Momentum-inspired Low-Rank Coordinate Descent for Diagonally Constrained SDPs

Junhyung Lyle Kim, Jose Antonio Lara Benitez, Mohammad Taha Toghani, Cameron Wolfe, Zhiwei Zhang, Anastasios Kyrillidis
Rice University
Houston, TX, USA
jlylekim,jx24,mt72,crw13,zhiwei,anastasios@rice.edu

ABSTRACT
We present a novel, practical, and provable approach for solving diagonally constrained semi-definite programming (SDP) problems at scale using accelerated non-convex programming. Our algorithm non-trivially combines acceleration motions from convex optimization with coordinate power iteration and matrix factorization techniques. The algorithm is extremely simple to implement, and adds only a single extra hyperparameter—momentum. We prove that our method admits local linear convergence in the neighborhood of the optimum and always converges to a first-order critical point. Experimentally, we showcase the merits of our methodology for solving Diagonally Constrained SDPs at scale. Based on the low-rank coordinate descent approach to solving MaxCut and MaxSat SDPs at scale [59], we present a low-rank coordinate descent approach to solving Diagonally Constrained SDPs at scale [59].

1 INTRODUCTION
Background. This work focuses on efficient ways to solve semi-definite programming instances (SDPs) with diagonal constraints:

\[
\min_{X \succeq 0} \langle C, X \rangle \quad \text{s.t.} \quad X_{ii} = 1, \ i = 1, \ldots, n. \tag{1}
\]

Here, \(X \in S^n\) is the \(n \times n\) symmetric optimization variable and \(C \in S^n\) is a problem-dependent cost matrix. The above formulation usually appears in practice as the convex relaxation of quadratic form optimization over discrete variables.

\[
\min_{x \in \mathbb{D}^n} \langle x, Cx \rangle, \tag{2}
\]

\(\mathcal{D}\) is a discrete set on the unit imaginary circle:

\[
\mathcal{D} = \left\{ e^{\frac{2\pi m}{M}} : m = 0, 1, \ldots, M - 1 \right\},
\]

where \(i = \sqrt{-1}\) denotes the imaginary unit. Different values for \(M\) define different realizations of the discrete set \(\mathcal{D}\). For example, \(M = 2\) reduces \(\mathcal{D}\) to the binary set \(\{\pm 1\}\), while \(\mathcal{D}\) becomes \(\{\pm 1, \pm i\}\) for \(M = 4\).\(^1\) The formulation in (2) appears in many applications, including MaxCut or Max-k-Cut [5, 14, 16, 20, 21, 26, 29, 39, 49, 51], MaxSAT [20], maximal margin classification [19], semi-supervised learning [58], correlation clustering on dot-product graphs [55], community detection [1, 25], and (quantized) phase synchronization in communication networks [10, 64]. Other problems that can be cast as (quantized) quadratic form optimization over the unit complex torus include phase recovery [56], angular synchronization [50], and optimization problems in communication systems [27, 32, 33, 37, 42].

Typically, solving (2) is computationally expensive due to the presence of discrete structures. Discrete algorithmic solutions have been developed to solve (2) with near-optimal performance and rigorous theoretical guarantees by exploiting the problem’s structure [24, 30, 38, 41]. However, the go-to techniques for solving (2) are oftentimes continuous. For example, methods exist that relax combinatorial entities in (2) and find a solution with non-linear, continuous optimization. Such techniques can be roughly categorized into convex [17, 23, 44, 45, 53, 54] and non-convex, continuous approaches [8, 9, 11, 31, 59].

Recently, numerous algorithms have been proposed for solving large-scale SDP instances of the form (1). [59] present a low-rank coordinate descent approach to solving MaxCut and MaxSat SDP instances that are 10-100\(x\) faster than state-of-the-art non-convex solvers, and 9 to 10\(^3\)\(x\) faster than convex SDP solvers—with comparable or improved solution quality.

This Paper. We present an algorithmic prototype for solving diagonally constrained SDPs at scale. Based on the low-rank property of SDPs [2, 6, 47] (see Lemma 2.1), we focus on solving a non-convex, but equivalent, formulation of (1):

\[
\min_{V \in \mathbb{R}^{k \times n}} f(V) \equiv \langle C, V^TV \rangle \quad \text{s.t.} \quad \|v_i\|_2 = 1, \forall i. \tag{3}
\]

Here, \(V \in \mathbb{R}^{k \times n}\) is the optimization variable and the linear constraints on \(X\) in (1) translate to \(V \in (S^{k-1})^n\) (i.e., each column vector \(v_i\) satisfies \(\|v_i\|_2 = 1\)). Our algorithm, named Mixing Method++, adds momentum to a coordinate power iteration-style technique for solving SDPs by incorporating ideas from accelerated convex

\(^1\)Many other realizations of the set \(\mathcal{D}\) exist. For example, we define \(\mathcal{D} = \{1, e^{\pm \frac{2\pi}{3}}, e^{\pm \frac{4\pi}{3}}\}\) for Max-3-Cut [21, 51].
optimization and matrix factorization. As its predecessor [59], the algorithm is simple to implement, requiring only one additional hyperparameter - momentum. From an empirical perspective, we test the algorithm on MaxCut, MaxSAT, and MIMO signal detection applications, where we demonstrate significant speedups in comparison to other state-of-the-art solvers (e.g., 9× to 10× speedup on MaxSAT instances) with comparable or improved solution quality. Additionally, we prove that our accelerated method admits local linear convergence in the neighborhood of the optimum and always converges to a first order stationary point. To the best of our knowledge, this is the first theoretical result that incorporates momentum into a coordinate-descent approach for solving SDPs.

2 BACKGROUND

Notation. We use $\mathbb{S}^n$ to denote the cone of positive semi-definite (PSD) matrices. We denote the k-dimensional sphere by $S^k$ where $k - 1$ indicates the dimension of the manifold. The product of $n$, $k$-dimensional spheres is denoted by $(S^k)^n$. For any $v \in \mathbb{R}^n$ and $M \in \mathbb{R}^{n \times m}$, we define $\|v\|_2$ and $\|M\|_F$ as the $\ell_2$ and Frobenius norms. For a matrix $V \in \mathbb{R}^{k \times m}$, we denote its $i$-th column interchangeably as $V_i$ or $v_i$. For $a \in \mathbb{R}^n$, we define $D_a$ as the $n \times n$ diagonal matrix with entries $(D_a)_{ii} = a_i$. We denote the vector of all $1$'s as $\mathbf{1}$ and the Hadamard product as $\circ$. Singular values are denoted as $\sigma(\mathbf{.})$, while $\sigma_{\text{min}}(\mathbf{.})$ represents minimum non-zero singular value(s). The matrix $C$ corresponds to the cost matrix of the optimization problem (1).

