10^{-4}	7.33×10^{-4}
	CUADMM (sGS)	$\mathbf{2.2}$	9.64×10^{-4}	6.17×10^{-4}

REFERENCES

[1] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. URL http://docs.mosek.com/9.0/toolbox/index.html.

[2] Liang Chen, Defeng Sun, and Kim-Chuan Toh. An Efficient Inexact Symmetric Gauss-Seidel Based Majorized ADMM for High-Dimensional Convex Composite Conic Programming. *Mathematical Programming*, 161(1–2): 237–270, April 2016.

[3] Timothy A Davis and W Hager. CHOLMOD: Supernodal Sparse Cholesky Factorization and Update/Downdate, 2005.

[4] Shucheng Kang, Xiaoyang Xu, Jay Sarva, Ling Liang, and Heng Yang. Fast and Certifiable Trajectory Optimization, 2024. URL https://arxiv.org/abs/2406.05846.

[5] Shucheng Kang, Guorui Liu, and Heng Yang. Global Contact-Rich Planning with Sparsity-Rich Semidefinite Relaxations, 2025. URL https://arxiv.org/abs/2502.02829.

[6] Jean B Lasserre. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

[7] Jean B Lasserre. Convergent SDP-Relaxations in Polynomial Optimization with Sparsity. *SIAM Journal on optimization*, 17(3):822–843, 2006.

[8] Victor Magron and Jie Wang. TSSOS: a Julia Library to Exploit Sparsity for Large-Scale Polynomial Pptimization, 2021. URL https://arxiv.org/abs/2103.00915.

[9] Hans D. Mittelmann. Several SDP-Codes on Sparse and other SDP Problems, 2006. URL https://plato.asu.edu/ftp/sparse_sdp.html.

[10] Michel Schubiger, Goran Banjac, and John Lygeros. GPU Acceleration of ADMM for Large-Scale Quadratic Programming. *Journal of Parallel and Distributed Computing*, 144:55–67, October 2020.

[11] Defeng Sun, Kim-Chuan Toh, Yancheng Yuan, and Xin-Yuan Zhao. SDPNAL+: A Matlab Software for Semidefinite Programming with Bound Constraints (version 1.0), 2019. URL https://arxiv.org/abs/1710.10604.

[12] Kim-Chuan Toh, Michael J Todd, and Reha H Tütüncü. SDPT3—a MATLAB Software Package for Semidefinite Programming, version 1.3. *Optimization methods and software*, 11(1-4):545–581, 1999.

[13] Jie Wang, Victor Magron, and Jean-Bernard Lasserre. TSSOS: A Moment-SOS Hierarchy that Exploits Term Sparsity. *SIAM Journal on optimization*, 31(1):30–58, 2021.

APPENDIX

TABLE IV
PROPERTIES OF THE SPOT PROBLEMS WITH **SHORT** PLANNING HORIZONS.

Problem	N	Step (dt)	Term sparsity	Blocks	Largest block	Vector length	Constraints	Non-zeros in A	Non-zeros in A (%)
Push Box	1	0.05	MF	37	45	3,813	4,113	9,985	6.36×10^{-4}
Push T	1	0.01	MF	290	13	1,834	997	2,806	1.53×10^{-3}
Push Bot	1	0.05	MF	40	66	8,433	11,434	26,194	2.72×10^{-4}
Planar Hand	1	0.05	MF	122	120	55,179	66,008	156,635	4.30×10^{-5}
Tunnel	1	0.1	NON	42	165	93,500	105,338	280,092	2.84×10^{-5}

TABLE V
PROPERTIES OF THE SPOT PROBLEMS WITH **LONG** PLANNING HORIZONS.

Problem	N	Term sparsity	Blocks	Largest block	Vector length	Constraints	Non-zeros in A	Non-zeros in A (%)
Push Box	30	MF	1,316	45	170,251	154,256	374,258	1.43×10^{-5}
Push T	30	MF	18,875	13	86,315	53,290	143,908	3.13×10^{-5}
Planar Hand	10	MF	1,553	120	533,718	483,707	1,310,651	8.42×10^{-7}
Tunnel	10	NON	420	165	821,060	820,271	2,617,338	3.82×10^{-7}

TABLE VI
PROPERTIES OF THE GENERIC MULTI-BLOCK SDPS.

Problem	Blocks	Largest block	Vector length	Constraints	Non-zeros in A	Non-zeros in A (%)
`chs_500`	4,998	10	274,890	99,974	269,892	9.82×10^{-6}
`ros_2000`	1,999	6	41,968	19,988	39,969	4.76×10^{-5}
`taha1a`	14	252	116,676	3,002	177,420	5.07×10^{-4}