A solver for clustered low-rank SDPs arising from multivariate polynomial (matrix) programs

Nando Leijenhorst, 4582640

Master Thesis,
MSc Applied Mathematics,
Delft University of Technology

Delft, May 26, 2021
Supervisor:
Dr. D. de Laat

A solver for clustered low-rank SDPs arising from multivariate polynomial (matrix) programs

by

Nando Leijenhorst

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday June 2, 2021 at 14:00 AM.

Student number:	4582640	
Masters program:	Applied Mathematics, optimization	
Faculty:	Electrical Engineering, Mathematics and Computer Science	
Thesis committee:	Prof. dr. D. C. Gijswijt,	TU Delft
	Dr. D. de Laat,	TU Delft, supervisor
	Dr. J. W. van der Woude,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

In this thesis, we give a primal-dual interior point method specialized to clustered low-rank semidefinite programs. We introduce multivariate polynomial matrix programs, and we reduce these to clustered low-rank semidefinite programs. This extends the work of Simmons-Duffin [J. High Energ. Phys. 1506, no. 174 (2015)] from univariate to multivariate polynomial matrix programs, and to more general clustered low-rank semidefinite programs.

Clustered low-rank semidefinite programs are programs with low-rank constraint matrices where the positive semidefinite variables are only used within clusters of constraints. The free variables can be used in any constraint, and can be used to connect clusters. The solver uses this structure to speed up the computations in two ways. First, the low rank structure is used to reduce matrix products to products of the form $u^T M v$, where M is a matrix and u and v are vectors, as already suggested by Löfberg and Parrilo in [43rd IEEE CDC (2004)]. Second, an additional block-diagonal structure is introduced due to the clusters. This gives the possibility to do operations such as the Cholesky decomposition block-wise.

We apply this to sphere packing and spherical cap packing. For sphere packing, the speed of the solver is compared to the often used arbitrary precision solver SDPA-GMP, showing the theoretical speedup in time complexity. We give a new three-point bound for the maximum density when packing spherical caps of N sizes on the unit sphere.

Contents

1. Introduction

A polynomial optimization problem can be stated as follows: given n-variate polynomials p, $g_1, \ldots, g_m$, compute

$$\inf\{p(x) \mid x \in \mathbb{R}^n \text{ with } g_i(x) \geq 0 \text{ for } i = 1, \ldots, m\},$$

or, equivalently, compute

$$\sup\{\lambda \in \mathbb{R} \mid p - \lambda \geq 0 \text{ on } S\}, \tag{1.1}$$

where $S = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0 \text{ for } i = 1, \ldots, m\}$. These kind of problems, although usually in a more difficult form, can for example be found in robust optimization for control theory [27] and extremal geometry [1, 2, 10]; see [25] for more applications. In general, the problem (1.1) is NP-hard, see [21] for an overview of instances. Therefore, it is common to relax the problem using weighted sums of squares of polynomials, which are polynomials of the form

$$s(x) = s_0(x) + \sum_{i=1}^{m} g_i(x) s_i(x),$$

where $s_i(x) = \sum_{j=1}^{m_i} (q_i^j(x))^2$ are sums of squares of polynomials. Weighted sums of squares of polynomials are trivially nonnegative on S, hence this can be used to relax (1.1) by requiring $p - \lambda$ to be a weighted sum of squares. In general, this relaxation is strict: a nonnegative polynomial is not necessarily a weighted sum of squares of polynomials. Hilbert for instance characterized the cases for $S = \mathbb{R}^n$ for which a nonnegative polynomial can always be written as a sum of squares polynomial:

Theorem 1.1 ([16])**.** *Any nonnegative, n-variate polynomial of total degree d is a sum of squares polynomial if and only if $n = 1$, $d = 2$, or $(n, d) = (2, 4)$.*

For $(n, d) = (2, 6)$, a famous example is the Motzkin polynomial

$$M(x, y) = x^4y^2 + x^2y^4 - 3x^2y^2 + 1,$$

which can be proven to be nonnegative by the arithmetic mean-geometric mean inequality, but cannot be written as a sum of squares of polynomials. When S is compact, Putinar showed that strictly positive polynomials on S can always be written as weighted sums of squares of polynomials if $N - \sum_j x_j^2$ can be written as a weighted sum of squares [26]. Note that this is an algebraic certificate that S is compact.

These relaxations are useful, because optimization over sums of squares of polynomials can be done using semidefinite programming. Suppose s is a sum of squares of polynomials, i.e., $s(x) = \sum_i q_i(x_i)^2$. Then, given a basis of polynomials $b(x)$, we can write

$$s(x) = \sum_i b(x)^T q_i q_i^T b(x) = b(x)^T (\sum_i q_i q_i^T) b(x) = b(x)^T Q b(x) = \mathrm{Tr}(Q b(x) b(x)^T),$$

where q_i are the coefficients of $q_i(x)$ with respect to the basis $b(x)$, and $Q = \sum_i q_i q_i^T$ is by construction positive semidefinite. Conversely, by the spectral decomposition, every positive semidefinite matrix Q gives rise to a sum of squares polynomial $s(x)$.

We can use this to rewrite a relaxation to a semidefinite program (SDP), which is a program of the form

$$\inf\{\langle C, Y\rangle \mid \langle A_i, Y\rangle = b_i \text{ for } i = 1, \ldots, m,\, Y \succeq 0\}\,,$$

where $\langle A, B\rangle = \operatorname{Tr}(B^T A)$ is the trace inner product. An often used method for this translation is coefficient matching, based on the principle that two polynomials are equal if they have the same coefficients when written as $\sum_k c_k p_k(x)$ for a basis of polynomials $\{p_k\}_{k\geq 0}$. In this way, we can write

$$\sum_{k=0}^{d} c_k^s p_k(x) = s(x) = \operatorname{Tr}(Q b(x) b(x)^T) = \sum_{k=0}^{d} \operatorname{Tr}(Q B_k) p_k(x)\,,$$

where each element of B_k is the kth coefficient of the corresponding element of $b(x)b(x)^T$ with respect to the basis $\{p_k\}_{k\geq 0}$. This reduces to constraints

$$\langle B_k, Q\rangle = c_k^s \qquad \text{for } k = 0, \ldots, d\,.$$

A different method is sampling: two univariate polynomials of degree d are equal when they evaluate to the same value on $d+1$ distinct points. More generally, one can choose a set of sample points $\{x_k \in \mathbb{R}^n \mid k = 1, \ldots, K\}$ such that the values of n-variate polynomials on these sample points uniquely determine the polynomial in the space of n-variate polynomials up to a certain degree d. An example of such points is the set of rational points in the unit simplex with denominator d [22]. Such a set of sample points allows us to write

$$s(x_k) = \operatorname{Tr}(Q b(x_k) b(x_k)^T) \qquad \text{for } k = 1, \ldots, K\,,$$

which directly gives semidefinite programming constraints.

Both methods have different properties which can be exploited while solving the SDP. The first method, coefficient matching, leads to sparse constraint matrices. Suppose that the basis $\{p_k\}_{k\geq 0}$ and the vector $b(x)$ consist of polynomials of strictly increasing degree, such as the monomials where $p_k(x) = x^k$. Then $(b(x)b(x)^T)_{ij}$ has degree at most $i + j$, and hence $(B_k)_{ij} = 0$ when $i + j < k$. This sparsity can be exploited when solving the SDP by reducing the cost for matrix products, as is done in for example in the SDP solver SDPA [32, 33].

Sampling has the advantage that it leads to rank one constraint matrices $b(x_k)b(x_k)^T$. This can be exploited to speed up certain computations [22, 29] because matrix vector products can be used instead of matrix products; see also Chapter 3.

In the Chapter 2, we will focus on the inner workings of an SDP solver for the following SDP, with free variables y:

$$\inf\{\langle C, Y\rangle \mid \langle A_i, Y\rangle + By = b_i\,,\ \text{for } i = 1, \ldots, m\}\,.$$

To connect to (1.1), the free variable can be used to model the λ in $p-\lambda$. More generally, an SDP can be used to model constraints of the form

$$\langle A(x), Y\rangle + b(x)^T y \geq 0 \quad \text{ on } S\,,$$

where Y is a (positive semidefinite) matrix variable, y are scalar variables, $A(x)$ is a polynomial matrix and $b(x)$ a polynomial vector.

Recall that, if the method of sampling is used to obtain semidefinite programming constraints from polynomial constraints, the constraint matrices A_i are of a very specific form. They have a rank 1 block which corresponds to the variable matrix of one polynomial constraint, and are zero on blocks not corresponding to that polynomial constraint. This generalizes to what we call a clustered low-rank semidefinite program:[1]

$$\begin{aligned}
\sup \quad & \sum_{j=1}^{J}\sum_{l=1}^{L_j} \operatorname{Tr}(C^{j,l}Y^{j,l}) + b^T y && \text{with } Y^{j,l} \in \mathcal{S}^{U_{j,l}}, y \in \mathbb{R}^N \\
\text{s.t.} \quad & \sum_{l=1}^{L_j} \operatorname{Tr}(A_*^{j,l}Y^{j,l}) + B^j y = c^j && \text{for } j = 1,\ldots,J\,, \\
& Y^{j,l} \succeq 0\,,
\end{aligned}$$

where $\operatorname{Tr}(A_*^{j,l}Y^{j,l})$ denotes the vector with elements $\operatorname{Tr}(A_i^{j,l}Y^{j,l})$, and the matrices $A_i^{j,l}$ have low rank. Note that the only connection between the clusters j are the variables y. This is a more general version of the SDP used for the arbitrary precision SDP solver SDPB [29], where $L_j = 2$ and the constraint matrices $A_i^{j,l}$ are of a specific rank 1 or rank 2 form.

In Chapter 3, we use the structure of the clustered low-rank SDP to speed up the main computations in the solver given in Chapter 2. This is done mainly in two ways. Firstly, we use the low rank structure to reduce matrix products to products of the form $u^T M v$, where u, v are vectors and M is a matrix. Secondly, the clustered nature gives block structures during the solving, which allows for computations on smaller matrices and parallel computing. This extends the work of Simmons-Duffin [29]. The solver has been named CLRS for clustered low-rank solver. We give an implementation of CLRS[2] in Julia [4], which is a readable, high-level programming language with a Just-In-Time (JIT) compiler. In addition, we derive the time complexity and compare this to SDPA-GMP, to the extend possible.

In Chapter 4, we introduce a multivariate polynomial matrix program (MPMP), and give a derivation from the MPMP to the clustered low-rank SDP. This is an extension from the univariate polynomial matrix programs introduced by [29]. An MPMP can be seen as an SDP with linear polynomial matrix inequalities as constraints: $\sum_i y_i A_i \succeq 0$ where $A_i \in \mathbb{R}[x_1,\ldots,x_n]^{m\times m}$. Note that this also includes polynomials ($m = 1$) and normal linear matrix inequalities ($n = 0$).

[1] Note the difference between low-rank semidefinite programming (i.e., semidefinite programming with a rank constraint on the solution matrix), and solving low-rank semidefinite programs (i.e., semidefinite programs with low-rank constraint matrices).

[2] The code is available via github: `https://github.com/nanleij/Clustered-Low-Rank-SDP-solver`

Chapter 5 uses the problem of packing spheres of N different sizes in $\mathbb{R}^n$ as example of an MPMP. This is used to compare the speed of SDPB and CLRS as well as the commonly used arbitrary precision SDP solver SDPA-GMP, using results of [10].

Another example of an MPMP can be found in Chapter 6, where we consider the problem of packing N sizes of spherical caps on a unit sphere. Both two and three-point bounds on the maximum density are given; the two-point bound for N sizes of caps in [10] and the three-point bound for equal sized caps [1, 2, 13, 23] are extended to a three-point bound for N sizes of caps. A more general k-point bound, an extension of results of [11], is also obtained. Symmetry reductions as in [13, 23] are used, for which the possibility of using low ranks in the solver is imperative. The symmetry reductions lead to new research questions regarding the combination of sampling for SDPs and invariance theory.

2. A primal-dual interior point method for SDPs with free variables

In this chapter, we discuss a general primal-dual interior point method for semidefinite programs (SDPs) with free variables, following the discussion from [29]. This will be specialized in Chapter 3 to a clustered low-rank SDP, similar to the clustered rank 1 structure of SDPB [29], giving a speed up. We consider programs of the form

$$\begin{aligned} \sup \quad & \operatorname{Tr}(CY) + b^T y && \text{with } Y \in \mathcal{S}^M,\ y \in \mathbb{R}^N \\ \text{s.t.} \quad & \operatorname{Tr}(A_* Y) + By = c\,, \\ & Y \succeq 0\,, \end{aligned} \tag{2.1}$$

where

$$\begin{aligned} C, A_1, \ldots, A_p &\in \mathcal{S}^M\,, \\ B &\in \mathbb{R}^{P\times N}\,, \\ c &\in \mathbb{R}^P\,, \\ b &\in \mathbb{R}^N\,. \end{aligned}$$

Here $\operatorname{Tr}(A_* Y)$ is the vector with entries $\operatorname{Tr}(A_i Y)$ for $i = 1, \ldots, P$.

In the next section, the dual to this program is given and weak duality is proven. We use this in Section 2.2 to obtain the algorithm.

2.1 Duality

Program (2.1) is usually referred to as the dual program $\mathcal{D}$, with the following primal program $\mathcal{P}$:

$$\begin{aligned} \inf \quad & c^T x && \text{with } x \in \mathbb{R}^P,\ X \in \mathcal{S}^M \\ \text{s.t.} \quad & X = \sum_{p=1}^{P} x_p A_p - C\,, \\ & B^T x = b\,, \\ & X \succeq 0\,. \end{aligned} \tag{2.2}$$

Using that X and Y are positive semidefinite, it is easy to prove weak duality.

Theorem 2.1 ([29, Theorem 2.2])**.** *Given a feasible point (x, X) of $\mathcal{P}$ and a feasible point (y, Y) of $\mathcal{D}$, the duality gap $c^T x - \operatorname{Tr}(CY) - b^T y$ is nonnegative. If the duality gap vanishes, then (x, X) and (y, Y) are both optimal and $XY = 0$.*

Proof. Suppose (y, Y) and (x, X) are feasible solutions of $\mathcal{D}$ and $\mathcal{P}$ respectively. Using the constraints of $\mathcal{P}$ and $\mathcal{D}$ gives the duality gap

$$\begin{aligned} c^T x - \mathrm{Tr}(CY) - b^T y &= c^T x - \mathrm{Tr}(\sum_p x_p A_p Y) + \mathrm{Tr}(XY) - x^T B y \\ &= x^T (c - By - \mathrm{Tr}(A_* Y)) + \mathrm{Tr}(XY) \\ &= \mathrm{Tr}(XY) \geq 0\,. \end{aligned}$$

Suppose the duality gap vanishes. As the primal gives an upper bound on the value of the dual, and the dual a lower bound on the value of the primal, both bounds are attained and optimal. Furthermore, $\mathrm{Tr}(XY) = 0$, implying that $XY = 0$. □

2.2 The algorithm

The general structure is modelled after SDPA [32, 33], using a Newton search direction. The main difference is the existence of free variables; with SDPA these variables require an extra diagonal block and slack variables.

As is common in primal-dual algorithms, the optimality condition $XY = 0$ is deformed to $XY = \mu I$. This can be interpreted in terms of the logarithmic barrier function $X \mapsto -\ln\det X$. For $\mu \in \mathbb{R}_+$, this defines together with the constraints in $\mathcal{P}$ and $\mathcal{D}$ a unique family of solutions, the central path. A primal-dual method follows this path to the optimum ($\mu = 0$).