Low-rank Property of SDPs. Focusing on (1), the number of linear constraints on $X$ is far less than the number of variables within $X$. This constitutes such SDPs as weakly constrained, which yields the following result [6, 47].

**Lemma 2.1.** The SDP in (1) has a solution with rank $k = \lceil \sqrt{2n} \rceil$.

One can enforce low-rank solutions to weakly constrained SDPs by defining $X = V^T V$, where $V \in \mathbb{R}^{k \times n}$ and $(k + 1)/2 > n$ [10, 11]. This low-rank parameterization, shown in (3), yields a non-convex formulation of (1) with the same global solution. This formulation can be solved efficiently because the conic constraint in (1) is automatically satisfied.\(^1\)

**Mixing Method.** Consider again the low-rank SDP parameterization given in (3). To connect this formulation with its discrete form in (2), assume $k = 1$, which yields the following expression:

$$\min_{v \in \mathbb{R}^{1 \times n}} v C u^T \quad \text{s.t.} \quad |v_i| = 1, \forall i,$$

(4)

Notice that $v$ is normalized entry-wise during each optimization step. To solve (4), one can consider using related algorithms, such as a power iteration-style methods [40], as shown below.

$$u \leftarrow C v, \quad \sigma^+ \leftarrow \text{normalize}(u),$$

Here, normalize($u$) projects $u$ such that $|u_i^2| = 1$ for each entry of $\sigma^+$. This is the crux of the Mixing Method [59], which applies a coordinate power iteration routine to sequentially update each column of $V$ and solve (3); see Algorithm 1. Variants of Algorithm 1 that utilize random coordinate selection instead of cyclic updates for the columns of $V$ have also been explored [15].\(^1\)

\(^1\)Handling a PSD constraint requires an eigenvalue decomposition of $n \times n$ matrices, leading to a $O(n^3)$ overhead per iteration.

**Algorithm 1 Mixing Method [59]**

1: Input: $C \in \mathbb{S}^n$, $V \in (S^k)^n$.
2: while not yet converged do
3:   for $i = 1$ to $n$ do
4:     $v_i \leftarrow \text{normalize}(\sum_{j \neq i} c_{ij} v_j)$
5:   end for
6: end while

In (3), diagonal elements of $C$ do not contribute to the optimization due to the constraint $|v_i|^2 = 1$, and thus can be set to zero. When considering only the $i$-th column of $V$, the objective $(C, V^T V)$ takes the form $2a_i^T \sum_{j \neq i} c_{ij} v_j$. The full gradient of the objective with respect to $v_i$ can be computed in closed form at each iteration, yielding the coordinate power iteration routine outlined in Algorithm 1. Interestingly, this procedure also comes with theoretical guarantees [59], which we include below for completeness.\(^3\)

**Theorem 2.1.** The Mixing Method converges linearly to the global optimum of (3), under the assumption that the initial point is close enough to the optimal solution.

3 MIXING METHOD++

Towards a Momentum-inspired Update Rule. In this work, we introduce Mixing Method++, which uses acceleration techniques [43, 48] to significantly improve upon the performance of Mixing Method. For gradient descent, a classical acceleration technique is the Heavy-Ball method [48], which iterates as follows:

$$w^{t+1} = w^t - \eta \nabla f(w^t) + \beta (w^t - w^{t-1}).$$

Here, $w$ is the optimization variable, $f(\mathbf{.})$ is a differentiable loss function, $\eta$ is the step size, and $\beta$ is the momentum parameter. Intuitively, the heavy-ball method exploits the history of previous updates by moving towards the direction $w^t - w^{t-1}$, weighted by the momentum parameter $\beta$. A similar momentum term can be naively incorporated into power iteration:

$$w^{t+1} = \text{normalize}(Cw^t + \beta (Cw^t - w^t)).$$

where there is no notion of a step size $\eta$. By adapting this accelerated power iteration scheme to the Mixing Method update rule, we arrive at the following recursion:

$$\delta_i = \text{normalize}(g_i + \beta(g_i - v_i)),$$

where $g_i = -\sum_{j \neq i} c_{ij} v_j$. Because $g_i$ is not normalized before the addition of the momentum term, $g_i$ and $v_i$ can be of significantly different magnitude. As such, we normalize $g_i$ as an intermediate step and adopt a double-projection approach per iteration; see Algorithm 2. Such double projection ensures that $u_i$ and $v_i$ are of comparable magnitude when the momentum term is added, resulting in significantly-improved momentum inertia.

**Properties of Mixing Method++.** The proposed algorithm is comprised of a two-step update procedure. The last normalization projects $u_i + \beta(u_i - u_i)$ onto the unit sphere, which ensures the updated matrix is feasible. In contrast, the first normalization ensures

\(^3\)Most of global guarantees in [59] hold for a variant of the Mixing Method that resembles coordinate gradient descent with step size, rather than the coordinate power iteration method that is used and tested in practice.
Algorithm 2 Mixing Method++

1: Input: $C \in \mathbb{S}^n, V \in (S^{k-1})^n, 0 \leq \beta < 1.$
2: while not yet converged do
3:   for $i = 1$ to $n$ do
4:     $u_i \leftarrow$ normalize $(-\sum_{j \neq i} c_{ij}v_j)$
5:     $v_i \leftarrow$ normalize $(u_i + \beta (u_i - v_i))$
6:   end for
7: end while

Figure 1: The minimum norm of the columns of $V$ throughout iterations of Mixing Method and Mixing Method++ on the MaxCut and MIMO objectives.

that $-\sum_{j \neq i} c_{ij}v_j$ and $v_i$ are comparable in magnitude, as the former may have a norm greater than 1. The overall computational cost of the added momentum step is negligible, making the step-wise computational complexity of Mixing Method++ roughly equal to that of the Mixing Method.

We show that Mixing Method++ converges linearly when $V$ lies in a neighborhood of the global optimum and always reaches a first-order stationary point. Despite the similarity of our convergence rates to previous work [59], achieving convergence in theory is not always feasible, even in convex, non-stochastic cases; see [4, 13, 18, 34, 36, 48]. Nonetheless, we observe acceleration in practice, and Mixing Method++ with $\beta = 0.8$ yields a robust algorithm with significant speed-ups in comparison to Mixing Method.

