Low-Rank Augmented Lagrangian Methods for SDPs without Trace Bounds

Jacob M. Aguirre

School of ISyE, Georgia Tech

November 7, 2026

INFORMS Annual Meeting, San Francisco, CA (15 min)

SDP Formulation handled by HALLaR

- HALLaR is a low-rank AL method which handles

$$\min_{X \in \mathbb{S}^n} \quad \{ \langle C, X \rangle \ : \ \mathcal{A}(X) = b, \ \operatorname{tr} X \leq R, \ X \succeq 0 \} \tag{P}$$

  where $R > 0$, $b \in \mathbb{R}^m$, $C \in \mathbb{S}^n$, and $\mathcal{A} : \mathbb{S}^n \to \mathbb{R}^m$ is a linear map

- HALLaR+ is a new variant of HALLaR that handles problems without the trace bounding constraint

SDP formulation handled by HALLaR+ is

$$\min_{X \in \mathbb{S}^n} \quad \{ \langle C, X \rangle \ : \ \mathcal{A}X = b, \quad X \succeq 0 \} \tag{P}$$

whose **dual** is

$$\max_{p \in \mathbb{R}^m} \quad \{ \langle b, p \rangle : S := C - \mathcal{A}^* p \succeq 0 \} \tag{D}$$

Assumption: the optimal solution set of (P) is nonempty and bounded, and there exists a KKT pair (X_*, p_*) for (P) and (D), i.e.,

$$\mathcal{A}X_* = b, \quad X_* \succeq 0, \quad S_* := C - \mathcal{A}^* p_* \succeq 0, \quad \langle S_*, X_* \rangle = 0$$

Computational difficulties in solving huge SDPs

- Projecting (usually dense) matrices onto $\mathbb{S}^n_+$ or Δ^n (which many projection-type methods do) is expensive since it requires full eigen-decomposition. Instead, HALLaR+ computes min eigen-pairs of structured matrices (e.g., sparse ones)
- Projecting onto $\{X : \mathcal{A}X = b\}$ (which a few methods do) is costly. Being a primal-dual method, HALLaR+ never performs this projection
- Storing the $n \times n$ matrix variable X requires huge amount of memory
  - a laptop with 32 GB of memory can only store a fully dense X with $n \approx 65,000$
  - HALLaR+ addresses this limitation by using a low-rank factorization scheme proposed by Burer and M** (2003) where all iterates have the form $X = UU^T$ for some $n \times r$ matrix U with $r \ll n$, and only U is stored

Outline of HALLaR+

HALLaR+ is an inexact augmented Lagrangian (AL) method

- Let initial multiplier $p_0 \in \mathbb{R}^m$ and penalty parameter $\beta_0 > 0$ be given
- for every $t \geq 1$, its t-th iteration consists of

$$X_t \approx \underset{X \in \mathbb{S}^n, X \succeq 0}{\operatorname{argmin}} \left\{ \mathcal{L}_{\beta_{t-1}}(X; p_{t-1}) := \langle C, X \rangle + \langle p_{t-1}, b - \mathcal{A}X \rangle + \frac{\beta_{t-1}}{2} \|b - \mathcal{A}X\|^2 \right\} \quad (AL_t)$$

$$p_t = p_{t-1} + \beta_{t-1}(b - \mathcal{A}X_t), \qquad \beta_t = 2\beta_{t-1}$$

- HALLaR+ uses a hybrid Frank-Wolfe (HFW) variant to solve subproblem (AL_t)
- HFW generates X_t in factored form $X_t = Y_t Y_t^T$ where # of columns of $Y_t \ll n$
- Only Y_t is computed and stored (X_t is **never** computed)

Outline of HFW for solving subproblem (AL_t)

0) Let $\tilde{U}_0 \in \mathbb{R}^{n \times \hat{r}_1}$ be the near-global solution Y_{t-1} found for subproblem (AL_{t-1}) and set $k = 1$

1) Starting from $\tilde{U}_{k-1} \in \mathbb{R}^{n \times r_k}$, use a nonconvex composite optimization solver, called ADAP-AIPP, to find a near stationary point U_k of the **unconstrained** problem

$$\min_{U \in \mathbb{R}^{n \times r_k}} \quad \mathcal{L}_{\beta_{t-1}}(UU^T; p_{t-1}) \qquad (P_k)$$

2) If $U_k U_k^T$ is a near-solution for subproblem (AL_t), then set $Y_t = U_k$ and **stop**