2.2.1 The search direction

Here we will use the Newton search direction to approximately move to the point on the central path corresponding to μ. In the algorithm, μ will decrease at each iteration, thus following the central path to the point with $\mu = 0$. The search direction is defined by replacing the variable $q = (x, X, y, Y)$ by $q + dq$ in the constraints and optimality condition, and solving the resulting equations linearized in dq. Following [29], we have:

$$\begin{aligned} X + dX &= \sum_i (x_i + dx_i) A_i - C\,, \\ B^T(x + dx) &= b\,, \qquad (2.3) \\ \mathrm{Tr}(A_*(Y + dY)) + B(y + dy) &= c\,, \qquad (2.4) \\ XY + X\ dY + dX\ Y &= \mu I\,. \end{aligned}$$

Defining the residues

$$\begin{aligned} P &\coloneqq \sum_i x_i A_i - X - C\,, \\ p &\coloneqq b - B^T x\,, \\ d &\coloneqq c - \mathrm{Tr}(A_* Y) - By\,, \\ R &\coloneqq \mu I - XY\,, \qquad (2.5) \end{aligned}$$

gives

$$dX = P + \sum_i dx_i A_i \,,$$
$$dY = X^{-1}(R - dX\, Y)\,. \tag{2.6}$$

We use this together with (2.3) and (2.4) to obtain a system of equations for dx and dy. Substituting the expressions for dY and dX in (2.4) gives

$$\mathrm{Tr}(A_*(Y + X^{-1}(R - (P + \sum_i dx_i A_i)Y))) + B(y + dy) = c\,.$$

Hence

$$\begin{aligned} -\mathrm{Tr}(A_* X^{-1} \sum_p dx_p A_p Y) + Bdy &= c - By - \mathrm{Tr}(A_* Y) - \mathrm{Tr}(A_* X^{-1}(R - PY)) \\ &= d + \mathrm{Tr}(A_* X^{-1}(PY - R))\,. \end{aligned}$$

Let $Z = X^{-1}(PY - R)$, and $S \in \mathbb{R}^{P\times P}$ the so-called Schur complement matrix defined by $S_{ij} = \mathrm{Tr}(A_i X^{-1} A_j Y)$. This then results, together with (2.3), in the system of equations

$$T \begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} -d - \mathrm{Tr}(A_* Z) \\ p \end{pmatrix}\,, \text{ with } T \coloneqq \begin{pmatrix} S & -B \\ B^T & 0 \end{pmatrix}\,. \tag{2.7}$$

Note that (2.6) gives a dY which is not necessarily symmetric, hence we take

$$\widehat{dY} = \frac{1}{2}(dY + dY^T)$$

instead.

2.2.2 Predictor-Corrector directions

Forming the matrix S is often the most costly operation, due to the matrix products which need to be computed for elements $S_{ij} = \mathrm{Tr}(A_i X^{-1} A_j Y)$. Therefore it is common practice to reuse the left-hand side of the system (2.7) in the following way, due to Mehrotra [24]. First the search direction as explained in the previous section is computed. This results in the predictor direction $dq_p = (dx_p, dX_p, dy_p, dY_p)$. This is used to refine the corrector search direction by improving the approximation of the linearized system by solving the equations corresponding to the constraints together with

$$XY + X\; dY + dX\; Y + dX_p\; dY_p = \mu I\,.$$

This only affects the definition of R, which becomes

$$R_c \coloneqq \mu I - XY - dX_p dY_p\,,$$

instead of (2.5). Note that this only changes the right hand side of (2.7), hence most of the work done in solving the system of equations can be reused.

2.2.3 The full algorithm

Given the input, the algorithm consists of the following steps [29].

1. Initialize $q = (x, X, y, Y) = (0, \Omega_p I, 0, \Omega_d I)$, where Ω_p and Ω_d are parameters chosen by the user.

2. Compute the residues and terminate if they satisfy the termination criteria.

3. Take $\mu = \mathrm{Tr}(XY)/K$, where K is the number of rows of X. Define $\mu_p = \beta_p \mu$, where
$$\beta_p = \begin{cases} 0 & \text{if } q \text{ is both primal and dual feasible,} \\ \beta_{\text{infeasible}} & \text{otherwise.} \end{cases}$$
The parameter $\beta_{\text{infeasible}} \in (0, 1)$ is chosen by the user.

4. Compute the predictor search direction $dq_p = (dx_p, dX_p, dy_p, dY_p)$ with $R_p = \mu I - XY$.

5. Compute $\mu_c = \beta_c \mu$ as follows: define $r = \mathrm{Tr}((X + dX_p)(Y + dY_p))/(\mu K)$. Let $\beta = r^2$ if $r < 1$, and r otherwise. Then, with $\beta_{\text{feasible}} \in (0, 1)$ chosen by the user, we set
$$\beta_c = \begin{cases} \min(\max(\beta_{\text{feasible}}, \beta), 1) & \text{if } q \text{ is primal and dual feasible,} \\ \max(\beta_{\text{infeasible}}, \beta) & \text{otherwise.} \end{cases} \tag{2.8}$$

6. Compute the corrector search direction $dq_c = (dx_c, dX_c, dy_c, dY_c)$ with $R = \mu_c I - XY - dX_p dY_p$.

7. Determine steplengths by
$$\alpha_p = \min(\gamma \alpha(X, dX_c), 1)\,,$$
$$\alpha_d = \min(\gamma \alpha(Y, dY_c), 1)\,,$$
where $\alpha(M, dM)$ is the largest number such that $M + \alpha(M, dM) dM$ is positive semidefinite.

8. Update the primal and dual variables:
$$x \leftarrow x + \alpha_p dx_c\,,$$
$$X \leftarrow X + \alpha_p dX_c\,,$$
$$y \leftarrow y + \alpha_d dy_c\,,$$
$$Y \leftarrow Y + \alpha_d dY_c\,.$$

9. Repeat from step 2.

An example of a termination criterion [29] is that the primal and dual error ($\max\{|P_{ij}|, |p_i|\}$ respectively $\max\{|d_i|\}$) , and the duality gap, defined by

$$\frac{|\texttt{PrimalObjective} - \texttt{DualObjective}|}{\max(1, |\texttt{PrimalObjective} + \texttt{DualObjective}|)} ,$$

are 'small enough'. The maximum size is quantified by the user.

For the step length we need to determine $\alpha(M, dM)$. This can be done by a Cholesky decomposition $M = LL^T$ in combination with the computation of a minimum eigenvalue: $M + \alpha(M, dM)dM = L(I + \alpha(M, dM)L^{-1}dML^{-T})L^T$ is postive semidefinite if and only if $\alpha(M, dM)\lambda_{\min} \geq -1$, where $\lambda_{\min}$ denotes the minimum eigenvalue of $L^{-1}dML^{-T}$. Hence computing $\lambda_{\min}$ gives directly the value of $\alpha(M, dM)$

3. A primal-dual interior point method for clustered low-rank SDPs

In this chapter, we will introduce the clustered low-rank semidefinite program, and show how the structure can be used to speed up the computations of the search direction in the algorithm of Chapter 2. Furthermore, the time complexity will be derived. We compare this to the complexity of SDPA-GMP, a general purpose arbitrary precision semidefinite programming algorithm.

The general structure of the clustered low-rank SDP is similar to the structure of the SDP which can be solved with SDPB [29, 20]. However, a clustered low-rank SDP is more general. The solver, called CLRS for clustered low rank solver, can be seen as a customized version of SDPB, and is programmed in Julia. Both SDPB and CLRS use arbitrary precision arithmetic; this is inherent to the problems considered (the conformal bootstrap and problems in extremal geometry). Some other solvers such as SDPT3 [31] can also use low-rank structures, but do not use the structure of the clusters and use machine precision.

It should be noted that there is a difference between low-rank semidefinite programming, where the solution is required to have low rank, and solving a low-rank semidefinite program, where the constraint matrices have low rank but the solution can be of any rank.

3.1 The clustered low-rank SDP

The SDP is called

- Clustered, because the constraints are divided into J clusters. The clusters use different positive semidefinite matrix variables, and are only connected through free scalar variables.
- Low-rank, because the constraint matrices are assumed to have low rank.

This leads to the following form of the program:

$$\begin{aligned}
\text{sup} \quad & \sum_{j=1}^{J} \sum_{l=1}^{L_j} \operatorname{Tr}(C^{j,l} Y^{j,l}) + b^T y && \text{with } Y^{j,l} \in \mathcal{S}^{U_{j,l}}, y \in \mathbb{R}^N && (3.1)\\
\text{s.t.} \quad & \sum_{l=1}^{L_j} \operatorname{Tr}(A_*^{j,l} Y^{j,l}) + B^j y = c^j \,, && \text{for } j = 1, \ldots, J \,, && (3.2)\\
& Y^{j,l} \succeq 0 \,,
\end{aligned}$$

where

$$\begin{aligned} C^{j,l}, A_1^{j,l}, \ldots, A_{P_j}^{j,l} &\in \mathcal{S}^{U_{j,l}}, && \text{for } j = 1, \ldots, J,\ l = 1, \ldots, L_j, \\ B^j &\in \mathbb{R}^{P_j \times N}, && \text{for } j = 1, \ldots, J, \\ c^j &\in \mathbb{R}^{P_j}. && \text{for } j = 1, \ldots, J. \end{aligned}$$

As before, $\mathrm{Tr}(A_*^{j,l} Y^{j,l})$ is the vector with entries $\mathrm{Tr}(A_i^{j,l} Y^{j,l})$. Furthermore, $A_i^{j,l}$ are assumed to have low rank, i.e.,

$$A_i^{j,l} = \sum_{r=1}^{\eta_{j,l}} H_{i,r}^{j,l} v_{i,r}^{j,l} (v_{i,r}^{j,l})^T, \tag{3.3}$$

where $\eta_{j,l}$ typically is less than or equal to 4. In fact, sampling normal (non-matrix) polynomial constraints will give a rank equal to 1. Note that, because $A_i^{j,l}$ is symmetric, the vectors and $H_{i,r}^{j,l}$ can be chosen to be real. Although the algorithm will work also for full rank matrices, the time complexity is quadratic in the rank. This makes the solver only useful for low rank matrices. For efficiency, a slightly different input format is also allowed. We will expand on this in Chapter 4, where this structure appears.

Note that this formulation fits into the more general case given in Chapter 2 by placing the $Y^{j,l}$ as blocks on the diagonal of Y, and $A_i^{j,l}$ in the corresponding blocks for constraints (j, i). Hence the algorithm for general SDPs with free variables can be used.

As will be shown in Chapter 4, this structure is particularly suited for polynomial (matrix) constraints which are modelled through sampling. However, the solver can in principle also be used for other problems with a low-rank structure and clustered constraints.

3.2 Speedups obtained by the structure

In this section, the speedups which can be obtained by exploiting the structure of a clustered low-rank SDP are explained. This is partly an extension of the methods used in SDPB [29].

3.2.1 Solving the system - exploiting the clustered structure

We use the decomposition of [29] to solve the system of equations for the search directions. Recall the system of equations in the general case:

$$T \begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} -d - \mathrm{Tr}(A_* Z) \\ p \end{pmatrix}, \text{ where } T = \begin{pmatrix} S & -B \\ B^T & 0 \end{pmatrix}$$

and S is defined by $S_{pq} = \mathrm{Tr}(A_p X^{-1} A_q Y)$. Recall that the clustered low-rank SDP can be seen as special case where $A_{(j,i)}$ consists of the blocks $A_i^{j,l}$ for $l = 1, \ldots, L_j$, and Y is block diagonal with blocks corresponding to (j, l).

In general, S is dense. However, in the case of clustered constraints, p and q range over tuples (j, i), and $A_{(j,i)}$ is only possibly non-zero in blocks corresponding to (j, l) for

$l = 1, \ldots, L_j$. Furthermore, Y and X are block diagonal, which means that products $X^{-1}A_{(j',i)}Y$ cannot have non-zeros on a block $j \neq j'$. Hence $S_{(j,i),(j',i)} = 0$ whenever $j \neq j'$, i.e., S has a block diagonal structure with blocks corresponding to the clusters.

As in SDPB, we can use the following decomposition for T to reduce the solving to solving several triangular systems of equations. Let CC^T be the Cholesky decomposition of S. Note that this can be computed block-wise because S is block diagonal. Then

$$\begin{pmatrix} S & -B \\ B^T & 0 \end{pmatrix} = \begin{pmatrix} C & 0 \\ B^T C^{-T} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & B^T C^{-T} C^{-1} B \end{pmatrix} \begin{pmatrix} C^T & -C^{-1}B \\ 0 & I \end{pmatrix}.$$

Because C is lower triangular, the outer matrices are lower or upper triangular and hence can be solved in quadratic time. Furthermore, the matrix $Q = B^T C^{-T} C^{-1} B$ is positive semidefinite, hence a Cholesky decomposition can be used to solve the middle matrix. Although it depends on the problem, the dimension of T (which equals $N + \sum_j P_j$) is often much larger than $\dim Q = N$, giving a speedup. Numerical conditioning can be an issue if either S or T is ill-conditioned. This is advised to be solved in SDPB by increasing the precision [20]. Note that the clustered low rank structure is not necessary for this decomposition. However, without the structure of the the clusters S would be dense, and thus the complexity would only decrease from $\mathcal{O}((P+N)^3)$ to $\mathcal{O}(P^3 + N^3)$, which does not make much of a difference when $P \gg N$.

3.2.2 Computing S - exploiting the low-rank structure

Recall that, due to the clusters, we only need to compute $S_{(j,i),(j',i')}$ for $j' = j$. Furthermore, using the linearity and the cycle property of the trace as well as the low-rank structure of the constraint matrices, we can write

$$\begin{aligned} S_{(j,i_1),(j,i_2)} &= \sum_{l=1}^{L_j} \operatorname{Tr}(A_{i_1}^{j,l} (X^{j,l})^{-1} A_{i_2}^{j,l} Y^{j,l}) \\ &= \sum_{l=1}^{L_j} \sum_{r_1=1}^{\eta_{j,l}} \sum_{r_2=1}^{\eta_{j,l}} H_{i_1,r_1}^{j,l} H_{i_2,r_2}^{j,l} \operatorname{Tr}\Big(v_{i_1,r_1}^{j,l} (v_{i_1,r_1}^{j,l})^T (X^{j,l})^{-1} v_{i_2,r_2}^{j,l} (v_{i_2,r_2}^{j,l})^T Y^{j,l} \Big) \\ &= \sum_{l=1}^{L_j} \sum_{r_1=1}^{\eta_{j,l}} \sum_{r_2=1}^{\eta_{j,l}} H_{i_1,r_1}^{j,l} H_{i_2,r_2}^{j,l} \Big((v_{i_1,r_1}^{j,l})^T (X^{j,l})^{-1} v_{i_2,r_2}^{j,l} \Big) \Big((v_{i_2,r_2}^{j,l})^T Y^{j,l} v_{i_1,r_1}^{j,l} \Big). \end{aligned}$$

This shows that $S_{(j,i_1),(j,i_2)}$ is a sum over products of so-called bilinear pairings $u^T M v$, which can be precomputed. Furthermore, the precomputing can be done efficiently because the bilinear pairings are of the form $v_i^T M v_j$. For each v_j, we can compute Mv_j (with the cost quadratic in the length of v_j), after which we can compute a simple vector inner product for each v_i. Hence with P vectors of size M, the cost will be $\mathcal{O}(PM^2 + P^2 M)$. Note that the number of bilinear pairings is quadratic in the rank of the constraint matrices $A_i^{j,l}$.

3.3 Time complexity

In this section, the time complexity for the main steps of CLRS will be determined, as well as the complexity for SDPA-GMP. The complexities will be compared in Section 3.3.3.

We will use the notation introduced in Section 3.1: Let N be the number of free variables, which connect the clusters indexed by $j = 1, \ldots, J$. Let cluster j consist of P_j constraints, with L_j matrix variables. Let $U_{j,l}$ and $\eta_{j,l}$ denote the sizes respectively the rank of the constraint matrices.

In practice, N, P_j and $U_{j,l}$ are often related, and L_j and $\eta_{j,l}$ are fixed. This yields a more compact expression for a specific problem, giving more insight in how the solving time scales with the problem size in one or two variables.

3.3.1 CLRS

The complexities of the most important operations in CLRS are summarized in Table 1. Due to the block structure of the Schur complement matrix S, nearly all operations on different constraints j are completely separate, except for the Cholesky decomposition of the matrix Q. Therefore, the complexity of all other operations are given per constraint. Note that the separation of the constraints allows for easy parallel computations.