4 CONVERGENCE ANALYSIS

We begin by presenting some notation for brevity and readability. $f^*$ is used to denote the optimal value of the objective function $f(V) = \langle C, V^\top V \rangle$. The matrices $V$ and $\hat{V}$ refer to the current and next iterates from the inner iteration of Algorithm 2. For each $i \in [n]$, we define the family of maps $Z_i : \mathbb{R}^{k \times n} \rightarrow \mathbb{R}^{k \times n}$ as:

$$V \mapsto \begin{pmatrix} v_1 & \ldots & \hat{v}_i & v_i & \hat{v}_{i+1} & \ldots & v_n \end{pmatrix}.$$ (5)

In words, $Z_i(V)$ is the partially updated $V$ matrix (i.e., within the inner loop of Algorithm 2) before updating $v_i$ to $\hat{v}_i$. Thus, only the vector $v_i$ changes from $Z_i(V)$ to $Z_{i+1}(V)$. Notice that $Z_{n+1}(V) = \hat{V}$.

We now present our main theoretical results. All proofs are provided in Section 6.4 We adopt Assumption 3.1 from [59].

Assumption 4.1. For $i \in [n]$, assume $\|\sum_{j=1}^n c_{ij}v_j\|_2$ do not degenerate in the procedure. That is, all norms are always greater than or equal to a constant value $\delta > 0$.

Put simply, Assumption 4.1 states that the update rule will never yield $u_i$ with zero norm. We find that this assumption consistently holds in practice for both Mixing Method and Mixing Method++ applied over several objectives. In particular, we track the smallest value of $\|\sum_{j \neq i} c_{ij}v_j\|_2$ observed during the inner loop of Mixing Method and Mixing Method++, then plot these minimum norms throughout all iterations in Figure 1. As can be seen, Assumption 4.1 consistently holds for both MaxCut and MIMO objectives. Furthermore, if Assumption 4.1 does not hold, we can exploit the simple trick from [15] in practice. Given this assumption, no further assumptions are required for the second normalization step provided that $0 \leq \beta < 1$, as summarized below.

Observation 4.1. $w_i = \|u_i - \beta v_i\|_2 \neq 0$, whenever $0 \leq \beta < 1$.

Proof. Based on upper and lower triangle inequality bounds and the fact that $u_i$ and $v_i$ are both in $S^{k-1}$, we get the inequality

$$1 \leq \|1 + \beta\| - \beta \|v_i\|_2 \leq 1 + 2\beta.$$ \hfill \Box

Our first technical lemma lower bounds the decrease in the objective over inner iterations of Algorithm 2.

Lemma 4.1. Let $\hat{V}, V$ be the next and previous iteration of Algorithm 2, respectively. We define $y_i = \|g_i\|_2$, where $g_i = \sum_{j \neq i} c_{ij}v_j$. We then derive the following.

$$f(V) - f(\hat{V}) \geq \frac{1 - \beta}{1 + \beta} \sum_{i=1}^n y_i \|v_i - \hat{v}_i\|_2^2.$$ Further, $f$ is non-increasing whenever $0 \leq \beta < 1$.

Next, we lower bound the magnitude of the update from $V$ to $\hat{V}$ based on the objective residual, showing that the magnitude of our update will be large until we approach an optimum.

Lemma 4.2. Under assumption 4.1, there exists positive constants $\gamma$ and $\zeta$ such that

$$\|V - \hat{V}\|_F^2 \geq (\zeta - \gamma \|\Delta\|_F) (f(V) - f^*),$$

where $\Delta := (D_uD_y + \beta D_y) - (D_uD_y + \beta D_y)$. Finally, we upper bound the value of $\|\Delta\|_F^2$ with respect to the objective residual, which later enables the definition of a neighborhood for local linear convergence within Theorem 4.1.

Lemma 4.3. Under Assumption 4.1, $\exists \tau > 0$ such that

$$\|w + \beta E\| \Omega y - (w^* + \beta E) \Omega y^*\|_2^2 \leq \tau (f(V) - f^*).$$

Drawing upon these technical lemmas, we show Mixing Method++ converges linearly within a neighborhood of the optimum.
We test the set of all solvers used on problem instance $\mathcal{O}$, which is defined as the difference between objective values given by each solver and the lowest objective value achieved by any solver. We often report performance in terms of the objective residual, where

$$\beta = \frac{Obj(i)^{*} - Obj(i)}{\inf_{j \in \mathcal{O}} Obj(j)^{*} - Obj(j)}$$

for some positive constant $\kappa$ and $\zeta$, $\tau$, and $\gamma$ as defined in Lemmas 4.2 and 4.3. If $V \in \mathcal{N}_s$, we then have:

$$f(V) - f^* \leq (1 - \rho) (f(V) - f^*)$$

where $\rho \in (0,1]$. This result shows that Algorithm (2) converges linearly when $V \in \mathcal{N}_s$.

5 EXPERIMENTAL RESULTS

We test Mixing Method++ on three well-known applications of (1): MaxCut, MaxSAT, and MIMO signal detection. We use the same formulation in [59] to solve MaxCut and MaxSAT, while for MIMO we follow the experimental setup in [35]. We implement Mixing Method++ in C with $\beta = 0.8$. Each experiment is run on a single core in a homogeneous Linux cluster with 2.63 GHz CPU and 16 GB of RAM. We compare Mixing Method++ with wide variety of solvers depending on the application. For MaxCut, we compare with CGAL [63], Sedumii [52], SDPNAL+ [61], SDPLR [11], MoSeK [3], and Mixing Method [59]. For MaxSAT, we compare with Mixing Method and Loanda [7], Lastly, for MIMO signal detection, we compare with Mixing Method.

We often report performance in terms of the objective residual, which is defined as the difference between objective values given by each solver and the lowest objective value achieved by any solver. More formally, let $\text{Obj}_{ij}$ be the value of the objective function given by optimizer $i$ on SDP instance $j$. Then, the objective residual of optimizer $i$ on instance $j$ is defined as $|\text{Obj}_{ij} - \min_{p \in \mathcal{O}} \text{Obj}_{ij}|$, where $\mathcal{O}$ is the set of all solvers used on problem instance $j$.

5.1 MaxCut

The SDP relaxation of the MaxCut problem is given by

$$\max_{\|w\|_1 = 1} \frac{1}{2} \sum_{ij} c_{ij} \left( \frac{1-w_i w_j}{2}\right)$$

Table 1 lists the number of solved instances, median(log(objective residual)), and the pairwise acceleration ratio for different SDP solvers tested on MaxCut. Mixing Method++ uses only 20s while Mixing Method needs 200s. MoSeK and Sedumi consume more than $10^3$ and $10^4$ seconds respectively.