3) Perform a FW step w.r.t. subproblem (AL_t) at U_k to obtain $\tilde{U}_k \in \mathbb{R}^{n \times r_{k+1}}$, set $k \leftarrow k + 1$, and go to step 1

Step 2 - Check near optimality of $U_k U_k^T$ for subproblem $(\mathbf{AL}_t)$

- Compute a min eigenpair $(\lambda_{\min}(S_k), v_{\min}(S_k))$ of $S_k := \nabla_X \mathcal{L}_{\beta_{t-1}}(U_k U_k^T; p_{t-1})$, with $\|v_{\min}(S_k)\| = 1$, and set

$$y_k = \begin{cases} v_{\min}(S_k) & \text{if } \lambda_{\min}(S_k) < 0, \\ 0 & \text{otherwise} \end{cases}$$

- If

$$\lambda_{\min}(S_k) \;\geq\; -\epsilon_d \qquad |\langle S_k U_k, U_k \rangle| \;\leq\; \epsilon_c$$

  then declare U_k to be a near-solution of (AL_t)

Remark. The above termination condition is related to the one used in a method by Duchi et al. (2020) for solving a static convex problem over $\mathbb{S}^n_+$.

Step 3 - Performing the FW step at $U_k U_k^T$ for $(\mathbf{AL}_t)$

- Given U_k and y_k obtained in steps 1 and 2, respectively, compute the **two-dimensional** search

$$(\alpha_k, \omega_k) = \underset{\alpha,\omega}{\operatorname{argmin}} \left\{ \mathcal{L}_{\beta_{t-1}} \left(\alpha U_k U_k^T + \omega y_k y_k^T ; p_{t-1} \right) : \alpha \geq 0,\ \omega \geq 0 \right\}$$

and set

$$(\tilde{U}_k, r_{k+1}) = \begin{cases} \left(\sqrt{\omega_k}\, y_k\, ,\ 1 \right) & \text{if } \alpha_k = 0 \\ \left(\left[\sqrt{\alpha_k}\, U_k\, ,\ \sqrt{\omega_k}\, y_k \right] ,\ r_k + 1 \right) & \text{otherwise} \end{cases}$$

Remark. The two-dimensional conic search above is motivated by Laue (2012), who proposed a related search for a static convex problem over $\mathbb{S}_+^n$.

Theorem (Iteration-Complexity for HALLaR+)

Given a tolerance $\epsilon > 0$, HALLaR+ finds in

$$\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$$

iterations, and hence eigenpair computations, an ϵ-solution $(\bar{X}, \bar{p})$ of (P)-(D), i.e.,

$$\begin{aligned}
&(\epsilon\text{-primal feasibility}) && \|b - \mathcal{A}\bar{X}\| \leq \epsilon, && \bar{X} \succeq 0 \\
&(\epsilon\text{-dual feasibility}) && \lambda_{\min}(\bar{S}) \geq -\epsilon \\
&(\epsilon\text{-complementarity}) && \left|\langle \bar{S}, \bar{X} \rangle\right| \leq \epsilon
\end{aligned}$$

where $\bar{S} := C - \mathcal{A}^* \bar{p}$

Solvers included for Comparison

Our benchmark uses three SDP collections and compares the single GPU version of **cuHALLaR+** (**C**) against three other single-GPU solvers:

- **cuHALLaR** (**Julia**): original GPU implementation of the HALLaR method (Aguirre et al. 2025) for SDPs with a trace-bounding constraint
- **cuLoRADS** (**Julia**): augmented Lagrangian/ADMM method that solves the block ADMM subproblems using conjugate gradient (Han et al. 2024)
- **CARDAL** (**C**): curvature-aware, rank-adaptive augmented Lagrangian method based on the low rank factorization (Li et al. 2026)

All experiments were run on a single NVIDIA H200 GPU with 144 GB of memory

The benchmark is based on four groups of SDP instances:

- Matrix Completion SDP relaxations
- Maximum Stable Set SDP relaxations
- Quadratic Assignment Problem
- Instances from the Mittelmann SDP collection

Termination criteria:

$$\max\left\{\frac{\|\mathcal{A}X-b\|_2}{1+\|b\|_2},\ \frac{\left|\min\{0,\lambda_{\min}(C-\mathcal{A}^*p)\}\right|}{1+\|C\|_F},\ \frac{\left|\langle C,X\rangle-\langle b,p\rangle\right|}{1+\left|\langle C,X\rangle\right|+\left|\langle b,p\rangle\right|}\right\}\leq\epsilon$$