To compute S, one needs to compute $\sum_{l=1}^{L_j} \eta_{j,l}^2 P_j^2$ bilinear pairings for each cluster j. As noted in Section 3.2.2, this can be done by computing for each constraint the products $Y^{j,l} v_{i,r}^{j,l}$ after which the products can be combined into bilinear pairings through vector inner products. This takes a total of $\mathcal{O}(\sum_{l=1}^{L_j} \eta_{j,l} P_j U_{j,l}^2)$ operations for the matrix vector products, and $\mathcal{O}(\sum_{l=1}^{L_j} \eta_{j,l}^2 P_j^2 U_{j,l})$ operations for the vector inner products.

The matrix S has blocks of size $\mathcal{O}(P_j)$. A full Cholesky decomposition of an $n \times n$ matrix costs $\frac{1}{3}n^3$ operations, resulting in $\mathcal{O}(P_j^3)$ operations for a block S_j. Computing $C_j^{-1} B_j$ amounts to solving N triangular systems, one for each column of B_j, where C_j is the j-th Cholesky block of S. This costs in total $\mathcal{O}(N(\dim C_j)^2) = \mathcal{O}(N P_j^2)$ operations. Multiplying the resulting matrix with its transpose costs $\mathcal{O}(N^2 P_j)$ operations, obtaining the matrix Q of size $N \times N$. The final Cholesky decomposition of Q costs $\frac{1}{3}N^3$ operations. Solving the system can now be done by back- and forward substitution, which has a complexity of $\mathcal{O}(N^2 + \sum_j P_j^2)$, which is subdominant to the cost of computing $C_j^{-1} B_j$ and Q. Therefore this will not be taken into account in the final complexity.

Summarizing, this gives a total complexity of

$$\mathcal{O}\Big(N^3 + \sum_{j=1}^{J} (N^2 P_j + N P_j^2 + P_j^3 + \sum_{l=1}^{L_j} (\eta_{j,l} P_j U_{j,l}^2 + \eta_{j,l}^2 P_j^2 U_{j,l}))\Big).$$

3.3.2 SDPA-GMP

In SDPA-GMP, the variables y need to be incorporated in the variable matrix Y as a diagonal block. Therefore, the matrix $S_{pq} = \mathrm{Tr}(A_p X^{-1} A_q Y)$ becomes dense; for each

Operation	Complexity	Reason
Matrix vector products per (j,l)	$\mathcal{O}(\eta_{j,l}P_jU_{j,l}^2)$	$\eta_{j,l}P_j$ products of dimension $U_{j,l}$
Vector inner products per (j,l)	$\mathcal{O}(\eta_{j,l}^2P_j^2U_{j,l})$	$\eta_{j,l}^2P_j^2$ products of dimension $U_{j,l}$
Forming S_j	$\mathcal{O}(\sum_{l=1}^{L_j}(\eta_{j,l}P_jU_{j,l}^2+\eta_{j,l}^2P_j^2U_{j,l}))$	
Cholesky decomposition $S_j=C_jC_j^T$	$\mathcal{O}(P_j^3)$	Per block of size $\mathcal{O}(P_j)$
$C^{-1}B$	$\mathcal{O}(NP_j^2)$	N triangular systems with P_j variables
$Q=B^TC^{-T}C^{-1}B$	$\mathcal{O}(N^2P_j)$	Matrix sizes $N\times P_j$ and $P_j\times N$
Cholesky decomposition of Q	$\mathcal{O}(N^3)$	$\dim Q=N$
Solving the complete system	$\mathcal{O}(N^2+\sum_j P_j^2)$	triangular systems of size N and P_j
Total complexity	$\mathcal{O}\Big(N^3+\sum_{j=1}^{J}(N^2P_j+NP_j^2+P_j^3+\sum_{l=1}^{L_j}(\eta_{j,l}P_jU_{j,l}^2+\eta_{j,l}^2P_j^2U_{j,l}))\Big)$	

Table 1: Complexity per iteration for CLRS. Each row gives the complexity for one operation, acting on one constraint j (except for the Cholesky decomposition of Q, which combines all constraints), possibly using earlier formed expressions. This is accumulated in the last row. Because solving the system is subdominant to computing $C^{-1}B$ and Q in all cases, it is not included in the total complexity.

pair (p,q), the diagonal block is used, whereas for CLRS this is encoded in the B matrix. The complexities for computing S and calculating the Cholesky decomposition of S are summarized in Table 2, where the cost of computing S is split into contributions of the diagonal block and the remainders of the blocks.

Computing an element S_{pq} requires computing $X^{-1}(A_qY)$. This cost $\mathcal{O}(N)$ operations for the diagonal block for any combination of tuples $p=(j_1,i_1)$ and $q=(j_2,i_2)$. When $j_1\neq j_2$, the nonzero blocks of A_p and A_q do not overlap except for the diagonal block. Therefore SDPA-GMP does not calculate the corresponding matrix products, but only the diagonal block [33]. The diagonal block has to be used for every pair of tuples. This costs a total of $\mathcal{O}(N(\sum_j P_j)^2)$ operations, $\mathcal{O}(N)$ for each pair of tuples p,q. When $j_1=j_2=j$, both have L_j blocks at the same positions. These blocks each have a size of $U_{j,l}\times U_{j,l}$, giving a total complexity of $\mathcal{O}(\sum_{l=1}^{L_j}U_{j,l}^3)$ for the product $X^{-1}A_qY$, disregarding the cost of the diagonal block. The product can be computed per row q, and then be used to compute $\langle X^{-1}A_qY,A_p\rangle$ for each column in that row. Hence the computational cost is $\mathcal{O}(\sum_{l=1}^{L_j}(U_{j,l}^3P_j+P_j^2U_{j,l}^2))$ for a constraint j: P_j rows, each needing a product of three matrices of size $U_{j,l}\times U_{j,l}$, and the inner product costs $\mathcal{O}(U_{j,l}^2)$ per pair of constraints with the same j.

To solve the system, a Cholesky decomposition is used. This costs $\mathcal{O}(\dim S^3)=\mathcal{O}((\sum_j P_j)^3)$ operations because S is dense. In total, this gives a time complexity of

$$\mathcal{O}(N(\sum_j P_j)^2+\sum_j\sum_{l=1}^{L_j}P_j(U_{j,l}^3+P_jU_{j,l}^2))+(\sum_j P_j)^3)\,.$$

This complexity analysis uses the dense mode of SDPA-GMP. In case the matrices $A_i^{j,l}$ are sparse, the computational cost is decreased. SDPA-GMP uses three different formulas to compute the elements of the Schur complement matrix S, depending on the sparsity

of the constraint matrices. As some problems, especially polynomial constraints, can be modelled as an SDP with sparse constraint matrices, the complexity corresponding to calculating the Schur complement matrix might not be fully applicable anymore. Constraint matrices with $\mathcal{O}(U_{j,l})$ elements may reduce the complexity of computing S by a factor $U_{j,l}$. As accessing elements in a sparse matrix format has some extra overhead, the constant in front of the expressions may be increased. Furthermore, it is well possible that not all matrices are sparse, thus that the practical time complexity of computing S with SDPA-GMP is between the given complexity and the reduced complexity.

Operation	Complexity	Reason
Contribution of diagonal blocks to S	$\mathcal{O}(N(\sum_j P_j)^2)$	$(\sum_j P_j)^2$ tuples with cost N
Products $X^{-1}A_qY$, no diagonal block	$\mathcal{O}\big(P_j \sum_{l=1}^{L_j} U_{j,l}^3\big)$	L_j blocks of size $U_{j,l} \times U_{j,l}$
Matrix inner products $\langle A_p, X^{-1}A_qY\rangle$, no diagonal block	$\mathcal{O}\big(P_j^2 \sum_{l=1}^{L_j} U_{j,l}^2\big)$	L_j blocks with $U_{j,l}^2$ elements per A_p
Forming S	$\mathcal{O}\big(N(\sum_j P_j)^2 + \sum_j \sum_{l=1}^{L_j} U_{j,l}^3 P_j + P_j^2 U_{j,l}^2)\big)$	
Cholesky decomposition of S	$\mathcal{O}((\sum_j P_j)^3)$	$\dim S = \sum_j P_j$
Total complexity	$\mathcal{O}(N(\sum_j P_j)^2 + \sum_j \sum_{l=1}^{L_j}(P_j U_{j,l}^3 + P_j^2 U_{j,l}^2) + (\sum_j P_j)^3)$	

Table 2: Complexity per iteration for SDPA-GMP. Each row gives the complexity for one operation, possibly using earlier formed expressions. This is accumulated in the last row. It should be noted that the second term may differ in the case of sparse constraint matrices.

3.3.3 Comparison between CLRS and SDPA-GMP

The first thing to notice when comparing the complexities of CLRS and SDPA-GMP for clustered low-rank SDPs is that the constraints are completely separated in CLRS, whereas they interfere in SDPA-GMP. This gives products of sums over j in the complexity of SDPA-GMP. This can be especially important for programs with a large number of clusters, each with a moderate to large number of constraints.

Furthermore, the factor $U_{j,l}^3 P_j + U_{j,l}^2 P_j^2$ (in SDPA-GMP) is decreased to $U_{j,l}^2 P_j + U_{j,l} P_j^2$ at the expense of extra terms with higher powers of N (in CLRS). As noted before, this term in the complexity of SDPA-GMP can be reduced in the case of sparse constraint matrices.

The required precision also needs to be considered. Calculations with a factor 2 more precision cost about twice as long, hence a larger precision gives an overall slow down. The solver CLRS may require a higher precision, because in addition to the matrix T, which is of similar conditioning as the Schur complement matrix in SDPA-GMP, the matrix Schur complement matrix S of the specialized solver can have a bad conditioning. Experiments show that polynomial constraints may require a precision of at least 350 bits, while SDPA-GMP requires up to 200 bits [23].

Lastly, in CLRS, nearly everything can be easily parallellized because of the block structures. Although parallellization is also possible for computing the Schur complement matrix in SDPA-GMP, it has not been implemented.[1] As most, if not all, current

[1] In several other solvers of the SDPA family, the computation of S is parallellized [33]

computers have multiple cores, parallellization is a great asset of a solver.

4. Multivariate polynomial (matrix) programs

In this chapter, we introduce the multivariate polynomial matrix program (MPMP). The notation is partly similar to the notation of Simmons-Duffin in [29], where a specific case (univariate polynomial matrix programs on $\mathbb{R}_+$ (PMP)) was introduced. In addition, we derive the translation to a clustered low-rank SDP, using a form of Putinar's theorem for polynomial matrices originally proven by Scherer and Hol [27, 19]. This leads to a relaxation of the problem, with a convergence guarantee.

Although everything is done with matrices, it should be noted that everything still holds for polynomial constraints, as well as 'polynomial' matrix constraints where $n = 0$. The case of polynomial constraints is already widely used.

4.1 Preliminaries

Let $\mathbb{R}[x]$ denote the ring of polynomials, and $\mathbb{R}[x]^{m\times m}$ the ring of polynomial matrices of size $m \times m$ in variables $x = (x_1, \ldots, x_n)$. Unless otherwise stated, we assume that $x \in \mathbb{R}^n$. Let $G \subseteq \cup_m \mathrm{Sym}(\mathbb{R}[x]^{m\times m})$, where $\mathrm{Sym}(\mathbb{R}[x]^{m\times m})$ is the set of all symmetric $m \times m$ polynomial matrices in n variables. Define $\mathcal{S}^m \coloneqq \mathrm{Sym}(\mathbb{R}^{m\times m})$, the special case where $n = 0$, and let $\mathcal{S}^m_{\succeq 0}$ denote the set of all $m \times m$ positive semidefinite matrices.

We define the semi-algebraic set generated by G as

$$S_G \coloneqq \{x \in \mathbb{R}^n \mid g(x) \succeq 0 \, \forall g \in G\}\,.$$

The quadratic module generated by G is given by

$$M_G \coloneqq \mathrm{Cone}(\{p^T g p \mid g \in G \cup \{1\}, p \in \mathbb{R}[x]^{t_g \times m}\})\,,$$

where t_g is the size of the corresponding g. M_G is said to be archimedean if

$$\forall p \in \mathbb{R}[x]^{m\times m} \; \exists N \in \mathbb{N} : NI - p^T p \in M_G\,.$$

This is equivalent to the definition which can be used to verify that a quadratic module M_G is archimedean: $\exists N \in \mathbb{N} : (N - \sum_i x_i^2)I \in M_G$, see [19, Lemma 8].

A polynomial matrix $f \in \mathrm{Sym}(\mathbb{R}[x]^{m\times m})$ is positive semidefinite (positive definite) on S if $f(x)$ is positive semidefinite (positive definite) for all $x \in S$. This is denoted by $f \succeq 0$ ($f \succ 0$) on S.

4.2 The multivariate polynomial matrix program

Let $G_j \subseteq \cup_t \mathrm{Sym}(\mathbb{R}[x]^{t\times t})$ be finite for $j = 1, \ldots, J$, $P_i^j \in \mathrm{Sym}(\mathbb{R}[x]^{m\times m})$ for $i = 1, \ldots, N$ and $j = 1, \ldots, J$, and $b \in \mathbb{R}^N$. We define the multivariate polynomial matrix program

(MPMP) as

$$\begin{aligned} \sup \quad & b^T y && \text{with } y \in \mathbb{R}^N \\ \text{s.t.} \quad & P_y^j \coloneqq P_0^j + \sum_{i=1}^{N} y_i P_i^j \succeq 0 && \text{on } S_{G_j}, \text{ for } j = 1, \ldots, J\,. \end{aligned} \tag{4.1}$$

For $n = 1$, $G_j = \{x\}$, this is the PMP introduced in [29]. When $n = 0$, this is a normal primal semidefinite program [32] except for the objective which maximizes instead of minimizes.

We call program (4.1) strictly feasible if there is a feasible solution y such that $P_y^j \succ 0$ on S_{G_j} for all $j = 1, \ldots, J$. Note that, for any feasible y_f and strictly feasible y_s, a conic combination $\alpha y_f + \beta y_s$ is a strictly feasible solution whenever $\beta > 0$.

4.3 Relaxation of the program

The translation of (4.1) to an SDP relies on the following theorem, originally proven by Scherer and Hol in [27]. We give a detailed proof following [19] in Appendix A. Note that $f \in M_G$ implies that $f \succeq 0$ on S_G.

Theorem 4.1 ([27, 19])**.** *Let $f \in \mathbb{R}[x]^{m\times m}$, $G \subseteq \mathbb{R}[x]$. Suppose M_G is archimedean. If $f \succ 0$ on S_G, then $f \in M_G$.*

In the univariate case, the theorem can be strengthened for S_G consisting of $\mathbb{R}$, $\mathbb{R}_{\geq a}$ or $[a, b]$, when reducing G to its simplest form. In these cases we have equivalence between $f \in M_G$ and $f \succeq 0$ on S_G. See Appendix B for the relevant theorems and proofs.

Theorem 4.1 allows us to approximate the constraints by elements of M_G, which can in turn be approximated by elements of the truncated quadratic module

$$M_{G,d} \coloneqq \operatorname{Cone}(\{F^T g F \mid g \in G \cup \{1\},\, F \in \mathbb{R}[x]^{t_g \times m},\, \deg(F^T g F) \leq d\})\,.$$

In this expression, we use $\deg(F)$ for the maximum total degree over all elements of a polynomial matrix F. This gives the relaxed problem:

$$\begin{aligned} \sup \quad & b^T y && \text{with } y \in \mathbb{R}^N \\ \text{s.t.} \quad & P_y^j = P_0^j + \sum_{i=1}^{N} y_i P_i^j \in M_{G_j,d} && \text{for } j = 1, \ldots, J\,. \end{aligned} \tag{4.2}$$

For $d \to \infty$, we have convergence of value of the relaxation to the value of the original problem.

Theorem 4.2. *Let $p^* < \infty$ and p_d^* denote the optimal values of program* (4.1) *and* (4.2) *respectively, with M_{G_j} archimedean for all j. Suppose* (4.1) *is strictly feasible. Then:*

- *for all $d \in \mathbb{N}$, $p_d^* \leq p_{d+1}^* \leq p^*$*
- *$\lim_{d\to\infty} p_d^* = p^*$*

Proof. Let $d \in \mathbb{N}$. Because $M_{G,d} \subseteq M_{G,d+1}$, we have $p_d^* \leq p_{d+1}^*$. Furthermore, because $f \in M_{G,d}$ implies that $f \succeq 0$ on S_G, any feasible solution for p_d^* is also feasible for p^*. Hence $p_d^* \leq p^*$ for all d.