Table 1: Results on 203 MaxCut instances. SI stands for solved instances; MLOR stands for median(log(objective residual)). Mixing Method++ solved most instances efficiently with good precision.

| Solver       | # SI | MLOR  | Acc. ($\times$) |
|--------------|------|-------|-----------------|
| Sedumi       | 60   | -4.30 | 360.20          |
| MoSeK        | 74   | -3.24 | 348.37          |
| SDPNAL+      | 52   | -4.33 | 1316.35         |
| CGAL         | 92   | 0.93  | 49.19           |
| SDPLR        | 95   | -2.81 | 5.26            |
| Mixing Method| 109  | -2.80 | 5.26            |
| Mixing Method++ | 111 | -3.87 | 1              |

Figure 2: G40: $n = 2000$, number of edges = 11766. Mixing Method++ converges within 1s while Mixing Method uses about 4s. MoSeK and Sedumi need $10^2$ and $10^3$ seconds respectively to converge.

Figure 3: uk: $n = 4824$, number of edges $= 6837$. Consider the CPU time when each solver reaches objective residual $10^{-2}$. Mixing Method++ uses only 20s while Mixing Method needs 200s. MoSeK and Sedumi consume more than $10^3$ and $10^4$ seconds respectively.

where $C$ is the adjacency matrix. Thus, we aim to minimize the following objective function:

$$f_{\text{MaxCut}} = \left\langle C, \frac{\mathbb{1}_{n \times n} - Y^T Y}{4} \right\rangle \text{ s.t. } \|y\|_2 = 1 \forall i$$

where $V \in \mathbb{R}^{k \times n}$ and $\mathbb{1}_{n \times n}$ is the $n \times n$ matrix of all 1s. We consider the following benchmarks within our MaxCut experiments.

- **Gset**: 67 binary-valued matrices, produced by an autonomous, random graph generator. The dimension $n$ varies from 800 to $10^4$.
- **Dimacs**: 152 symmetric matrices with $n$ varying from 39 to 50,912,018, chosen for the 10-th Dimacs Implementation Challenge. We consider the 136 instances with dimension $n \leq 8 \cdot 10^6$.

We extensively tested our theoretical bound on $\beta$ in experiments. After testing all values in the set $[0,1]$ with an interval of 0.01, we observe that $\beta = 0.8$ indeed yields the best performance on over 90% of instances.

When an instance is solved by both Mixing Method++ and solver $s$ within the 24 hour time limit, the pairwise acceleration ratio is defined as the CPU time taken by solver $s$ divided by that of Mixing Method++.
with respect to CPU time of Mixing Method++ does not harm its precision, which is evident in its median logarithm objective residual of $-3.87$. Such performance is preceded only by Sedumi ($-4.30$) and SDPNAL+ ($-4.33$), both of which are more than $300\times$ slower than Mixing Method++.

We also study two specific cases of MaxCut: $G40$ from Gset and $uk$ from Dimacs10. In Figures 2 and 3, we plot objective residual and the objective residual of $s_{3v90c800-6}$. We also study two specific MaxCut instances, Mixing Method++ converges significantly faster than other solvers that were considered.

### 5.2 MaxSAT

Let $s_{ij} \in \{\pm 1, 0\}$ be the sign of variable $i$ in clause $j$. The following problem provides an upper bound to the exact MaxSAT solution:

$$\max_{\|u\|=1} \sum_{j=1}^{m} \left(1 - \frac{\|V_{s_j}\|^2 - (|s_j| - 1)^2}{4|s_j|}\right),$$

where $s_j = [s_{ij}, ..., s_{nj}]^T$. Therefore, the MaxSAT SDP relaxation can then be solved by minimizing the following, related objective:

$$f_{\text{MaxSAT}} = \langle C, V^TV \rangle \text{ s.t. } \|v_i\|_2 = 1 \forall i,$$

where $V \in \mathbb{R}^{k \times n}$ and $C = \sum_{j=1}^{m} \frac{s_{ij}s_{ij}^T}{4|s_j|}$.

We consider 850 problem instances from the 2016-2019 MaxSAT competition. Each instance is categorized as random, crafted or industrial. We evaluate Mixing Method++ as a partial MaxSAT solver\(^2\) and compare its performance to that of the Mixing Method and Loandra; the latter was the best partial solver in the 2019 MaxSAT competition. The best solution given by all solvers is used as ground truth when the optimal solution is not known.

The average approximation ratio of each solver across all MaxSAT instances is shown in Table 2. Mixing Method++ achieves the best average approximation ratio. We also consider two specific MaxSat instances: $s3v90c800-6$ from the random track and from the industrial track $wb.4m8s4.dimacs.filtered$. In Figures 4 and 5, we plot both the objective residual and the cost (i.e., the number of unsatisfied clauses) with respect to CPU time for $s3v90c800-6$ and $wb.4m8s4.dimacs.filtered$ instances, respectively. Mixing Method++ converges faster than Mixing Method in both cases and returns the best solution on the $s3v90c800-6$ instance. Although Loandra provides the best solution to $wb.4m8s4.dimacs.filtered$ quickly, Mixing Method++ still outperforms Mixing Method significantly.

### 5.3 MIMO

The MIMO signal detection setting is defined as follows:

$$\arg \min_{x \in S} \|y - Hx\|_2^2,$$

where $y \in \mathbb{R}^m$ is the signal vector, $H \in \mathbb{R}^{m \times n}$ is the channel matrix, and $x \in S$ is the transmitted signal. It is assumed that $y = Hx + v$, where $v$ is a noise vector. For our experiments, we consider the constellation $\{\pm 1\}$, which yields $S = \{\pm 1\}^n$. The MIMO SDP relaxation can then be solved by minimizing the following objective:

$$f_{\text{MIMO}} = \langle C, V^TV \rangle \text{ s.t. } \|v_i\|_2 = 1 \forall i,$$

\(^2\)We do not include CGAL and SDPLR in the plots, because the intermediate solution given by those methods is infeasible, which makes intermediate objective values difficult to quantify.

\(^3\)A partial solver is only required to provide the assignment it finds with the least number of violated clauses, while a complete solver must also provide proof that such an assignment is optimal.
We observe that Mixing Method sometimes decreases the objective residual when

\[ \text{Mixing Method}^+ \]

and

\[ \text{Mixing Method} \]

where

\[ \sigma^2 \]

and \( \theta \) is determined by the signal-to-noise ratio (SNR) (i.e., \( \sigma^2 = \frac{\text{SNR}}{\text{SNR}} \)). We perform tests with SNR \( \in \{8, 16\} \) and problem sizes \( \{m, n\} \in \{(16, 16), (32, 32), (64, 32)\} \). We plot the objective residual achieved by both Mixing Method and Mixing Method++ on MIMO instances with respect to CPU time in Figure 6. As can be seen, Mixing Method++ matches or exceeds the performance of the Mixing Method in all experimental settings.