Computational Results: Matrix Completion, Stable Set, and Quadratic Assignment Problem

Instance		Time (s); /rank where shown			Ratio
Problem	Size	**cuLoRADS**	**cuHALLaR**	**cuHALLaR+**	**cuLo/ cuHA+**
Matrix completion: rank r; size $(n_1, n_2; m)$					
$r = 3$	240K, 360K; 71.8M	744.89	12.82/3	**10.74/3**	69.36
$r = 3$	320K, 480K; 97.9M	1187.79	17.41/3	**14.43/3**	82.31
$r = 3$	400K, 600K; 124M	2037.88	22.31/3	**18.13/3**	112.40
$r = 3$	480K, 720K; 151M	$*$/1.555e−6	28.13/3	**24.41/3**	–
$r = 3$	640K, 960K; 206M	3102.16	39.25/3	**30.36/3**	102.18
$r = 3$	800K, 1.2M; 261M	4124.44	53.3/3	**30.82/3**	133.82
Maximum stable set: size $(n; m)$					
G11	800; 1.60K	113.20	**4.23/2**	4.36/2	25.96
G55	5K; 12.5K	445.35	81.11/45	**64.68/48**	6.89
G81	20K; 40K	$*$/3.724e−1	64.20/3	**48.91/2**	–
$H_{14,2}$	16.4K; 115K	15.44	3.170/2	**0.93/2**	16.60
Quadratic assignment (QAPLIB): size $(n; m)$					
tai50a	2.5K; 2.55K	$*$/3.573e−2	–	**18.169/70**	–
tai60a	3.6K; 3.66K	$*$/4.454e−2	–	**31.262/85**	–
tai80a	6.4K; 6.48K	$*$/1.581e−2	–	**117.579/113**	–
tai100a	10K; 10.1K	$*$/4.399e−2	–	**174.861/141**	–

Table: Single-H200 results. Target tolerance: 10^{-5} for matrix completion and stable set; 10^{-4} for QAP (six-hour limit). $*/N$: reported maximum relative error N; –: no result or ratio. cuLoRADS times include dual-infeasibility evaluations.

Benchmark for some Mittelmann SDP instances

Problem Instance			Time (s)			Time ratio
name	n	m	**CARDAL**	**cuLoRADS**	**cuHALLaR+**	**cuLo/ cuHA+**
1et.2048	2,048	22,529	908.64	$*/4.858\mathrm{e}{-6}$	65.62	–
G40_mb	2,000	2,001	1456.10	59.20	15.12	3.92
hand	1,296	1,297	94.26	4.22	3.09	1.37
p_auss2_3.0	9,115	9,115	215.86	68.08	28.75	2.37
cancer_100	569	10,469	624.05	$*/1.578\mathrm{e}{-6}$	42.79	–
checker_1.5	3,970	3,970	210.92	28.90	7.60	3.80
foot	2,208	2,209	$*/1.150\mathrm{e}{-6}$	74.28	95.39	0.78
G48_mb	3,000	3,001	542.90	15.35	4.82	3.18
G60_mb	7,000	7,001	$*/2.775\mathrm{e}{-5}$	141.42	35.51	3.98
ice_2.0	8,113	8,113	236.10	72.39	51.47	1.41
rendl1	2,000	2,001	$*/3.668\mathrm{e}{-6}$	11.18	10.39	1.08
theta102	500	37,467	1560.72	17.54	15.28	1.15

Table: Single-H200 results with target tolerance 10^{-5} and a 3,600-sec time limit. $*/N$: reported maximum relative error N; –: no ratio.

Conclusion

- We have developed CPU and GPU solvers, HALLaR+ and cuHALLaR+, for solving huge SDPs **in general standard form**, i.e., without requiring the user to supply a trace bound
- HALLaR+ (resp., cuHALLaR+) can often solve SDPs with dimension pair on the order of millions in under a few hours (resp., few minutes)
- We are currently working on variants that handle SDPs with inequality constraints and/or block-structure matrix variable, and with the option of using multiple GPUs

References

- R.D.C. M***, A. Sujanani, D. Cifuentes. A low-rank augmented Lagrangian method for large-scale semidefinite programming based on a hybrid convex-nonconvex approach. *Mathematical Programming* (2026).
- J. M. Aguirre, D. Cifuentes, V. Guigues, R.D.C. M***, V.H. Nascimento, A. Sujanani. cuHALLaR: A GPU Accelerated Low-Rank Augmented Lagrangian Method for Large-Scale Semidefinite Programming. *arXiv preprint arXiv:2505.13719* (2025)
- J. M. Aguirre, R.D.C. M***, A. Sujanani. A Low-Rank Augmented Lagrangian Frank-Wolfe-Based Method for SDPs with Unbounded Feasible Regions (forthcoming)
- J. M. Aguirre, A.J. Kleywegt, R.D.C. M***. A Sparse Augmented Lagrangian Frank-Wolfe-Based Method for Large-Scale Linear Programming (forthcoming)
- cuHALLaR+ code can be downloaded from `https://github.com/OPTHALLaR`

Thank you all!