Let $\varepsilon > 0$, and let $y \in \mathbb{R}^N$ be a strictly feasible, ε-optimal solution for p^*. This is possible, because the feasible region is convex, and there is a strictly feasible solution by assumption. By Theorem 4.1, $P_y^j(x) = P_0^j(x) + \sum_i y_i P_i^j(x) \in M_{G_j}$ for all j. Hence there is a d such that $P_y^j(x) \in M_{G_j,d}$ for all j, which means that there is a d such that y is feasible for problem (4.2). Furthermore, we have $p^* - p_d^* \leq \varepsilon$, which means that $p_d^* \to p^*$. □

In practise, it is possible that a problem is not strictly feasible. This can be solved by adding εI to the constant terms P_0^j for all j, making any solution of the original problem strictly feasible in the new problem. For decreasing ε, this converges to the original problem.

4.4 Translation to a clustered low-rank SDP

We will rewrite program (4.2) to the form (3.1); the clustered low-rank SDP. Each polynomial matrix constraint will form a cluster of constraints, and the y variables will be the free variables connecting the constraint.

We consider f of the form $F^T g F$ for some $g \in \mathbb{R}[x]^{t\times t}$ and $F \in \mathbb{R}[x]^{t\times m}$. For individual elements f_{ij} we have

$$f_{ij} = \operatorname{Tr}(fE_{ij}) = \operatorname{Tr}(F^T g F E_{ij}) = \operatorname{Tr}(gF_i F_j^T)\,,$$

where F_i is the i-th column of F and E_{ij} the standard basis of $\mathbb{R}^{m\times m}$.

Define $P \coloneqq \sum_{i,j} F_i F_j^T \otimes E_{ij} = \operatorname{vec}(F)\operatorname{vec}(F)^T \in \mathbb{R}[x]^{mt\times mt}$, with $\operatorname{vec}(F)$ being the column vector of all columns of F. Then

$$f = \operatorname{Tr}_t((g \otimes I_m)P)\,,$$

where Tr_t denotes the partial trace over the first tensor factor, the linear extension of the mapping $\operatorname{Tr}_t : \mathbb{R}^{t\times t} \otimes \mathbb{R}^{m\times m} \to \mathbb{R}^{m\times m}$ defined by $\operatorname{Tr}_t(A \otimes B) = \operatorname{Tr}(A)B$.

Let $q_i(x)$ be a basis for the n-variate polynomials up to (total) degree $\delta = \deg(F)$, and $q_\delta(x)$ the vector of these polynomials, of length $n(\delta)$. Using that $\operatorname{vec}(F)^T = Z(q_\delta(x) \otimes I_{mt})$ for some $Z \in \mathbb{R}^{1\times n(\delta)mt}$ gives $P = (q_\delta(x) \otimes I_{mt})^T Z^T Z (q_\delta(x) \otimes I_{mt})$, which can be used to write

$$\begin{aligned} f &= \operatorname{Tr}_t((g \otimes I_m)P) \\ &= \operatorname{Tr}_{n(\delta)t}(Y((q_\delta(x) \otimes I_t)g(x)(q_\delta(x) \otimes I_t)^T \otimes I_m)) \\ &= \operatorname{Tr}_{n(\delta)t}(Y(Q_{g,\delta}(x) \otimes I_m))\,, \end{aligned} \tag{4.3}$$

where $Y = Z^T Z$ is positive semidefinite of rank 1 and $Q_{g,\delta}(x) = (q_\delta(x) \otimes I_t)g(x)(q_\delta(x) \otimes I_t)^T$. Recalling that Y can be decomposed into $Z^T Z$ for any positive semidefinite matrix

Y gives equivalence between the two forms of f (cf. Lemma 1 of [27]). We can rewrite Q to obtain a slightly simpler form. As $g(x_k) = I_1 \otimes g(x_k)$, we have

$$Q_{g,\delta}(x) = (q_\delta(x) I_1 q_\delta(x)^T) \otimes (I_t g(x) I_t^T) = (q_\delta(x_k) q_\delta(x)^T) \otimes g(x)\,.$$

Now let $f \in M_{G,\delta}$, similar to the constraints of program (4.2). Then we have

$$f = \sum_{g \in G \cup \{1\}} \sum_{i \in \mathcal{I}_g} F_{g,i}^T g F_{g,i}\,,$$

where $\mathcal{I}_g$ denotes the set of indices for matrices $F_{g,i}$ corresponding to $g \in G$. Using (4.3) per $g \in G$, we obtain

$$f(x) = \sum_{g \in G \cup \{1\}} \mathrm{Tr}_{n(\delta)t}(Y_g (Q_{g,\delta}(x) \otimes I_m))\,.$$

Note that, due to linearity of the trace, only one variable matrix is needed per $g \in G$, and Y_g is not necessarily rank 1 anymore.

Let $x_1, \ldots, x_K \in \mathbb{R}^n$ be a unisolvent set of points for polynomials up to degree δ, i.e., a set such that the polynomials up to degree δ are uniquely determined by the values on $x_1, \ldots, x_K$. Equality of matrices is equality per entry, thus this gives the constraints

$$(P_y(x_k))_{rs} = \sum_{g \in G \cup \{1\}} \mathrm{Tr}(Y_g (Q_{g,\delta}(x_k) \otimes \bar{E}_{rs}))\,,$$

where $\bar{E}_{rs} = \frac{1}{2}(E_{rs} + E_{sr})$ are the standard basis matrices of $\mathrm{Sym}(\mathbb{R}^{m \times m})$.

As the Y_g are different for different constraints j, this is exactly the form of (3.1). Denoting Y_g^j by $Y^{j,l}$ for $G_j \cup \{1\} = \{g_{j,l} \mid l = 1, \ldots, L_j\}$ gives the constraint matrices

$$\begin{aligned} A^{j,l}_{(r,s,k)} &= Q_{g_{l,j},\delta_j}(x_k) \otimes \bar{E}_{rs}\,, \\ B^j_{(r,s,k),i} &= -(P_i^j(x_{k,j}))_{rs}\,, \\ c^j_{(r,s,k)} &= P_0^j(x_{k,j})_{rs}\,, \\ C &= 0\,. \end{aligned} \tag{4.4}$$

In the case that $A^{j,l}_{(r,s,k)}$ is of low rank, this is the structure required by the customized version of SDPB. In these equations, we have $j = 1, \ldots, J$, $1 \le r \le s \le m$, $k = 1, \ldots, K_j = n(\delta_j)$ and $1 \le i \le N$.

In the next part, we focus on two aspects which are important for computations. First of all, a low rank is preferred due to the the time complexity being quadratic in the rank. The term $\bar{E}_{rs}$ has either rank one or rank two, which is fixed for a certain element of the initial constraint matrix, but the rank of $Q_{g,\delta}(x_k)$ depends on g and thus is part of the input. As the problem depends on S_G, g can be modified as long as S_G stays the same. Recall that, with $g \in \mathbb{R}[x]^{t \times t}$,

$$Q_{g,\delta}(x_k) = (q_\delta(x_k) q_\delta(x_k)^T) \otimes g(x_k)\,.$$

Using this together with $\text{rank}(A \otimes B) = \text{rank}(A)\text{rank}(B)$, we have

$$\text{rank}(Q_{g,\delta}(x_k)) = \text{rank}(g)\,,$$

hence $Q_{g,\delta}(x_k)$ is of rank 1 exactly when $g(x_k)$ is of rank 1. Note that this is certainly the case when g is a polynomial.

For general $g \in \mathbb{R}[x]^{t\times t}$ with $t > 1$, $Q_{g,\delta}(x_k)$ will not be of rank 1. However, by taking $G' \subset \mathbb{R}[x]$ consisting of all principal minors for all $g \in G$, we obtain $S_{G'} = S_G$ with

$$Q_{g',\delta}(x_k) = g'(x_k)q_\delta(x_k)q_\delta(x_k)^T \tag{4.5}$$

of rank 1 for all $g' \in G'$. In some cases, the number of minors which need to be included is equal to one, because the other minors are trivially nonnegative. A comparison in time complexity using Section 3.3 gives that expressing g in terms of minors costs a factor $t^3/2^{t-1}$ less to compute the contribution of g to S_j compared to using the matrix g, where $t = \text{rank}(g)$. For $t \leq 11$, this is larger than 1, hence for most practical purposes using the principal minors is preferable above using a polynomial matrix as weight.

Secondly, the total degree of $F^T gF$ should be at most d, because we are trying to model elements of $M_{G,d}$. This can be obtained by taking $\delta = \lfloor \frac{d-\deg(g)}{2} \rfloor$ as maximum total degree of F. Note that the minors of a matrix g are in general of higher degree than g itself. Hence taking the minors instead of the matrices increases the number of the variable blocks $Y_{l,j}$, but decreases the size $n(\delta)mt$ per variable, for a maximum degree d. This increases the preference of using the principal minors of g instead of g itself even further.

4.5 Modifications of the solver for MPMPs

Recall that the constraint matrices for the clustered low-rank SDP solver CLRS introduced in Chapter 3 were required to be of the form

$$A_i^{j,l} = \sum_{r=1}^{\eta_{j,l}} H_{i,r}^{j,l} v_{i,r}^{j,l} (v_{i,r}^{j,l})^T\,,$$

with $\eta_{j,l}$ relatively low. However, the constraint matrices resulting from the MPMPs are of the form

$$A_{(i,r,s)}^{j,l} = H_i v_i v_i^T \otimes \bar{E}_{r,s}\,, \tag{4.6}$$

where $\bar{E}_{r,s} = \frac{1}{2}(e_r e_s^T + e_s e_r^T)$ and we suppressed the superscript j, l. We can write $A_i^{j,l}$ as rank 2 matrix with eigenvectors

$$v_i \otimes \frac{1}{2}(e_r + e_s)$$

and

$$v_i \otimes \frac{1}{2}(e_r - e_s)\,.$$

This will increase the sizes of the matrices by a factor m. Even if matrix vector products are performed faster using the sparsity in the second tensor factor, we can increase the speed by directly using $\bar{E}_{r,s} = \frac{1}{2}(e_r e_s^T + e_s e_r^T)$. Let us consider computing an element of the Schur complement matrix S for $A_i^{j,l}$ of the form (4.6). We have, similar to exploiting the low rank structure:

$$\begin{aligned}
S_{(j,i_1,r_1,s_1),(j,i_2,r_2,s_2)} &= \sum_{l=1}^{L_j} \operatorname{Tr}\Big((H_{i_1} v_{i_1} v_{i_1}^T \otimes \bar{E}_{r_1,s_1})(X^{-1})(H_{i_2} v_{i_2} v_{i_2}^T \otimes \bar{E}_{r_2,s_2}) Y_{l,j}\Big) \\
&= \sum_{l=1}^{L_j} \frac{H_{i_1} H_{i_2}}{4} (v_{i_1} \otimes e_{s_1})^T (X^{-1})(v_{i_2} \otimes e_{r_2})(v_{i_2} \otimes e_{s_2})^T Y (v_{i_1} \otimes e_{r_1}) \\
&+ (r_1 \leftrightarrow s_1) + (r_2 \leftrightarrow s_2) + (r_1 \leftrightarrow s_1, r_2 \leftrightarrow s_2) \\
&= \sum_{l=1}^{L_j} \frac{H_{i_1} H_{i_2}}{4} v_{i_1}^T (X^{-1})^{s_1,r_2} v_{i_2} v_{i_2}^T Y^{s_2,r_1} v_{i_1} \\
&+ (r_1 \leftrightarrow s_1) + (r_2 \leftrightarrow s_2) + (r_1 \leftrightarrow s_1, r_2 \leftrightarrow s_2)\,,
\end{aligned}$$

where we again suppressed the superscript j, l everywhere for clarity. This reduces products of (effectively) size $2U_{j,l}$ to products of size $U_{j,l}$. Note that, even though four terms are required, this equals the number of terms required for a normal rank 2 matrix. This directly generalizes to higher rank matrices with a factor $\otimes E_{rs}$. In the complexity for the specialized solver, this gives $P_j \to \frac{m_j(m_j+1)}{2} P_j$ and $\eta_{j,l} \to 2\eta_{j,l}$.

For SDPA-GMP, this leads to constraint matrices of m_j times the size, i.e., $U_{j,l} \to m_j U_{j,l}$. Because the constraint matrices become sparse with $\mathcal{O}(U_{j,l}^2)$ elements, the time complexity for the matrix inner product increases by a maximum factor of 2 when $r \neq s$. Whether the matrix multiplications are done in dense or sparse mode may depend on m_j. For small m_j, the dense mode may be used, giving a complexity for $m_j^5 P_j U_{j,l}^3$ for the matrix multiplications instead of $P_j U_{j,l}^3$. In the case of coefficient matching, the constraint matrices used in SDPA-GMP can become sparse with $\mathcal{O}(mU_{j,l})$ elements, depending on the polynomial basis used.

4.6 Sample points

Originally, the sample points were required to be in S_G to use SDPB. This requirement is removed in CLRS, by making it possible to give $Q = \sum_r H_r v_r v_r^T$ as input. Hence the unisolvent set of sample points can be chosen to be a known unisolvent set of good quality for polynomial interpolation.

Intuitively, a set of sample points should be in or closely around the set S_G. In that case, it is less likely that a small error of the value in one sample point will have a large influence on the values in S_G, and thus on the feasibility of the program given the floating point result. Furthermore, sample points should not be very close together, which would give nearly linearly dependent constraints and thus ill conditioning of the problem.

Although we focus on choosing sample points in this section, choosing a good basis can give additional improvements.

4.6.1 Checking whether a set is unisolvent

Let $X = \{x_k \mid k = 1, \ldots, K = \binom{n+d}{n}\}$ be a set of points, and let $q_1(x), \ldots, q_K(x)$ be a basis of polynomials for the space of n-variate polynomials of degree at most d. The set X is unisolvent if a polynomial p is uniquely determined by its values on these sample points, i.e., the system of equations

$$\begin{pmatrix} q_1(x_1) & \cdots & q_K(x_1) \\ \vdots & \ddots & \vdots \\ q_1(x_K) & \cdots & q_K(x_K) \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_K \end{pmatrix} = \begin{pmatrix} p(x_1) \\ \vdots \\ p(x_K) \end{pmatrix} \tag{4.7}$$

has a unique solution. This is the case precisely when the Vandermonde matrix $(q_i(x_k))_{k,i=1}^{K}$ is non-singular. The solution c gives the coefficients of p when expressed in the basis q. Hence checking whether a set of sample points is unisolvent is equivalent with checking whether the determinant of the Vandermonde matrix equals zero. A small determinant is related to an ill conditioned system: a small error in the polynomial evaluations can lead to a large error in the coefficients. Hence if two polynomials are nearly equal on the sample points, they can still have a large difference on other points in S_G. Thus a combination of sample points and basis with a large condition number of the Vandermonde matrix are preferred.

4.6.2 Special cases

For several cases of n and a domain S, a good set of interpolation points is known. In this section, we will review several of such sets.

Chebyshev points

Let $n = 1$, with domain $S = [-1, 1]$. Note that any domain $[a, b]$ can be obtained through an affine transformation. In this case, a well known set of sample points is the set of Chebyshev points. For $d \in \mathbb{N}$, the points are defined as the zeros of the Chebyshev polynomial T_{d+1}, given by

$$x_k^d = \cos\Big(\frac{2k-1}{2(d+1)}\pi\Big) \qquad \text{for } k = 1, \ldots, d+1\,.$$

Another related set of sample points are the Chebyshev-Gauss-Lobatto points

$$z_j^d = \cos\Big(\frac{j-1}{d}\pi\Big) \qquad \text{for } j = 1, \ldots, d+1\,.$$

Padua points

Let $n = 2$, and $S = [-1, 1]^2$. Again, a set of sample points for the more general set $[a, b] \times [c, d]$ can be obtained by an affine transformation. The Padua points can be defined via the Chebyshev-Gauss-Lobatto points [6]. Let z_j^d be defined as before, and let

$$C_{d+1}^E = \{z_j^d \mid j - 1 \text{ is even}\}\,,$$
$$C_{d+1}^O = \{z_j^d \mid j - 1 \text{ is odd}\}\,.$$

The Padua points are

$$P_d = (C^E_{d+1} \times C^O_{d+2}) \cup (C^O_{d+1} \times C^E_{d+2})\,.$$

Note that this is a subset of the product set $C_{d+1} \times C_{d+2}$.