5.4 The Effect of Sparsity

We empirically observe that Mixing Method++ achieves different levels of acceleration based on the sparsity of \( C \).\(^9\) To further understand the impact of sparsity on the performance of Mixing Method++, we compare Mixing Method++ to Mixing Method for objectives with different sparsity levels. In particular, we construct a MaxCut objective for a graph with 500 nodes and consider adjacency matrices with three different sparsity levels: 0.8, 0.5, and 0.2. In Figure 7, we plot the objective residual of both Mixing Method and Mixing Method++ on these problem instances. As can be seen, the acceleration achieved by Mixing Method++ is more significant when \( C \) is sparse. Furthermore, decreasing the value of \( \beta \) improves the performance of Mixing Method++ on MaxCut instances when \( C \) is dense.

5.5 Adaptive Momentum

We observe that Mixing Method sometimes decreases the objective residual faster than Mixing Method++ during early stage iterations. Inspired by this observation and previous work on adaptive momentum schedules \([12, 46, 57]\), we explore whether dynamically adjusting the value of \( \beta \) between iterations can further improve the performance of Mixing Method++. We propose the following momentum schedule:

\[ \beta_i = \beta (1 - \exp(-\alpha/T)) \]

where \( \beta = 0.8 \) is the default momentum, \( t \) is the current iteration, \( T \) is the total number of iterations, and \( \alpha \) is a constant such that \( \exp(-\alpha) \approx 0 \). Intuitively, this schedule begins with a low momentum value, which quickly increases to \( \beta = 0.8 \), thus forming a warm-up schedule for the momentum. As shown in Figure 8, this schedule improves the convergence speed of Mixing Method++.

6 LOCAL LINEAR CONVERGENCE

We now give detailed proof of Theorem 4.1. Let \( g_i \) be the linear combination in step 4 of Algorithm 1, before projecting onto the unit sphere:

\[ g_i = \sum_{j<i} c_{ij} \delta_j + \sum_{j>i} c_{ij} \delta_j. \]

We also define \( y_i = ||g_i||_2 \) and \( f(V) = \langle C, V^T V \rangle \) as the objective value for some viable solution \( V \). Therefore, step 4 of Algorithm 1 can be expressed as: \( y_i u_i = -g_i \). Thus, the changes presented in the Algorithm 2 can be written as the combination of 4 and the momentum, in step 5:

\[ u_i = -\frac{g_i}{y_i}, \quad \delta_i = \frac{(1 + \beta) u_i - \beta \delta_i}{w_i}. \]
where \( w_i = \|(1 + \beta) u_i - \beta \xi_i \|_2 \). This leads to the following one line relation,
\[
\begin{align*}
\hat{v}_i \hat{v}_i = -(1 + \beta) g_i - \beta g_i v_i, \quad \forall i \in [n].
\end{align*}
\] (6)

Herein, the matrices \( V \) and \( \hat{V} \) refer to the current and the end state of the inner iteration in algorithm 2. For each \( i \in [n] \), we define the family of maps \( Z_i : (S^{k-1})^n \to (S^{k-1})^n \) as follows
\[
Z_i(V) e_j = \begin{cases} 
\hat{v}_j, & \text{if } j < i, \\
\hat{v}_j, & \text{if } j \geq i,
\end{cases}
\] (7) where \( \{e_j\} \) is the canonical basis in \( \mathbb{R}^n \). It is trivial to see that \( Z_i(V) \) corresponds to the matrix in the inner cycle before updating \( v_i \) to \( \hat{v}_i \). Thus, only the vector \( v_i \) has been changed from \( Z_i(V) \) to \( Z_{i+1}(V) \). Also notice that \( Z_{n+1}(V) \) corresponds to the end of the inner cycle, i.e., \( Z_{n+1}(V) = \hat{V} \).

**Matrix Representation**

The transition from the state \( v_i \) to \( \hat{v}_i \) is expressed in the relation (6), then for any \( i \in [n] \) in each inner cycle we have,
\[
(1 + \beta) \sum_{j=1}^i c_{ij} \hat{v}_j + w_i \hat{v}_i \hat{v}_i = -(1 + \beta) \sum_{j>i} c_{ij} v_j - \beta g_i v_i. \quad \text{(8)}
\]

Let \( L \) be the strictly lower triangular matrix of \( C \) so (8) leads us to the representation:
\[
((1 + \beta) L + D_w D_y) \hat{V}^T = -((1 + \beta) L^T + \beta D_y) V^T. \quad \text{(9)}
\]

**Proof of Lemma 4.1**

Let \( \beta \) be in \([0,1]\). When fixing all the others variables, \( f \) with respect to \( v_i \) is given by
\[
\langle C, V^T v_i \rangle = 2v_i^T \left( \sum_j c_{ij} v_j \right) + \text{constant}.
\]

Since only \( v_i \) is changed, during the \( i \)th coordinate-wise updating, the only part of the objective that will change is \( 2v_i^T g_i \). That is,
\[
f(Z_i(V)) - f(Z_{i+1}(V)) = 2w_i \hat{v}_i - \beta \hat{v}_i.
\]

The updating rule of algorithm 2 is given by the recursion (6), so using this into the above equation yields
\[
f(Z_i(V)) - f(Z_{i+1}(V)) = \frac{2w_i (w_i - \beta)}{1 + \beta} (1 - \hat{v}_i^2 \hat{v}_i),
\]
\[
= \frac{w_i (w_i - \beta)}{1 + \beta} \| v_i - \hat{v}_i \|_2^2.
\]

From observation 4.1, we have \( w_i - \beta \geq 1 - \beta \). Thus,
\[
f(Z_i(V)) - f(Z_{i+1}(V)) \geq w_i \frac{1 - \beta}{1 + \beta} \| v_i - \hat{v}_i \|_2^2.
\]

Adding over all the \( i \), leads us to the desired conclusion
\[
f(V) - f(\hat{V}) = \sum_{i=1}^n f(Z_i(V)) - f(Z_{i+1}(V)) \geq \frac{1 - \beta}{1 + \beta} \sum_{i=1}^n w_i \| v_i - \hat{v}_i \|_2^2.
\]

The following observation will be useful later on.

**Observation 6.1.** For all \( i \in [n] \), \( w_i^* = 1 \)

**Proof.** Since \( w_i^* \) is a fixed point, using this fact in Eq. (6) yields \( (w_i^* + \beta) w_i^* = u_i^* \). Taking the norm, \( w_i^* + \beta = 1 + \beta \), since \( u_i^* \) and \( v_i^* \) are in the unit sphere.