Low-Rank Idea: Burer-M*** Method (2003)

- Burer-M*** proposed a low-rank method in 2003 to circumvent all 3 of these issues
  - Especially the issue of storing X.
- Idea: restrict SDP to matrices of rank r. That is, they consider:

$$\min_{U} \quad \left\{ C \bullet UU^T \quad : \quad \mathcal{A}(UU^T) = b, \quad U \in \mathbb{R}^{n \times r} \right\} \qquad (P_r)$$

- Advantage: matrix variable U has $n \times r$ entries. X in (P) has $n(n+1)/2$
- Due to results from Pataki and Shapiro, if r is chosen large enough then if U_* is optimal for (P_r) then $U_* U_*^T$ is optimal for original problem (P).
  - Problem: (P_r) is nonconvex. Finding optimum is NP-hard.
- Nonconvexity is benign
  - If r large enough, then almost always, local minima of (P_r) are global minima of (P_r).
- Still, limited study of complexity/convergence rate guarantees

Step 1: Solving the Nonconvex Subproblem (P_k)

- Recall that, using $\tilde{U}_{k-1}$ as initial iterate, the goal of ADAP-AIPP is to find a near-stationary point U_k of

$$\min_{U \in \mathbb{R}^{n \times r_k}} \quad \mathcal{L}_{\beta_{t-1}}(UU^T; p_{t-1}) \qquad (P_k)$$

- ADAP-AIPP is an adaptive accelerated inexact proximal point method which is completely parameter-free; it is based on ideas from Carmon et al. '18, Paquette et al. '18, Kong et al. '19, and Sujanani & M** '23)
- More specifically, started from $V_0 = \tilde{U}_{k-1}$, ADAP-AIPP iterates as

$$V_i \approx \operatorname*{argmin}_{U \in \mathbb{R}^{n \times r_k}} \left\{ \mathcal{L}_{\beta_{t-1}}(UU^T; p_{t-1}) + \frac{1}{2\lambda_i} \|U - V_{i-1}\|^2 \right\}$$

  until a near-stationary point $V_i =: U_k$ is obtained, i.e., $\|R_k\|_F \leq \bar{\rho}$ where $R_k = 2S_k U_k$

- ADAP-AIPP uses an adaptive version of FISTA to solve its prox subproblems

0) Let $Y_{t-1} \in \mathbb{R}^{n \times \hat{r}_1}$ be the near-solution found for subproblem (AL_{t-1}) and set $\tilde{U}_0 = Y_{t-1}$ and $k = 1$

1) Starting from $\tilde{U}_{k-1} \in \mathbb{R}^{n \times r_k}$, use ADAP-AIPP to find a near stationary point U_k of

$$\min_{U \in \mathbb{R}^{n \times r_k}} \quad \mathcal{L}_{\beta_{t-1}}(UU^T; p_{t-1}) \qquad (P_k)$$

2) If $U_k U_k^T$ is a near-solution for subproblem (AL_t), then set $Y_t = U_k$ and **stop**

3) Perform a FW step w.r.t. (AL_t) at U_k to obtain $\tilde{U}_k \in \mathbb{R}^{n \times r_{k+1}}$, set $k \leftarrow k + 1$, and go to step 1

Remarks:

- Every iteration k of HFW computes a near-stationary solution U_k of (P_k) (using $\tilde{U}_{k-1}$ as a warm start) and performs a FW step to escape from U_k in case it is spurious
- All inner iterates generated by ADAP-AIPP, and hence U_k, have r_k columns
- $\tilde{U}_k$ generally has $r_k + 1$ columns, i.e., $r_{k+1} = r_k + 1$ (but $r_{k+1} = 1$ is possible but very unlikely)
- In practice, HFW usually perform only one iteration to solve most of the AL subproblems, in which case, its output has the same number of columns as its input