Rational points in the unit simplex

For general n, d, the set of rational points in the unit simplex with denominator d is a unisolvent set [22]. Again, an affine transformation is possible for other simplex-like sets S. However, the conditioning of the Vandermonde matrix is in general bad compared to for example Chebyshev points for $n = 1$ and Padua points for $n = 2$.

Random sample points

In a general set S_G, one can also take random sample points. These points are unisolvent with probability 1, but are likely to have bad conditioning.

4.6.3 Approximate Fekete points

Given a basis q for n-variable polynomials up to degree $2d$, the Fekete points in a compact domain $S \in \mathbb{R}^n$ are defined as the K points x_k that maximize the absolute value of the determinant of the Vandermonde matrix V. Note that this means that the set of points x_k is unisolvent. Furthermore, the Fekete points are independent of the basis, as a basis change multiplies the determinant with a constant factor.

Although Fekete points are well defined for any compact domain, they are hard to compute in general. Therefore, we can use approximate Fekete points instead, as suggested in [25, 30]. Given a (large) set of sample points, we can choose a square submatrix of the corresponding rectangle Vandermonde matrix which approximately maximizes the absolute value of the determinant. This problem is in general NP-hard [18], but the approach taken by [30] can work well in practice. However, the cost scales with the square of the dimension of the vectorspace of polynomials, i.e., approximately with $d^n/n!$.

4.6.4 Greedy maximization of the Vandermonde determinant

A more straightforward approach, also given in [30], is to perform a greedy maximization of the determinant of the resulting Vandermonde matrix: Iteratively choose columns from the matrix. In each iteration, the column with maximum norm is chosen, and the orthogonal projection of that column onto the columns of the matrix is subtracted from the matrix.

Both approaches can be used in combination with product sets of good sample points for $n = 1$ (e.g., the product set of Chebyshev points $C_{d+1} \times \ldots \times C_{d+n}$) to obtain a set of good quality for general n while reducing the time needed to compute them.

4.6.5 Symmetrized sampling

In the multivariate setting, it is sometimes possible to use symmetry in order to reduce the length of the basis and the required number of sample points. See for example the three-point bounds in Chapter 6. We will assume that the group inducing the symmetries is a permutation group G.

Naively, one might use the previous explained approaches to choose sample points, using a symmetrized basis. However, this may introduce (implicit) linear dependencies, which do not work well in most SDP solvers, including CLRS. Let us consider the case $n = 3$ with symmetry group S_3, i.e., the polynomials are invariant under all permutations of coordinates. Given a sample point (u, v, t), one can see this as representing the points $\sigma(u, v, t)$ for all $\sigma \in S_3$. Hence when the coordinates are distinct, this represents 6 points, when two of the three coordinates are the same, this represents 3 points and when $u = v = t$, this represents 1 point. As a non symmetric polynomial of total degree d is completely determined by $\binom{n+d}{d}$ sample points, we need the sum of the weights of the sample points to be equal to $\binom{n+d}{d}$:

$$\sum_{x \in X} w(x) = \binom{n+d}{d},$$

where $w(x)$ is the number of sample points the point x represents. Note that this means that $S_3X = \{\sigma x \mid \sigma \in S_3, x \in X\}$ is a unisolvent set for nonsymmetric n-variate polynomials of degree d.

In addition, it is natural to require the same amount of (symmetric) sample points as the number of basis polynomials, as this is one requirement for the system (4.7) to have a unique solution without explicit linear dependencies. This gives restrictions on the sets of sample points one can use, and using the approximate Fekete points obtained from using a symmetric basis in the computations does in general not work.

There are several (heuristical) approaches to solve this. First of all, one can use the rational points in the unit simplex with denominator d as base set, and take one point of each equivalence class, where two points x, y are equivalent if there is a group element σ such that $x = \sigma y$. In case of $G = S_n$, this means that we take all points with $x_1 \leq x_2 \leq \ldots \leq x_n$. As the simplex is invariant under permutations, this always gives a set with the right properties.

As mentioned before, this set is not a particularly good set of sample points. Another approach is to take the equivalent of the unit simplex out of the set X^n, where X is a unisolvent set for $n = 1$ for polynomials up to degree d. This amounts to taking the points $(x_{j_1}, \ldots, x_{j_n})$ where $x_j \in X$ and

$$\sum_{k=1}^{n} j_k - 1 \leq d\,.$$

This set can be symmetrized similarly to the unit simplex, but may perform better.

A third approach would be to fix a certain number of points (for example the points with $w(x) \leq 3$), and use the greedy maximization of the submatrix determinant to determine the best remaining points. This can be done more generally, by requiring a specific number of points with a certain weight.

Sampling is currently mostly used to numerically approximate integrals or to find the unique polynomial given the values on sample points. In those cases, having a larger number of sample points is not a problem; it requires a higher number of function evaluations but often gives a more precise result. However, a larger number of sample points than required is problematic when solving SDPs because of the linear dependencies it introduces. Further research into the combination of sampling and invariance theory is certainly needed.

5. Sphere packing

In this chapter, we consider the problem of packing spheres of N different sizes into an n-dimensional Euclidean space. Using results from de Laat, Oliviera and Vallentin in [10], we formulate a univariate MPMP to give an upper bound on the sphere packing density; the fraction of space covered.

Furthermore, the time complexity analysis in Section 3.3 is used to give the time complexity for this specific problem. This shows that the use of SDPB or CLRS can give a speedup of order $\mathcal{O}(d)$ compared to SDPA-GMP, where d is the maximum degree used in the program to compute the upper bound.

Finally, we give timing results for the case $N = 1$, showing the expected speedup in complexity.

5.1 Formulation as a univariate MPMP

Let us first state Theorem 5.1 from [10], which is the basis for the formulation. Recall that a function $f : \mathbb{R}^n \to \mathbb{C}$ is a Schwartz function if it is infinitely differentiable, and if any derivative of f multiplied with any power of $x_1, \ldots, x_n$ is a bounded function. A function f is a radial function if $f(x)$ only depends on the norm of x.

Theorem 5.1 ([10, Theorem 5.1])**.** *Let $r_1, \ldots, r_N > 0$, and let $f : \mathbb{R}^n \to \mathbb{R}^{N\times N}$ be a matrix valued function whose every component f_{ij} is a radial Schwartz function. Suppose f satisfies the following conditions:*

(i) The matrix $\left(\hat{f}_{ij}(0) - (\operatorname{vol} B(r_i))^{1/2}(\operatorname{vol} B(r_j))^{1/2}\right)_{i,j=1}^{N}$ is positive semidefinite, where $B(r)$ is the ball of radius r centered at the origin.

(ii) The matrix of Fourier transforms $\left(\hat{f}_{ij}(t)\right)_{i,j=1}^{N}$ is positive semidefinite for every $t > 0$.

(iii) $f_{ij}(w) \leq 0$ if $w \geq r_i + r_j$, for $i, j = 1, \ldots, N$.

Then the density of any sphere packing of spheres of radii $r_1, \ldots, r_N$ in the Euclidean space $\mathbb{R}^n$ is at most $\max\{f_{ii}(0) : i = 1, \ldots, N\}$.

For $N = 1$ and $r_1 = \frac{1}{2}$, this reduces to the well-known Cohn-Elkies linear programming bound [8].

Note that the conditions naturally give MPMP constraints (with $m = N$ or $m = 1$ and $n = 1$), in case polynomials equivalent to $\hat{f}_{ij}$ and f_{ij} can be found. We follow [10] in defining these functions.

Let $d \geq 0$ be an integer, and consider the polynomial matrix

$$\phi(t) = \sum_{k=0}^{d} a_k t^{2k},$$

where $a_k \in \mathcal{S}^N$ for all k. We choose the function f such that the Fourier transform $\hat{f}$ is defined by

$$\hat{f}(u) = \phi(\|u\|)e^{-\pi\|u\|^2},$$

where the (inverse) Fourier transform is applied elementwise. Lemma 5.2 from [10] gives us that f is defined by

$$f(w) = \sum_k a_k \frac{k!}{\pi^k} L_k^{n/2-1}(\pi\|w\|^2)e^{-\pi\|w\|^2},$$

where $L_k^{n/2-1}$ is the degree k Laguerre polynomial with parameter $n/2-1$.

Let us consider the conditions on f. We denote the elements of a_k by $a_{ij,k}$. These elements are the free variables y_i in program (4.1). Note that this gives a total number of $\frac{N(N+1)}{2}d$ variables, because the matrices a_k are symmetric. Condition (i) gives that

$$a_0 - \left((\operatorname{vol} B(r_i))^{1/2}(\operatorname{vol} B(r_j))^{1/2}\right)_{i,j=0}^{N} \succeq 0,$$

which is a normal MPMP constraint with matrices $P_{ij0}^1 = \bar{E}_{ij}$, and $P_{ijk}^1 = 0$ for $k > 1$. In this case, we define $\bar{E}_{ij}$ such that it has a 1 on positions (i,j) and (j,i), for convenient notation. Furthermore, $P_0^1 = -((\operatorname{vol} B(r_i))^{1/2}(\operatorname{vol} B(r_j))^{1/2})_{i,j=0}^N$.

Note that $e^{-\pi t} > 0$ for all $t \in \mathbb{R}$. Hence $\hat{f}(t) \succeq 0$ if and only if $\phi(t) \succeq 0$. As $\hat{f}(0) \succ 0$ by condition (i), this can be extended to $t \geq 0$, which can be given by S_G with $G = \{x\}$. Hence the second condition is

$$0 + \sum_{k=0}^{d}\sum_{i=1}^{N}\sum_{j=i}^{N} a_{ij,k}\bar{E}_{ij}t^{2k} \succeq 0 \qquad \text{on } S_{G_2} = \mathbb{R}_+.$$

The third condition gives N^2 polynomial constraints on the individual elements of f. The domain $||w|| \geq r_i + r_j$ is the semi-algebraic set given by the function $g_{ij}(w) = w^2 - (r_i + r_j)^2$. Hence this can be directly formulated as

$$-f_{ij}(w) = 0 - \sum_{k=0}^{d} a_{ij,k}\frac{k!}{\pi^k}L_k^{n/2-1}(\pi w^2) \geq 0 \qquad \text{on } S_{G_{ij}},$$

or, substituting $x = w^2$,

$$-f_{ij}(x) = 0 - \sum_{k=0}^{d} a_{ij,k}\frac{k!}{\pi^k}L_k^{n/2-1}(\pi x) \geq 0 \qquad \text{on } S_{G_{ij}},$$

where $G_{ij} = \{x - (r_i + r_j)^2\}$. Note that, even though the domains are not compact and hence the corresponding quadratic modules not archimedean, the equivalence between the resulting program and program (4.2) is established by the extensions of Theorem 4.1 in the univariate case, see Appendix B.

Lastly, the value to be optimized is the maximum of $f(0)_{ii}$ for $i = 1, \ldots, N$. This can be modelled in a standard way by adding a variable M and requiring that $M - f(0)_{ii} \geq 0$ for all i. The objective function becomes $-M$, because M should be minimal.

This can be summarized as

$$
\begin{aligned}
\sup \quad & -M \\
\text{s.t.} \quad & -\left((\operatorname{vol} B(r_i))^{1/2}(\operatorname{vol} B(r_j))^{1/2}\right)_{i,j=0}^{N} + \sum_{i\leq j} a_{ij,0}\bar{E}_{ij} \succeq 0\,, \\
& 0 + \sum_{k=0}^{d} \sum_{1\leq i\leq j\leq N} a_{ij,k}\bar{E}_{ij}x^k \succeq 0 && \text{on } S_{G_2} = \mathbb{R}_+\,, \\
& 0 - \sum_{k=0}^{d} a_{ij,k}\frac{k!}{\pi^k}L_k^{n/2-1}(\pi x) \geq 0 && \text{on } S_{G_{ij}} \text{ for } 1 \leq i \leq j \leq N\,. \\
& M - \sum_{k=0}^{d} a_{ii,k}\frac{k!}{\pi^k}L_k^{n/2-1}(0) \geq 0 && \text{for } i = 1, \ldots, N\,.
\end{aligned}
$$

Note that there is one true polynomial matrix constraint; the other constraints are either polynomial constraints or normal semidefinite programming constraints.

5.2 Time complexity

We use a maximum degree of $2d$ as is in general done; this makes it possible to use bases of degree at most d. Recall that the variables are given by $a_{ij,k}$, with $1 \leq i \leq j \leq N$ and $k = 0, \ldots, 2d$. We denote the number of required sample points for a constraint j by K_j, and the maximum length of the basis by U_j. Although the length of the basis depends on both l and j, this will be a difference of at most 1. In the notation of Section 3.3, we have $P_j = \frac{m_j(m_j+1)}{2}K_j = \mathcal{O}(m_j^2 K_j)$ and $U_{j,l} \approx U_j$. The conditions of Theorem 5.1 give:

(i) 1 constraint with $m_1 = N$, $n_1 = 0$, $L_1 = 1$. This gives $K_1 = U_1 = 1$.

(ii) 1 constraint with $m_2 = N$, $n_2 = 1$, $\delta_2 = 2d$ and $L_2 = 2$. This gives $K_2 = 2d + 1$, and $U_2 = d + 1$.

(iii) $\frac{N(N+1)}{2}$ constraints with $m_j = 1$, $n_j = 1$, $\delta_j = 2d$, and $L_j = 2$, which gives $K_j = 2d + 1$ and $U_j = d + 1$.

(iv) The minimization of the maximum of $a_{ii,0}$ gives N extra constraints, with $m_j = 1$, $n_j = 0$ (and thus $K_j = U_j = 1$), and $L_j = 1$.

This gives a complexity (in terms of d and N) of $\mathcal{O}(N^6 d^3)$ for the Cholesky decomposition of Q, $\mathcal{O}(N^6 d^2 + N^6 d + 2N^2 + N^6)$ for condition (i), $\mathcal{O}(2N^6 d^3 + 4N^6 d^3 + 12N^2 d^3 + 8N^6 d^3)$ for condition (ii) and $\mathcal{O}(N^2(2N^4 d^3 + 4N^2 d^3 + 12d^3 + 8d^3))$ for condition (iii). The extra constraints for the objective function add $\mathcal{O}(N(N^4 d^2 + N^2 d + 1))$, which is negligible compared to the other terms. This gives a leading term of $\mathcal{O}(N^6 d^3)$. In contrast, the

MPMP implementation in SDPA-GMP gives $\mathcal{O}(N^6d^3 + N^5d^4)$ due to the term $m_j^5U_j^3K_j$ in the general complexity (see Table 2 and Section 4.5; the m_j^5 comes from the term $\otimes E_{r,s}$). This is especially important for high degrees d for a constant N, which is the approach for obtaining bounds.

In [10, Problem B], the problem was modelled slightly differently. The polynomial matrix $\hat{f}$ was rewritten to a polynomial σ, which is a sum of squares if and only if $\hat{f} \succeq 0$ (similar to Theorem B.1). This polynomial was directly encoded as a sum of squares, hence condition (ii) was automatically satisfied. However, condition (iii) still gives N^2 polynomial constraints of degree $\mathcal{O}(d)$, with the same complexity. Another difference is that the constraints were rewritten to SDP constraints using coefficient matching instead of sampling. Because the Laguerre basis was used for stability, this does not introduce much sparsity for coefficient matching of low degrees, although it does for coefficient matching of high degrees. In general, it is unclear to what extend this increases the speed of the calculations.