Let \( S \) be defined as \((1 + \beta) C + D_w D_y + \beta D_y \) and let \( S^* \) be \((1 + \beta) C + D_w D_y + \beta D_y \); namely, the corresponding value of \( S \) when \( V \) is optimal in the optimization (3). We have the following observation.

**Observation 6.2.** \( S^* \) is PSD.

**Proof.** By observation 6.1 we have \( w_i^* = 1 \), so
\[
S^* = (1 + \beta) C + D_w D_y + \beta D_y = (1 + \beta) \left( C + D_y^* \right),
\]

where according to lemma 3.12, and since \( V \) is optimal, it follows that \( \sigma_{S_{1+\beta}} \geq 0 \), hence \( S^* \geq 0 \).

**Proof of Lemma 4.2**

By (9), we have
\[
\hat{V} = -V ((1 + \beta) L + \beta D_y) \left( (1 + \beta) L^T + D_w D_y \right)^{-1}.
\]

This leads to
\[
\begin{align*}
V - \hat{V} = & V ((1 + \beta) C + D_w D_y + \beta D_y) \\
& \left( (1 + \beta) L^T + D_w D_y \right)^{-1} \\
= & VS \left( (1 + \beta) L^T + D_w D_y \right)^{-1},
\end{align*}
\]

Using the definition \( \Delta \triangleq (D_w D_y + \beta D_y) - (D_w D_y + \beta D_y) \), and the fact that \( S = S^* + \Delta \) in the previous equation, we get
\[
\| V - \hat{V} \|_F^2 \geq ||V^T S^* V \|_F^2 + 2 \text{tr} (V^T V S^* R R^T \Delta). \quad \text{(10)}
\]

We claim that
\[
\|(w + \beta \mathbb{1}) \odot y - (w^* + \beta \mathbb{1}) \odot y^* \|_2 \Delta_\alpha + \Delta \geq 0.
\]

To see this, first notice \( D_w D_y + \beta D_y = D_{(w + \beta \mathbb{1}) \odot y} \). Thus,
\[
\Delta = D_{(w + \beta \mathbb{1}) \odot y} - D_{(w^* + \beta \mathbb{1}) \odot y^*}.
\]

Besides,
\[
\|(w + \beta \mathbb{1}) \odot y - (w^* + \beta \mathbb{1}) \odot y^* \|_2 \geq ||\Delta||_F.
\]

So the claim follows since \( \beta \geq 0 \) and termwise
\[
||\Delta||_F \geq ||\Delta_\alpha||_F + ||\Delta_t||_F \geq 0.
\]

Now, for the first part in (10), we get
\[
\| V S^* R \|_F^2 = \text{tr} (S^T V^T V S^* R R^T) \geq 0 \\
\geq \sigma_{S^*}^2 \text{tr}(V^T V S^*). \quad \text{(11)}
\]

For the second in (10), we get
\[
\text{tr}(V^T V S^* R R^T) \geq -||\Delta||_F \text{tr}(V^T V S^* R R^T) \geq \sigma_{S^*}^2 \text{tr}(V^T V S^*). \quad \text{(12)}
\]

this last inequality is given by our previous claim, the fact that \( S^* \geq 0 \) (observation 6.2) and
\[
\text{Tr}(V^T V S^* R R^T) = \text{Tr}(V^T V S^* D^2 U^T) = \text{Tr}(V^T V S^* D^2) \\
\leq \sigma_{S^*} \text{tr}(V^T V S^*). \quad \text{(13)}
\]
We know that \( \sigma^2_{\min}(R) = 2||\Delta_f||_F^2 \). Therefore, based on (15) and (16), it is known that \( ||Z_i s_i^*||^2 \) and \( ||v_i - s_i^*||^2 \) are both upper bounded by a positive factor of \( f(V) - f^* \).

The inner product of (12) with itself yields:

\[
\Delta_i^2 = \frac{1}{1 + \beta} \left( \frac{1}{1 + \beta} \right) ||Z_i s_i^*||^2 \leq 2||Z_i s_i^*||^2 + 2(y_i^* - \beta y_i) ||v_i - s_i^*||^2 \leq 2||Z_i s_i^*||^2 + 2(y_i^* + 2\beta y_i y_i^* + y_i^2) \left( ||v_i - s_i^*||^2 \right)
\]

Therefore, based on (15) and (16), we obtain: \( ||Z_i s_i^*||^2 \) and \( ||v_i - s_i^*||^2 \) are both upper bounded by a positive factor of \( f(V) - f^* \).

The inner product of (12) with itself yields:

\[
\Delta_i^2 = \frac{1}{1 + \beta} \left( \frac{1}{1 + \beta} \right) \left( ||Z_i s_i^*||^2 + 2||v_i - s_i^*||^2 \right) \leq 2||Z_i s_i^*||^2 + 2(y_i^* + 2\beta y_i y_i^* + y_i^2) \left( ||v_i - s_i^*||^2 \right)
\]

where \( i \) follows from the fact that \( s_i^* \) is a unit vector and \( ii \) follows from combining observation 6.1 with (14). Next, we have:

\[
\left( ||w + \beta 1|| \circ y - (w^* + \beta 1) \circ y^* \right) \leq \tau (f(V) - f^*)
\]

**Proof of Theorem 4.1**

Based on Lemma 4.3, a neighborhood can be selected around the optimum such that the value of \( f(V) - f^* \) is sufficiently small. We define this neighborhood through the selection of a positive constant \( \kappa \) such that the following inequality holds, where \( \tau \) is defined in Lemma 4.3:

\[
\left( \xi - \kappa \right)^2 \geq \tau (f(V) - f^*)
\]

Within this expression, \( \xi \) and \( \kappa \) are both defined in Lemma 4.2. Because \( \xi \) and \( \kappa \) are both strictly positive (i.e., see Lemma 4.2), \( \left( \frac{\xi - \kappa}{\gamma} \right)^2 \) is known to be strictly positive for all \( \kappa \in [0, \xi] \). Therefore, there always exists a value of \( \kappa \) such that (17) will be true within a sufficiently small neighborhood around the optimum. We combine (17) with the inequality from Lemma 4.3 to yield the following:

\[
\left( \frac{\xi - \kappa}{\gamma} \right)^2 \geq \tau (f(V) - f^*) \geq ||\Delta||^2_F
\]

Then, because \( \frac{\xi - \kappa}{\gamma} \) is strictly positive, we can manipulate the above expression to yield the following:

\[
\xi - \gamma ||\Delta||^2_F \geq \kappa
\]