5.3 Timing results

In this section, we present results regarding the speed comparison between SDPA-GMP, SDPB and our Julia solver CLRS. The problem considered is the Cohn-Elkies bound, which is equivalent to the N-sphere packing bound for $N = 1$ and $r = \frac{1}{2}$. All solvers use 1024 bits precision. The problem is purely used as test case; there are other techniques to solve this problem with good results [8]. In the case of SDPA-GMP, coefficient matching was used with the Laguerre basis, both as basis for the sum of squares polynomials and as basis for coefficient matching. For SDPB and CLRS, we also use the Laguerre basis for the sum of squares polynomials. The sample points used are the default sample points of SDPB. Furthermore, we set the stopping conditions to for all solvers to having a duality gap of at most 10^{-15} with primal and dual error thresholds of 10^{-15}. The remaining parameters were taken to be the default for the solver, where we took the parameters of SDPB as default for CLRS. All computations ran on the same computer.

5.3.1 Single thread

As can be seen in Figure 1, the complexity of SDPA-GMP is indeed higher than the complexity of the specialized solvers. In addition, CLRS and SDPB require less time to solve the problem than SDPA-GMP starting around $d = 50$.

Furthermore, CLRS is slightly faster than SDPB for this problem on one thread. As SDPA-GMP has no multi-threaded implementation, we used a single thread to generate this plot. An important remark is that SDPA-GMP requires less precision for this problem. Both SDPB and CLRS fail for $d = 80$ with a precision of 1024 bits, whereas SDPA-GMP does not (although the answer is incorrect; it is therefore not included in the figure). Furthermore, the problem can be solved with less bits with SDPA-GMP than SDPB and CLRS for most d. Using a different precision would show itself in the plot as a constant factor.

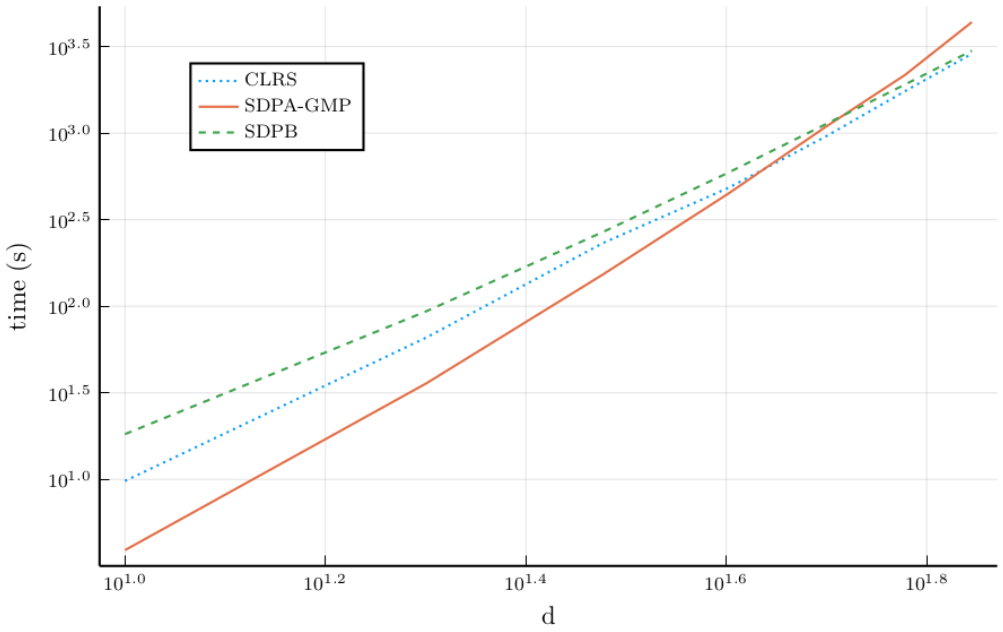

Figure 1: The time needed to compute the Cohn-Elkies bound for constant start parameters and varying degree on a logarithmic scale. For SDPA-GMP, coefficient matching was used. For all algorithms, a single thread was used.

5.3.2 Multiple threads

There are several ways in which multiple threads would be usable in CLRS. First of all, nearly everything can be done for each cluster (i.e. each polynomial constraint) separately. Extra threads can then be used to speed up operations such as matrix multiplications in certain clusters. SDPB uses this since it is easier to scale to several clusters of processors [20]. For the computation of the bilinear pairings and S, it is also possible to distribute the computations which have to be done for a single sample point over the threads. This has the advantage that the speedup obtained from the extra threads directly scales with the number of threads for this part of the iteration. However, as the computation of the bilinear pairings is often not the bottleneck, the approach taken by SDPB is superior due to less communication overhead.

In Figure 2, we show the time needed to solve the program with SDPB and CLRS, using one and two cores. The time needed for SDPA-GMP with one core is also included, as extra reference. There are several interesting aspects of this figure. First, both CLRS and SDPB are clearly faster than SDPA-GPM when using multiple cores. Although this does seem an unfair comparison, it shows an advantage of the parallel solvers; SDPA-GMP is not parallellized, even though it is possible for computing S.[1] Second, SDPB has a clear advantage on CLRS when using multiple cores. The reason for this is not com-

[1] In several other solvers of the SDPA family, the computation of S is parallellized [33].

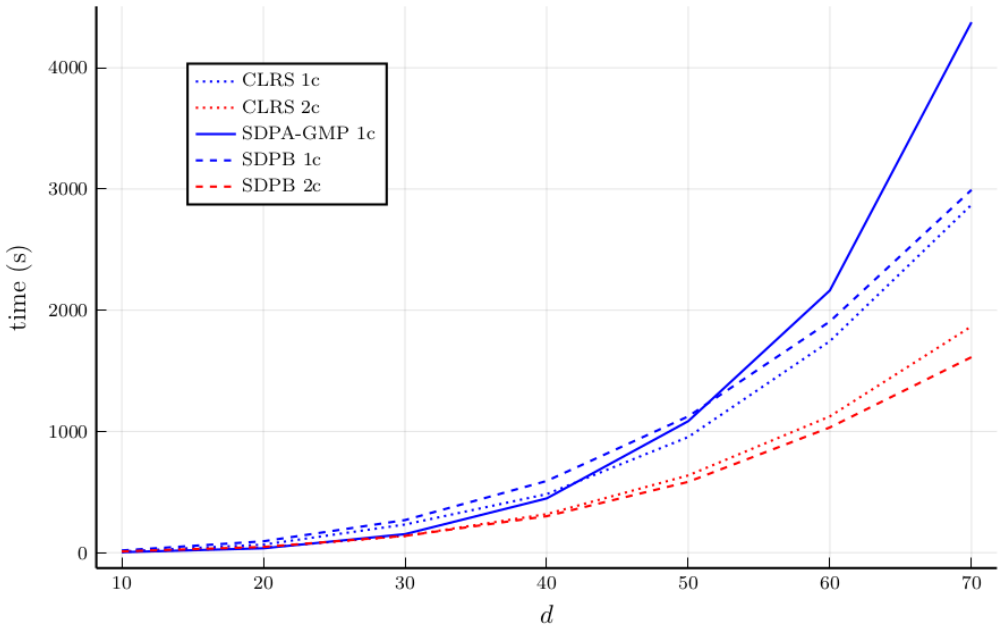

Figure 2: The time needed to compute the Cohn-Elkies bound for constant starting parameters and varying degree, with one and two cores. SDPA-GMP has no possibility to run with multiple cores. Different line styles represent different solvers (solid for SDPA-GMP, dashed for SDPB and dotted for CLRS). Different colors represent different amounts of cores used (blue for one core and red for two cores).

pletely clear when using two cores. A possible reason is extra communication overhead. In CLRS, the multithreading is done within functions, whereas SDPB runs computations completely on a core; only the calculations which connect clusters require communication between cores. Furthermore, SDPB optimizes the distribution of clusters over threads using timings of the first two iterations. In CLRS this is not done (at the time of writing this thesis); the threads are used according to the native (static) scheduler, which evenly divides the clusters over the threads. In this case, the distribution of the constraints over the threads should be optimal or near to optimal. In the case of more cores than constraints, SDPB assigns the extra cores to constraints in order to speed up operations such as matrix multiplication. It is not clear to what extend this would be possible with the linear algebra library which is currently used for CLRS.

It should be noted that this problem was chosen because of the simplicity and because it is possible to solve with all solvers. In the next chapter, among others a three-point bound is given using polynomial constraints in three variables. Because this requires more polynomial weights than two, it is not possible to use this with SDPB. Furthermore, we used symmetry reduction for extra speed, which introduces higher rank structures. This is the main other difference between the capabilities of CLRS and SDPB; the former can solve problems with higher rank structures whereas the latter cannot.

6. Spherical cap packing

In this chapter, we consider the problem of packing spherical caps of N different sizes on the unit sphere. This is related to spherical codes with unequal error protection. After the general introduction, we consider two-point bounds, both for $N = 1$ and $N > 1$, using the well-known Delsarte LP-bound and a theorem from de Laat et al. in [10]. In Section 6.3 the three-point bound for $N = 1$ of Bachoc and Vallentin [1, 2, 13] is considered, and extended to $N > 1$. This extension can also be generalized to the general k-point bound, as shown in Section 6.5.

6.1 Preliminaries

A spherical cap around a point e on the n-dimensional unit sphere S^{n-1}, the center, with angle θ is defined by

$$\mathrm{Cap}^{n-1}(e, \theta) = \{x \in S^{n-1} \mid e^T x \geq \cos(\theta)\}\,.$$

Let $w(\theta)$ be the fraction of the unit sphere $\mathrm{Cap}^{n-1}(e, \theta)$ covers:

$$w(\theta) = \frac{\omega_{n-1}(S^{n-2})}{\omega_n(S^{n-1})} \int_{\cos\theta}^{1} (1 - u^2)^{(n-3)/2} du\,, \tag{6.1}$$

where $\omega_n(S^{n-1}) = (2\pi^{n/2})/\Gamma(n/2)$ is the surface area of the unit sphere, and $\Gamma(x)$ is the gamma function.

Given a graph $G = (V, E)$, a set $I \subseteq V$ is independent if no two vertices in I are adjacent. Given a weight function $w : V \to \mathbb{R}_+$, the weighted independence number $\alpha_w(G)$ is the maximum weight of an independent set in G:

$$\alpha_w(G) = \max\{\sum_{x \in I} w(x) \mid I \subseteq V \text{ is independent}\}\,. \tag{6.2}$$

The problem of finding the maximum density of a packing of spherical caps is equivalent to finding the weighted independence number $\alpha_w(G)$ of the infinite graph $G = (V, E)$ with $V = S^{n-1} \times \{1, \ldots, N\}$, where two vertices $(x, i), (y, j) \in V$ are adjacent if the caps $\mathrm{Cap}^{n-1}(x, \theta_i)$ and $\mathrm{Cap}^{n-1}(y, \theta_j)$ overlap, i.e. when

$$x \cdot y > \cos(\theta_i + \theta_j)\,.$$

The density of a packing of spherical caps is given by the sum of the contributions of the caps in the packing to the density. Thus these contributions, the fraction of the sphere covered by a cap, are the weights. For the case $N = 1$, it is equivalent to take weights of 1, and consider the cardinality of the packing instead of the density.

6.2 Two-point bounds

A well-known upper bound for the (weighted) independence number of a graph $G = (V, E)$ is the (weighted) Lovász theta number:

$$\begin{aligned} \theta'_w(G) = \inf \quad & M & & \\ \text{s.t.} \quad & K - w^{1/2} \otimes (w^{1/2})^* & & \text{is a positive definite kernel,} \\ & K(x,x) \le M & & \text{for } x \in V\,, \\ & K(x,y) \le 0 & & \text{for } (x,y) \notin E \text{ with } x \neq y\,, \\ & M \in \mathbb{R}, K \in \mathcal{C}(V \times V) & & \text{is symmetric}\,. \end{aligned} \tag{6.3}$$

where $\mathcal{C}(V \times V)$ is the set of all continuous kernels on V. A kernel K on V is symmetric if $K(x,y) = K(y,x)$ for all $x, y \in V$, and positive definite if for all finite sets $C = \{x_1, \ldots, x_m\}$, the matrix $(K(x_i, x_j))_{i,j=1}^m$ is positive semidefinite. For the spherical cap packing problem, we can restrict the positive definite kernels to be $O(n)$ invariant: Any rotation of an (optimal) solution, gives another (optimal) solution. Hence the group average of an optimal solution is an optimal solution to $\theta'_w(G)$ restricted to invariant kernels.

Schoenberg [28] gives the following characterisation of $O(n)$-invariant positive definite kernels on S^{n-1}.

Proposition 6.1. *Let P_k^n denote the Gegenbauer polynomials with parameter $n/2 - 1$, normalized such that $P_k^n(1) = 1$ for all $n, k \in \mathbb{N}$. Each positive definite $O(n)$ invariant kernel on S^{n-1} can be written as*

$$K(x,y) = \sum_{k=0}^{\infty} a_k P_k^n(x \cdot y)\,, \tag{6.4}$$

where a_k are nonnegative, and the convergence is uniform and absolute.

Together with (6.3), this leads for $N = 1$ to the well-known linear programming bound of Delsarte et al. [12], taking $u = x \cdot y$:

$$\begin{aligned} \inf \quad & M & & \\ \text{s.t.} \quad & \sum_{k=0}^{\infty} a_k \le M - 1\,, & & \\ & \sum_{k=0}^{\infty} a_k P_k^n(u) \le -1 & & \text{for } 1 \le u \le \cos(\theta)\,, \\ & a_k \ge 0 & & \text{for all } k \ge 0\,. \end{aligned}$$

In [10], this is generalized to $N > 1$ using the following theorem:

Theorem 6.2 ([10, Theorem 2.1])**.** *A symmetric kernel $K \in \mathcal{C}(V \times V)$ with $V = S^{n-1} \times \{1, \ldots, N\}$ is positive and $O(n)$-invariant if and only if*

$$K((x,i),(y,j)) = f(x \cdot y)_{ij}\,, \tag{6.5}$$

with

$$f(u) = \sum_{k=0}^{\infty} A_k P_k^n(u)\,, \tag{6.6}$$

where $A_k \in \mathcal{S}^N$ *is positive semidefinite for all* $k \geq 0$ *and* $\sum_{k=0}^{\infty} |f_{k,ij}| < \infty$ *for all* $i, j = 1, \ldots, N$*, implying in particular that we have uniform convergence.*

The theorem can be proven using Bochner's characterisation for invariant kernels. This leads to the following upper bound [10], taking $(K - w^{1/2} \otimes (w^{1/2})^*)((x,i),(y,j)) = f(x \cdot y)_{ij}$ in (6.3):

$$\begin{array}{lll} \inf & M \in \mathbb{R} & \\ \text{s.t.} & f(1)_{ii} \leq M - w(\theta_i)\,, & \\ & f(u)_{ij} \leq -w(\theta_i)^{1/2} w(\theta_j)^{1/2} & \text{for } 1 \leq u \leq \cos(\theta_i + \theta_j)\,, \\ & A_k \succeq 0 & \text{for all } k \geq 0\,. \end{array} \tag{6.7}$$

with f as defined in Theorem 6.2.

In practice, the sum is restricted to a degree d to obtain a solvable program. A higher degree gives a better bound at the cost of a larger semidefinite program.

6.3 Three-point bounds

In [11], a general formulation for k-point bounds for the independence number $\alpha(G)$ is given. For $k = 3$ this gives

$$\begin{array}{lll} \inf & M \in \mathbb{R} & \\ \text{s.t.} & T \in \mathcal{C}(V \times V \times I_1)_{\succeq 0}\,, & \\ & B_3 T(S) \leq M - 1 & \text{for } S \in I_{=1}\,, \\ & B_3 T(S) \leq -2 & \text{for } S \in I_{=2}\,, \\ & B_3 T(S) \leq 0 & \text{for } S \in I_{=3}\,, \end{array} \tag{6.8}$$

where $I_{=k}$ is the set of independent sets of cardinality k and $I_k = \cup_{0 \leq i \leq k} I_{=k}$. The operator $B_3 : \mathcal{C}(V \times V \times I_1)_{\text{sym}} \to \mathcal{C}(I_3 \setminus \{\emptyset\})$ is given by

$$B_3 T(S) = \sum_{\substack{Q \subseteq S \\ |Q| \leq 1}} \sum_{\substack{x,y \in S \\ Q \cup \{x,y\} = S}} T(x,y,Q)\,.$$

Furthermore, $\mathcal{C}(V \times V \times I_1)_{\succeq 0}$ is the set of continuous functions on $V \times V \times I_1$ which are positive definite kernels on $V \times V$ for all elements in I_1. Bachoc and Vallentin characterise such kernels. To state the theorem, we define, for $i, j \geq 0$, the matrix elements

$$Y_k^n(u,v,t)_{ij} = u^i v^j (1-u^2)^{k/2} (1-v^2)^{k/2} P_k^{n-1}\left(\frac{t - uv}{\sqrt{(1-u^2)(1-v^2)}}\right)\,.$$

A matrix $U \in O(n)$ stabilizes $z \in S^{n-1}$ if $Uz = z$; the set of all such matrices is called the stabilizer $\text{Stab}_{O(n)}(z)$ of z.