The above inequality holds for all successive iterations of Algorithm 2. Then, based on the combination of Lemma 4.1 and assumption 4.1, \( f(V) - f(\hat{V}) \geq \delta \frac{1 - \beta}{1 + \beta} ||V - \hat{V}||^2_F \), and thus

\[
f(V) - f(\hat{V}) \geq \delta \frac{1 - \beta}{1 + \beta} ||V - \hat{V}||^2_F \]

where \( \hat{V} \) is derived by combining (18) and Lemma 4.2. This expression in turn implies the following:

\[
\left( 1 - \delta \kappa \frac{1 - \beta}{1 + \beta} \right) (f(V) - f^*) \geq f(\hat{V}) - f^*
\]

thus giving us the desired linear convergence. By substituting \( \rho = \delta \kappa \frac{1 - \beta}{1 + \beta} \), we arrive at the final result.
7 CONVERGENCE TO FIRST-ORDER CRITICAL POINT

Mixing Method++ not only has local linear convergence, but it always converges to a first-order critical point, per following Theorem:

Theorem 7.1. Let \( V^f = Z^f_{\mathbf{n}}(V) \), see definition (7), for all \( f \in \mathbb{N} \); i.e., the collection of points generated by Mixing Method++ after finishing the inner loop. Then, under the assumption 4.1, \( V^f \) converges to a limit point \( V \) and the Riemannian gradient at \( V \) is zero (Mixing Method++ converges to a first-order critical point).

Proof. For \( \beta \) in \([0, 1]\) we know by lemma 4.1 that the objective function is decreasing. Besides, \((S^{k-1})^n\) is compact, indeed limit point compact, so there exists a limit point \( \bar{V} \) such that \( f(\bar{V}) = \lim f(V^f) \) by continuity of \( f \). It is clear that \( \bar{V} \) is a fixed point in relation (9), so

\[
\bar{V} C = -\frac{\nabla D(\sigma + \beta \eta) \circ \hat{y}}{1 + \beta}, \tag{19}
\]

Since the constraint set is a product of spheres, its corresponding tangent space (henceforth denoted by the letter \( T \)) at the point \( \bar{V} \) is given by the product of tangent spaces of \( S^{k-1} \); namely

\[
T_{\bar{V}} \left( S^{k-1} \right)^n = \coprod_{j=1}^n T_{\bar{V}} \left( S^{k-1} \right), \tag{20}
\]

where \( \bar{V} \) is the \( j \)th column of \( \bar{V} \).

The above result gives us a way to characterize the tangent space. To this end, it is actually enough to work with the tangent space of \( S^{k-1} \).

By definition, for any \( v_j \), the vector \( \hat{v}_j \) is in the tangent space of \( S^{k-1} \) if and only if there exists a curve \( \phi : I \to S^{k-1} \) such that \( \phi(0) = v_j \) and \( \phi'(0) = \hat{v}_j \), where \( \phi' \) is the derivative of \( \phi \). Then \( \phi(t) \in S^{k-1} \) if satisfies \( \phi(t), \phi'(t) = 1 \) for all \( t \in I \). Differentiating on both sides leads us to \( 0 = \phi'(t), \phi'(t) + \phi(t), \phi(t) \). Evaluating at \( t = 0 \), we get

\[
2(\hat{v}_j, v_j) = 0,
\]

thus \( \{\hat{v}_j \in \mathbb{R}^k : (\hat{v}_j, v_j) = 0\} \subseteq T_{\bar{V}} S^{k-1} \), since both subspaces are of the same dimension, \( \{\hat{v}_j \in \mathbb{R}^k : (\hat{v}_j, v_j) = 0\} \approx T_{\bar{V}} S^{k-1} \).

Now by (20) and the previous analysis, we finally arrive to the characterization of the tangent space, namely

\[
T_{\bar{V}} \left( S^{k-1} \right)^n = \left\{ \bar{V} \in \mathbb{R}^{k \times n} : (\hat{v}_j, v_j) = 0 \quad \forall j \in [n] \right\},
\]

and \( \hat{v}_j \) (\( v_j \)) corresponds to the \( j \)th column of \( \bar{V} \) (\( V \)), respectively.

Let \( P^\perp : \mathbb{R}^{k \times n} \to T_{\bar{V}} \left( S^{k-1} \right)^n \) be the projector operator from the euclidean space to the tangent space at \( \bar{V} \), defined for any \( W \in \mathbb{R}^{k \times n} \) as follows,

\[
W \mapsto \left( w_1 - (\hat{v}_1, w_1)\hat{v}_1 \ldots w_n - (\hat{v}_n, w_n)\hat{v}_n \right).
\]

Let the Riemannian gradient of \( f \) denoted by \( \nabla f(\cdot) \), and the gradient of \( f \) defined in the entire euclidean domain as \( \nabla f(V) \). Using tools from matrix manifold the Riemannian gradient is given by

\[
\nabla f(\bar{V}) = P^\perp(\nabla f(\bar{V})).
\]

It is easy to see that \( \nabla f(\bar{V}) = 2\nabla C \), and by relation (19), this yields us to the equivalent version \( \nabla f(\bar{V}) = \frac{2}{1+\beta} \nabla D(\sigma + \beta \eta) \circ \hat{y} \).

Finally, \( \nabla f(\bar{V}) \) is a projection on the space of functions.

So, we have \((P^\perp(\nabla f(\bar{V})))_j = 0\), for all coordinates \( j \in [n] \), and thus

\[
\nabla f(\bar{V}) = P^\perp(\nabla f(\bar{V})) = 0.
\]

\( \Box \)

8 CONCLUSION AND FUTURE DIRECTIONS

We present a novel approach, Mixing Method++, to solve diagonally constrained SDPs. Mixing Method++ inherits the simplicity—and non-trivially preserves theoretical guarantees of—its predecessors. In practice, it yields not only faster convergence on nearly all tested instances, but also improvements in solution quality. Mixing Method++ adds one additional hyperparameter \( \beta \) for which we provide a theoretical upper bound (i.e., \( \beta < 1 \)). Using \( \beta = 0.8 \) experimentally leads to a robust algorithm, which outperforms Mixing Method and numerous other state-of-the-art SDP solvers. Theoretically validating the experimental acceleration provided by Mixing Method++ is still an open problem, which could potentially be handled with the help of Lyapunov Analysis [28, 60].

ACKNOWLEDGMENTS

AK acknowledges funding by the NSF (CCF-1907936). This work was partially done as MT’s and CW’s class project for “COMP545: Advanced Topics in Optimization,” Rice University, Spring 2021. AK thanks Danny Carey for his percussion performance at the song “Pneuma.”

REFERENCES

[1] E. Abbe. 2018. Community Detection and Stochastic Block Models. Foundations and Trends® in Communications and Information Theory 14, 1-2 (2018), 1-162.

[2] Farid Alizadeh, Jean-Pierre Haeberly, and Michael L. Overton. 1997. Complementarity and nondegeneracy in semidefinite programming. Mathematical programming 77, 1 (1997), 111–128.

[3] M. Moshef. 2019. Moshef optimization toolbox for MATLAB. User’s Guide and Reference Manual, version 4 (2019).

[4] M. Assran and M. Rabbat. 2020. On the Convergence of Nesterov’s Accelerated Gradient Method in Stochastic Settings. arXiv preprint arXiv:2002.12414 (2020).

[5] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. 1988. An application of combinatorial optimization to statistical physics and circuit layout design. Operations Research 36, 3 (1988), 493–515.

[6] Alexander I. Barvinok. 1995. Problems of distance geometry and convex properties of quadratic maps. Discrete & Computational Geometry 13, 2 (1995), 189–202.

[7] Jeremiah Berg, Emir Demirović, and Peter J. Stuckey. 2019. Core-Boosted Linear Programming for Incomplete MaxSAT. In Discrete & Computational Geometry (Stergiou (Eds.). Springer International Publishing, Cham, 39–56.

[8] S. Bhojanapalli, N. Boumal, P. Jain, and P. Netrapalli. 2018. Smoothed analysis of momentum-inspired low-rank coordinate descent for diagonally constrained SDPs. In Discrete & Computational Geometry (S. Burer and R. Monteiro. 2003. A nonlinear programming algorithm for solving rank minimization problems. SIAM Journal on Optimization 16, 1 (2005), 177–205.

[9] S. Bhojanapalli, Anastasios Kyrillidis, and Sujay Sanghavi. 2016. Dropping convexity for faster semi-definite optimization. In Discrete & Computational Geometry (S. Burer and R. Monteiro. 2003. A nonlinear programming algorithm for solving rank minimization problems. SIAM Journal on Optimization 16, 1 (2005), 177–205.

[10] N. Boumal. 2016. Nonconvex phase synchronization. SIAM Journal on Optimization 26, 4 (2016), 2355–2377.

[11] S. Burer and R. Monteiro. 2003. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. Mathematical Programming 95, 2 (2003), 329–357.
[12] John Chen, Cameron Wolfe, Zhao Li, and Anastasios Kyrillidis. 2019. Demon: Momentum Decay for Improved Neural Network Training. arXiv preprint arXiv:1910.04982 (2019).

[13] O. Devolder, F. Glineur, and Y. Nesterov. 1989. A discrete first-order method for large-scale MIMO detection with provable guarantees. IEEE Transactions on Signal Processing 62, 2 (2014), 682–695.

[14] M. Deva and D. Broque. 2014. Applications of cut polyhedra – II. J. Comput. Appl. Math. 55, 2 (1994), 217–247.

[15] Murat A. Erdogdu, Asuman Ozdaglar, Pablo A. Parrilo, and Nuri Denizcan Vanli. 2018. Convergence Rate of Block-Coordinate Maximization Burer-Monteiro Method for Solving Large SDPs. arXiv e-prints, Article arXiv:1807.04428 (July 2018). arXiv:1807.04428 pages. arXiv:1807.04428 [math.OC]

[16] Alan Frizee and Mark Jerrum. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[17] Michel X. Goemans and David P. Williamson. 2004. Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming. J. Comput. System Sci. 68, 2 (2004), 442–470.

[18] F. Gieseke, T. Pahikkala, and C. Igel. 2013. Polynomial runtime bounds for fixed-rank unsupervised least-squares classification. In Asian Conference on Machine Learning, 62–71.

[19] J. Hajek, Y. Wu, and J. Xu. 2016. Achieving exact cluster recovery threshold via integral quadratic constraints. In International Conference on Integer Programming and Combinatorial Optimization. Springer, 1–13.

[20] B. Gärtner and J. Matousek. 2012. Approximation algorithms and semidefinite programming. Springer Science & Business Media.

[21] E. Ghadimi, H. Feyzmahdavian, and M. Johansson. 2015. Global convergence of the heavy-ball method for convex optimization. In 2015 European control conference (ECC). IEEE, 310–315.

[22] J. Wang, T. Jebara, and S.-F. Chang. 2013. Semi-supervised learning using greedy elimination. SIAM Journal on Optimization 28, 2 (2018), 989–1016.

[23] R. V. Mises and Hilda Pollaczek-Geiringer. 1929. Praktische Verfahren der Gleichungsauslösung. ZAMM-Journal of Applied Mathematics and Mechanics 9, 2 (1929), 152–164.

[24] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[25] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[26] Alan Frizee and Mark Jerrum. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[27] M. Goemans and D. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[28] M. Goemans and D. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[29] M. Goemans and D. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[30] M. Goemans and D. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[31] E. Ghadimi, H. Feyzmahdavian, and M. Johansson. 2015. Global convergence of the heavy-ball method for convex optimization. In 2015 European control conference (ECC). IEEE, 310–315.

[32] J. Wang, T. Jebara, and S.-F. Chang. 2013. Semi-supervised learning using greedy elimination. SIAM Journal on Optimization 28, 2 (2018), 989–1016.

[33] R. V. Mises and Hilda Pollaczek-Geiringer. 1929. Praktische Verfahren der Gleichungsauslösung. ZAMM-Journal of Applied Mathematics and Mechanics 9, 2 (1929), 152–164.

[34] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[35] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[36] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[37] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[38] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[39] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[40] R. V. Mises and Hilda Pollaczek-Geiringer. 1929. Praktische Verfahren der Gleichungsauslösung. ZAMM-Journal of Applied Mathematics and Mechanics 9, 2 (1929), 152–164.

[41] A. Hartmann. 1996. Cluster-exact approximation of spin glass groundstates. Journal of the ACM (JACM) 42, 6 (1995), 1115–1145.

[42] R. V. Mises and Hilda Pollaczek-Geiringer. 1929. Praktische Verfahren der Gleichungsauslösung. ZAMM-Journal of Applied Mathematics and Mechanics 9, 2 (1929), 152–164.

[43] R. V. Mises and Hilda Pollaczek-Geiringer. 1929. Praktische Verfahren der Gleichungsauslösung. ZAMM-Journal of Applied Mathematics and Mechanics 9, 2 (1929), 152–164.

[44] R. V. Mises and Hilda Pollaczek-Geiringer. 1929. Praktische Verfahren der Gleichungsauslösung. ZAMM-Journal of Applied Mathematics and Mechanics 9, 2 (1929), 152–164.