Theorem 6.3. *Let $z \in S^{n-1}$ and $F_k \in \mathcal{S}_{\succeq 0}^{d-k+1}$ for $k = 0, \ldots, d$. Then*

$$(x, y) \mapsto \sum_{k=0}^{d} \mathrm{Tr}(F_k Y_k^n (x \cdot z, y \cdot z, x \cdot y)^T)$$

is a $\mathrm{Stab}_{O(n)}(z)$−invariant positive definite kernel on S^{n-1}, and every $\mathrm{Stab}_{O(n)}(z)$−invariant positive definite kernel is the uniform limit of kernels of this form.

Note that the transpose in the theorem is in fact not needed, because F_k is symmetric. However, this increases the similarity between Theorem 6.3 and Theorem 6.4, the generalization for N sizes of spherical caps. Furthermore, one can change the factor $u^i v^j$ in the definition of $Y_k^n(u, v, t)_{ij}$ to $q_i(u)q_j(v)$ for any other basis of polynomials q. This can be seen as changing basis for both F_k and Y_k^n. Originally, the basis consisted of the polynomials P_i^{n+2k} [1].

Using Theorem 6.3 to define $T(x, y, \{z\})$, and $T(x, y, \emptyset) = \sum_k a_k P_k^n(x \cdot y)$ gives the three-point bound [13, program 8]:

$$\begin{aligned}
\inf \quad & M \in \mathbb{R} && && (6.9)\\
\text{s.t.} \quad & \sum_{k=0}^{d} a_k + F(1,1,1) \leq M - 1\,, && \\
& \sum_{k=0}^{d} a_k P_k^n(u) + 3F(u,u,1) \leq -1 && \text{for } -1 \leq u \leq \cos(\theta)\,, \\
& F(u, v, t) \leq 0 && \text{for } (u, v, t) \in \Delta\,, \\
& F_k \in S_{\succeq 0}^{d-k+1}, a_k \geq 0 && \text{for } k = 0, \ldots, d\,,
\end{aligned}$$

where

$$F(u, v, t) = \sum_{k=0}^{d} \langle F_k, S_k^n(u, v, t) \rangle$$

and

$$\Delta = \{(u, v, t) \in \mathbb{R}^3 \mid -1 \leq u, v, t \leq \cos(\theta), 1 + 2uvt - u^2 - v^2 - t^2 \geq 0\}\,.$$

Furthermore, $S_k^n = \sum_{\sigma \in S_3} \sigma Y_k^n(u, v, t)$, where σ acts on Y_k^n by permuting its arguments. This symmetrization is possible because the points x, y and z behave similarly. The set Δ is obtained from considering all valid tuples of inner products of points on S^{n-1}; the first constraints in Δ give that the caps do not overlap, and the last constraint together with the bounds on u, v, t requires the centers to be on the sphere. Because F and Δ are symmetric in u, v and t, symmetry reduction can be used to solve the problem more efficiently; see Section 6.4 for the relevant details.

In the following, we will focus on the extension of this problem to N sizes of spherical caps. For this, a characterization similar to Theorem 6.3 is needed. As before, we denote the partial trace as Tr_m, where m is the dimension the trace is taken over.

Theorem 6.4. *Let $z \in S^{n-1}$ and $F_k \in \mathcal{S}_{\succeq 0}^{(d-k+1)N}$ for $k = 0, \dots, d$. Then*

$$((x,i),(y,j)) \mapsto \sum_{k=0}^{d} \mathrm{Tr}_{d-k+1}(F_k(Y_k^n(x\cdot z, y\cdot z, x\cdot y)^T \otimes I_N))_{ij}$$

is a $\mathrm{Stab}_{O(n)}(z)-$invariant positive definite kernel on $V = S^{n-1}\times\{1,\dots,N\}$, and every $\mathrm{Stab}_{O(n)}(z)-$invariant positive definite kernel on V is the uniform limit of this kernels of this form.

Here, $O(n)$ acts on V by $\gamma(x,i) = (\gamma^{-1}x, i)$ for $\gamma \in O(n)$, i.e. the action is restricted to the natural action on the sphere. Theorem 6.4 can be seen as an element-wise application of Theorem 6.3, similar to Theorem 6.2 being an element-wise application of Proposition 6.1.

Proof. We will prove it using Bochner's Theorem. Let $d \in \mathbb{N}$. As in [1] and [2], $\mathrm{Pol}_{\leq d}(S^{n-1})$ can be orthogonally decomposed as

$$\mathrm{Pol}_{\leq d}(S^{n-1}) = \bigoplus_{k=0}^{d}\bigoplus_{i=k}^{d} H_{k,i}^{n-1}\,, \tag{6.10}$$

where $H_{k,i}^{n-1}$ is isomorphic to Harm_k^{n-1}, the $O(n-1)$-irreducible space of homogeneous, harmonic polynomials of degree k in $n-1$ variables. To make this more concrete, we have

$$H_{k,i}^{n-1} = \mathrm{span}\{e_{k,i,0},\dots,e_{k,i,d-k}\}\,,$$

where for example

$$e_{k,i,j}(x) = (x\cdot z)^j \mathcal{Y}_k^{n-1}\Big(\frac{x-(x\cdot z)z}{\sqrt{1-x\cdot z}}\Big)\,.$$

In this expression, $\mathcal{Y}_k^{n-1}$ are the spherical harmonics in $n-1$ variables.

Using the decomposition (6.10), the space $\mathrm{Pol}_{\leq d}(S^{n-1})\otimes\mathbb{R}^{\{1,\dots,N\}}$ can be decomposed as

$$\mathrm{Pol}_{\leq d}(S^{n-1})\otimes\mathbb{R}^{\{1,\dots,N\}} = \bigoplus_{k=0}^{d}\bigoplus_{i=k}^{d}(H_{k,i}^{n-1}\otimes\mathbb{R}^N) = \bigoplus_{k=0}^{d}\bigoplus_{i=0}^{d-k}\bigoplus_{a=1}^{N} H_{k,(i,a)}^{n-1}\,.$$

A basis of $H_{k,(i,a)}^{n-1}$ can be given by $(x,b)\mapsto e_{k,i,j}(x)\delta_a(b)$ for $j = 0,\dots,d-k$, where $\delta_a(b)$ is the kronecker delta.

Bochner's characterization [5] gives that any positive definite kernel on V can be written as

$$\begin{aligned}
K((x,a),(y,a')) &= \sum_{k=0}^{d}\sum_{i,i'=1}^{m_k}\sum_{b,b'=1}^{N} F^k_{(i,b),(i',b')}\sum_{j=0}^{d-k} e_{k,i,j}(x)e_{k,i',j}(y)\delta_b(a)\delta_{b'}(a') \\
&= \sum_{k=0}^{d}\sum_{i,i'=1}^{m_k} F^k_{(i,a),(i',a')} Y_k^n(x\cdot e, y\cdot e, x\cdot y)_{i,i'} \\
&= \sum_{k=0}^{d} \mathrm{Tr}_{d-k+1}(F^k(Y_k^n(x\cdot e, y\cdot e, x\cdot y)^T\otimes I_N))_{aa'}\,,
\end{aligned}$$

where the matrices F^k are positive semidefinite. The sum over the basis is rewritten using Theorem 3.1 and 3.2 of [1].

Uniform convergence follows from [9, Theorem A.8]. In fact, in this case it is equivalent to stating that the set of polynomials is uniformly dense in the space of continuous functions on the unit sphere.

□

We would like to use program (6.8) in a similar way for an upper bound on the spherical cap packing density with N sizes of caps. However, the program gives a bound for the independence number $\alpha(G)$, the maximum number of points which form an independent set in the graph G. Hence it cannot be used directly for finding the maximum density of spherical caps on the unit sphere, which requires weights. Using the similarity between the weighted and unweighted $\theta'(G)$ number, we find the weighted three-point bound given in (6.11). Similarly, we can generalize the k-point bound of [11] to a weighted k-point bound, see Section 6.5; program (6.11) is the special case with $k = 3$.

$$
\begin{aligned}
\inf \quad & M \in \mathbb{R}, && F_k^l, A^k \succeq 0 \qquad (6.11)\\
\text{s.t.} \quad & \sum_{k=0}^{d} A_{ii}^k + F(1,1,1)_{iii} \leq M - w(\theta_i) && \text{for } i = 1, \ldots, N\,,\\
& \sum_{\sigma \in S_3} \sigma(F(u,v,t)_{ijl}) \leq 0 && \text{for } (u,v,t) \in \Delta_{ijl} \text{ and } i,j,l = 1, \ldots N\,,\\
& 2w(\theta_i)^{1/2} w(\theta_j)^{1/2} \sum_{k=0}^{d} A_{ij}^k P_k^n(u) && \\
& + w(\theta_i)^{2/3} w(\theta_j)^{1/3} (2F(1,u,u)_{iji} + F(u,u,1)_{iij}) && \text{for } u \in [-1, \cos(\theta_i + \theta_j)]\\
& + w(\theta_i)^{1/3} w(\theta_j)^{2/3} (2F(1,u,u)_{jij} + F(u,u,1)_{jji}) && \text{and } i,j = 1, \ldots, N\,,\\
& \leq -2w(\theta_i) w(\theta_j) &&
\end{aligned}
$$

where

$$F(u,v,t)_{ijl} = \sum_{k=0}^{d} \mathrm{Tr}_{d-k+1}(F_k^l (Y_k^n(u,v,t)^T \otimes I_N))_{ij}$$

and

$$
\begin{aligned}
\Delta_{ijl} = \{(u,v,t) \mid & -1 \leq u \leq \cos(\theta_i + \theta_l)\,,\\
& -1 \leq v \leq \cos(\theta_j + \theta_l)\,,\\
& -1 \leq t \leq \cos(\theta_i + \theta_j)\,,\\
& 1 + 2uvt - u^2 - v^2 - t^2 \geq 0\}\,.
\end{aligned}
$$

Furthermore, $\sigma \in S_3$ acts on $F(u,v,t)_{ijl}$ by permuting i,j,l; the permutation of u,v,t follows from $u = x_i \cdot x_l$, $v = x_j \cdot x_l$ and $t = x_i \cdot x_j$. This leads to the same constraints for (i,j,l) and $\sigma(i,j,l)$, hence we only need to consider $i \leq j \leq l$ while solving the program. Note that, for $N = 1$, this is exactly the three-point bound for the maximum cardinality of

a spherical code, scaled by a factor $w(\theta_1)$. In addition, setting $F(u,v,t)_{ijl} = 0$ (equivalent to not considering the part of the constraints on three points) gives the two-point bound (6.7).

Proposition 6.5. *Program* (6.11) *gives an upper bound to the spherical cap packing density with cap sizes* $\theta_1, \dots, \theta_N$.

Proof. Let $x_i \in S^{n-1}$ for $i = 1, \dots, m$ and $r : \{1, \dots, m\} \to \{1, \dots, N\}$ be such that

$$C = \bigcup_{i=1}^{m} \mathrm{Cap}^{n-1}(x_i, \theta_{r(i)})$$

is a packing of spherical caps. Consider the sum

$$\begin{aligned} S = &\sum_{i,j,l=1}^{m} F(x_i \cdot x_l, x_j \cdot x_l, x_i \cdot x_j)_{r(i)r(j)r(l)} w(\theta_{r(i)})^{1/3} w(\theta_{r(j)})^{1/3} w(\theta_{r(l)})^{1/3} \\ &+ \sum_{i,j=1}^{m} \sum_{k=0}^{d} A^k_{r(i)r(j)} P^n_k(x_i \cdot x_j) w(\theta_{r(i)})^{1/2} w(\theta_{r(j)})^{1/2} . \end{aligned}$$

Note that, by Theorems 6.4 and 6.2,

$$\begin{aligned} &\sum_{i,j=1}^{m} F(x_i \cdot x_l, x_j \cdot x_l, x_i \cdot x_j)_{r(i)r(j)r(l)} w(\theta_{r(i)}) w(\theta_{r(j)}) \geq 0 , \qquad \text{for } l = 1, \dots, N \\ &\sum_{i,j=1}^{m} w(\theta_{r(i)}) w(\theta_{r(j)}) \sum_{k=0}^{d} A^k_{r(i)r(j)} P^n_k(x_i \cdot x_j) \geq 0 . \end{aligned}$$

As $w(\theta_{r(l)}) \geq 0$, this means that $S \geq 0$. The sum S also equals

$$\begin{aligned} &\sum_i w(\theta_{r(i)}) \Big(F(1,1,1)_{r(i)r(i)r(i)} + \sum_{k=0}^{d} A^k_{r(i)r(i)} \Big) \\ &+ \sum_{i \neq j} \Big(w(\theta_{r(i)})^{1/2} w(\theta_{r(j)})^{1/2} \sum_{k=0}^{d} A^k_{r(i)r(j)} P(x_i \cdot x_j) \\ &+ w(\theta_{r(i)})^{2/3} w(\theta_{r(j)})^{1/3} \big(F(1, x_j \cdot x_i, x_i \cdot x_j)_{r(i)r(j)r(i)} \\ &+ F(x_j \cdot x_i, 1, x_j \cdot x_i)_{r(j)r(i)r(i)} + F(x_i \cdot x_j, x_i \cdot x_j, 1)_{r(i)r(i)r(j)} \big) \Big) \\ &+ \sum_{i \neq j \neq l} w(\theta_{r(i)})^{1/3} w(\theta_{r(j)})^{1/3} w(\theta_{r(l)})^{1/3} F(x_i \cdot x_l, x_j \cdot x_l, x_i \cdot x_j)_{r(i)r(j)r(l)} \\ &\leq \sum_i w(\theta_{r(i)})(M - w(\theta_{r(i)})) - \sum_{i \neq j} w(\theta_{r(i)}) w(\theta_{r(j)}) \\ &= \sum_i w(\theta_i) \Big(M - \sum_j w(\theta_j) \Big) , \end{aligned}$$

where the inequality is valid because of the constraints on F and because C is a non-overlapping packing of spherical caps. Hence $M \geq \sum_{i=1}^{m} w(\theta_{r(i)})$. □

6.4 Symmetry reduction

In [13], similar as in [23], program (6.9) was solved using symmetry reduction for the constraint in three variables. In this section, we will recap the method used, and apply it to the three-point bound for N sizes of spherical caps.

Let F be a 3-variate polynomial of degree $2d$, invariant under the action of S_3. We require F to be positive on

$$\Delta = \{(u,v,t) \in \mathbb{R}^3 \mid -1 \leq u,v,t \leq \cos(\theta), 1 + 2uvt - u^2 - v^2 - t^2 \geq 0\}\,.$$

Using Putinar's Theorem, F is positive on Δ implies that there are sums of squares polynomials $q_0, \ldots, q_4$ such that

$$\begin{aligned} F(u,v,t) = {} & q_0(u,v,t) + p(u)q_1(u,v,t) + p(v)q_2(u,v,t) \\ & + p(t)q_3(u,v,t) + (1 + 2uvt - u^2 - v^2 - t^2)q_4(u,v,t)\,, \end{aligned}$$

where $p(x) = (\cos(\theta) - x)(x+1)$. Reformulating Δ, we have

$$\Delta = \{(u,v,t) \in \mathbb{R}^3 \mid s_i \geq 0 \text{ for } i = 1, \ldots, 4\}\,,$$

where

$$\begin{aligned} s_1 &= p(u) + p(v) + p(t)\,, \\ s_2 &= p(u)p(v) + p(v)p(t) + p(u)p(t)\,, \\ s_3 &= p(u)p(v)p(t)\,, \\ s_4 &= 1 + 2uvt - u^2 - v^2 - t^2\,. \end{aligned}$$

For a proof of the equivalence see [23, Lemma 3.1]. This gives

$$F = q_0 + \sum_{i=1}^{4} s_i q_i\,.$$

As F and s_i are symmetric in u,v,t, we can assume without loss of generality that the q_i are also symmetric in u,v,t. As in [13], we define symmetrized variables

$$\phi_1 = u + v + t\,, \qquad \phi_2 = uv + vt + ut\,, \qquad \phi_3 = uvt\,.$$

Let $b_d(\phi_1, \phi_2, \phi_3)$ be a vector of basis polynomials in ϕ_1, ϕ_2 and ϕ_3 for polynomials up to total degree d in u,v,t. Then one can show as used in [13], by [14], that for every S_3-invariant sums of squares polynomial q of degree $2d$ there are positive semidefinite matrices Q_1, Q_2, Q_3 such that

$$q(u,v,t) = \operatorname{Tr}(Q_1(b_d b_d^T \otimes \Pi_1)) + \operatorname{Tr}(Q_2(b_{d-2} b_{d-2}^T \otimes \Pi_2)) + \operatorname{Tr}(Q_3(b_{d-1} b_{d-1}^T \otimes \Pi_3))\,,$$

where

$$\begin{aligned} &\Pi_1 = 1\,, \qquad \Pi_2 = \phi_1^2\phi_2^2 - 4\phi_2^3 - 4\phi_1^3\phi_3 + 18\phi_1\phi_2\phi_3 - 27\phi_3^2\,, \\ &\Pi_3 = \begin{pmatrix} 2\phi_1^2 - 6\phi_2 & -\phi_1\phi_2 + 9\phi_3 \\ -\phi_1\phi_2 + 9\phi_3 & 2\phi_2^2 - 6\phi_1\phi_3 \end{pmatrix}\,. \end{aligned}$$

Note that Π_3 is of rank 2 for general u, v, t. In SDPB, this would be another reason because of which the problem cannot be solved (next to the number of polynomial weights). However, this is not a problem because CLRS accepts low rank structures instead of rank 1 structure. At the end of the next section, the difference between using symmetry with the rank 2 structure and not using symmetry will be considered in the context of solving time.

Because every polynomial used is S_3-invariant, a sample point (u, v, t) actually represents constraints on $\sigma(u, v, t)$ for every $\sigma \in S_3$. Hence less sample points are needed for the unisolvent set. Seen differently, this is because the subspace of invariant polynomials has a lower dimension. See Section 4.6.5 for a discussion on how to choose sample points in a symmetrized setting, and Section 6.7 for results and a discussion regarding the quality of the solution.

For program (6.11), something similar can be done in case $i = j$, $i = l$, $j = l$ or $i = j = l$. Let us first consider the case $i = j = l$. Writing out all permutations gives the constraint

$$F(u,v,t)_{iii} + F(u,t,v)_{iii} + F(v,u,t)_{iii} + F(v,t,u)_{iii} + F(t,u,v)_{iii} + F(t,v,u)_{iii} \leq 0\,,$$

for $(u, v, t) \in \Delta_{iii} = \{(u, v, t) \mid u, v, t \in [-1, \cos(2\theta_i)], 1 + 2uvt - u^2 - v^2 - t^2 \geq 0\}$. Notice that everything is completely symmetric in u, v, t, hence we can use the same strategy as before.

Now suppose $i = j \neq l$. This gives

$$F(u,v,t)_{iil} + F(v,u,t)_{iil} + F(t,u,v)_{ili} + F(t,v,u)_{ili} + F(u,t,v)_{lii} + F(v,t,u)_{lii} \leq 0\,,$$

for $(u, v, t) \in \Delta_{iil} = \{(u, v, t) \mid u, v \in [-1, \cos(\theta_i + \theta_l)],\ t \in [-1, \cos(\theta_i + \theta_j)],\ 1 + 2uvt - u^2 - v^2 - t^2 \geq 0\}$. This is symmetric in u, v, and hence we can use the following reduction (as for [13, Program 2]). Let $b_d = b_d(u, v, t)$ be a vector of basis polynomials for the polynomials up to degree d, symmetric in u, v. Then, because of the decomposition

$$\mathbb{R}[u, v, t] = \mathbb{R}[u + v, uv, t] \oplus (u - v)\mathbb{R}[u + v, uv, t]\,,$$

any sums of squares polynomial $q(u, v, t)$ which is symmetric in u, v can be written as

$$q(u, v, t) = \mathrm{Tr}(Y_1 b_d b_d^T) + \mathrm{Tr}(Y_2 (b_{d-1} b_{d-1}^T) \otimes (u - v)^2)\,.$$

The other cases follow similarly.

6.5 k-point bounds

In analog with the generalization from spherical cap packing with one cap to spherical cap packing with N caps, for both the two and three-point bound, one can expect something similar to happen for k-point bounds. As mentioned before, de Laat et al. derive in [11] a hierarchy of k-point bounds for the independence number of a graph $G = (V, E)$. The corresponding three-point bound was shown in Section 6.3, and the two-point bound reduces to the θ' number. In this section, this is generalized to a k-point bound with N different weights, similar to the generalizations of the two and three-point bounds.

Let us first introduce the setting. Let $k \geq 2$, and let $\mathcal{C}(V^2 \times I_{k-2})_{\text{sym}}$ be the set of continuous real-valued functions on $V^2 \times I_{k-2}$ which are symmetric in the first two coordinates. As before, $I_{=k}$ is the set of independent sets of cardinality k, and $I_k = \cup_{i \leq k} I_{=k}$. An operator $T \in \mathcal{C}(V^2 \times I_{k-2})$ is said to be positive if for all $Q \in I_{k-2}$, the kernel $(x, y) \mapsto T(x, y, Q)$ on V^2 is positive. The set of all such operators is denoted by $\mathcal{C}(V^2 \times I_{k-2})_{\succeq 0}$. Define the following operator $B_k : \mathcal{C}(V^2 \times I_{k-2}) \to \mathcal{C}(I_k \setminus \{\emptyset\})$:

$$B_k T(S) = \sum_{\substack{Q \subseteq S \\ |Q| \leq k-2}} \sum_{\substack{x,y \in S \\ Q \cup \{x,y\} = S}} T(x, y, Q)\,.$$

Then the independence number is upper bounded by the following program [11, Proposition 2.1]:

$$\begin{array}{lll} \inf & M \in \mathbb{R} & \\ \text{s.t.} & T \in \mathcal{C}(V \times V \times I_1)_{\succeq 0}\,, & \\ & B_k T(S) \leq M - 1 & \text{for } S \in I_{=1}\,, \\ & B_k T(S) \leq -2 & \text{for } S \in I_{=2}\,, \\ & B_k T(S) \leq 0 & \text{for } S \in I_{=l}\,, 3 \leq l \leq k\,, \end{array} \tag{6.12}$$

Let $w_i \in \mathbb{R}_+$ be weights for $i = 1, \ldots, N$, and let $w : V \to \{w_i \mid i = 1, \ldots, N\}$ be an assignment of the weights to points in V. To take the weights into account, we change B_k to B'_k, defined by

$$B'_k T(S) = \sum_{\substack{Q \subseteq S \\ |Q| \leq k-2}} \Big(\prod_{z \in Q} w(z)^{1/(|Q|+2)} \Big) \sum_{\substack{x,y \in S \\ Q \cup \{x,y\} = S}} T(x, y, Q) w(x)^{1/(|Q|+2)} w(y)^{1/(|Q|+2)}\,.$$

We use the following program:

$$\begin{array}{lll} \inf & M \in \mathbb{R} & \\ \text{s.t.} & T \in \mathcal{C}(V \times V \times I_1)_{\succeq 0}\,, & \\ & B'_k T(S) \leq w(S)(M - w(S)) & \text{for } S \in I_{=1}\,, \\ & B'_k T(S) \leq -2w(S) & \text{for } S \in I_{=2}\,, \\ & B'_k T(S) \leq 0 & \text{for } S \in I_{=l}\,, 3 \leq l \leq k\,, \end{array} \tag{6.13}$$

where $w(S) = \prod_{x \in S} w(x)$.

Proposition 6.6. *Program* (6.13) *gives an upper bound on the weighted independence number with N weights.*

The proof is very similar to the proof of Proposition 6.5.

Proof. Let $T \in \mathcal{C}(V^2 \times I_1)_{\succeq 0}$ be a feasible solution to program (6.13), and let $C = \{x_1, \ldots, x_m\}$ be an independent set in V, with weights w_i and weight assignment $w : V \to \{w_i \mid i = 1, \ldots, N\}$.

Since T is positive, we have

$$\begin{aligned}\sum_{\substack{S\subseteq C\\ |S|\leq k, S\neq\emptyset}} B'_k T(S) &= \sum_{\substack{S\subseteq C\\ |S|\leq k, S\neq\emptyset}} \sum_{\substack{Q\subseteq S\\ |Q|\leq k-2}} \sum_{\substack{x,y\in S\\ Q\cup\{x,y\}=S}} T(x,y,Q)\left(w(Q)w(x)w(y)\right)^{1/(|Q|+2)}\\ &= \sum_{\substack{Q\subseteq C\\ |Q|\leq k-2}} w(Q)^{1/(|Q|+2)} \sum_{x,y\in C} T(x,y,Q)(w(x)w(y))^{1/(|Q|+2)} \geq 0\,,\end{aligned}$$

because $(x,y)\mapsto T(x,y,Q)$ is a positive kernel for all $Q\in I_{k-2}$. Furthermore, because T is feasible and C is an independent set, we have

$$\sum_{\substack{S\subseteq C\\ |S|\leq k, S\neq\emptyset}} B'_k T(S) \leq \sum_{x\in C} w(x)(M-w(x)) - \sum_{x\neq y\in C} w(x)w(y) = \sum_{x\in C} w(x)\left(M - \sum_{y\in C} w(y)\right)\,.$$

Note that the second constraint is split into (x,y) and (y,x), each using $-w(S)$ of the right hand side. Together, this gives $M\geq\sum_{x\in C} w(x)$. □

This is particularly useful for sets $V=V'\times\{1,\ldots,N\}$, where each $i=1,\ldots,N$ of the second set has a weight (such as in spherical cap packing with N sizes of caps). However, to obtain a solvable SDP, it is required that $T(x,y,Q)$ is representable by semidefinite matrices. Furthermore, even with symmetry reduction, the number of constraints will be proportional to N^2, due to the factor $w(S)$ in the second constraint set. If the last constraints cannot be fully reduced, which is the case for spherical cap packing with N sizes of caps, the number of constraints can even be proportional to N^k.

6.6 Conversion to MPMP and time complexity

All two and three-point bounds introduced in the previous sections can easily be formulated as MPMP constraints, similar to the conversion in Section 5.1. Let us give a quick recap of the most important constraints. The polynomial constraints are nearly of the required form; rewriting it in terms of single variables (the matrix elements of F_k) is enough to identify the P_i^j of the general MPMP. The constraints that might not be clear are the constraints on constants. This can be seen as a polynomial constraint of degree 0, hence requiring 1 sample point.

Recall that the complexity per iteration of CLRS is given by

$$\mathcal{O}\left(N_v^3 + \sum_j \left(N_v^2 m_j^2 K_j + N_v m_j^4 K_j^2 + m_j^6 K_j^3 + \sum_{l=1}^{L_j}\left(\eta_{j,l} m_j^2 K_j U_{j,l}^2 + \eta_{j,l}^2 m_j^4 K_j^2 U_{j,l}\right)\right)\right),$$

where N_v is the number of variables, m_j are the matrix sizes of the constraints, $U_{j,l}$ is the number of basis polynomials, K_j is the number of sample points, and $\eta_{j,l}$ is the rank of the polynomial weights; we directly used $P_j=\mathcal{O}(m_j^2 K_j)$. To make the difference clear between the number of scalar variables and the number of sizes of spherical caps,

we write N_v for the number of variables. We will apply this to get the complexities of computing the bounds for spherical cap packing. As the most interesting case is how the complexity changes with the degree d, we will only consider the term with the highest power of d. The programs for $N = 1$ are special cases of the programs with $N > 1$, and thus they will not be considered separately. We will use a maximum degree of $2d$. This is commonly done, because it leads to bases with degree d for the sums of squares factors. This introduces constants in the time complexity, which can be hidden in the big $\mathcal{O}$ notation.

For the two-point bound (6.7), the variables are $f_k \in \mathcal{S}^N$, hence it requires $\frac{N(N+1)}{2}(2d+1) = \mathcal{O}(N^2 d)$ variables to model $f_0, \ldots, f_{2d}$. Additionally, M is a variable which does not depend on d. For each $i = 1, \ldots, N$, there is one constraint on the diagonal value of $f(0)$, giving N constraints of constants. Furthermore, there are $\frac{N(N+1)}{2}$ polynomial constraints of degree $2d$, in $n = 1$ variable. These constraints give a complexity of $\mathcal{O}(N^2 d^3)$ for computing S. Furthermore, the complexity of computing the Cholesky decomposition of Q is $\mathcal{O}((N^2 d)^3) = \mathcal{O}(N^6 d^3)$. As N is kept constant to obtain bounds, only the complexity in d is of interest.

The three-point bound (6.11) requires variables $F_k^l \in \mathcal{S}^{N(2d-k+1)}$ and $A_k \in \mathcal{S}^N$, for $k = 0, \ldots, 2d$ and $l = 1 \ldots, N$, and the variable M for the objective. This gives a total number of

$$N_v = N \sum_{k=0}^{2d} \frac{N(2d-k+1)(N(2d-k+1)+1)}{2} + \frac{N(N+1)}{2} 2d + 1 = \mathcal{O}(N^3 d^3)$$

variables, giving already a complexity of $\mathcal{O}(N^9 d^9)$ for the Cholesky decomposition of Q in the algorithm. This is the most significant part in the complexity, although the other parts such as computing S, $C^{-1}B$ and Q have a similar complexity in d. As an example we will compute the complexity for computing the S block corresponding to the multivariate constraints: There are $\mathcal{O}(N^3)$ constraints with three variables have $K_j = \binom{3+2d}{3} = \mathcal{O}(d^3)$ and $U_j = \binom{3+2}{3} = \mathcal{O}(d^3)$, $L_j = 8$, giving a complexity of $\mathcal{O}(N^3 d^9)$. Note that, because the degree of the polynomials varies from 0 to 3, the actual values of $U_{j,l}$ will differ per block l. This gives a lower constant in front of the d^9, but does not change the maximum power of d.

In this case, SDPA-GMP gives a complexity of $\mathcal{O}(N^5 d^{12})$, due to the term $m_j^5 K_j U_{j,l}^3$. Again, the term m_j^5 comes from including the $E_{r,s}$, which gives $m_j^2 K_j$ constraints instead of K_j. However, note that in the multivariate case with the monomial basis, the constraint matrices will be very sparse when using coefficient matching. This means that the practical complexity will be greatly reduced.

6.6.1 Speedup due to symmetry reduction

In [23, 13], polynomial symmetry was used to reduce the size of the SDP, after which it was solved with SDPA-GMP. Here we will consider the improvement made by using symmetry with CLRS. First we consider the computation of the block of S corresponding to the multivariate constraint, costing $\mathcal{O}(\sum_l m_j U_{l,j}^2 K_j + m_j^2 U_{j,l} K_j^2)$ in complexity. Recall

that the constraint has $n = 3$ variables, and that it is written as

$$\begin{aligned}-F(u,v,t) = q_0(u,v,t) + p(u)q_1(u,v,t) + p(v)q_2(u,v,t) \\ + p(t)q_3(u,v,t) + (1 + 2uvt - u^2 - v^2 - t^2)q_4(u,v,t)\,,\end{aligned}$$

without symmetry reduction, where $p(u)$ has degree 2. Hence $\deg q_0 = 2d, \deg q_i = 2(d-1)$ for $i = 1,2,3$ and $\deg q_4 = 2(d-2)$. Note that the constants in the time complexity will be the same for both the symmetry reduced case as the normal case. Hence comparing them can be done by computing and dividing $\sum_l U_{l,j}K_j^2 + U_{j,l}^2 K_j$. Without symmetry reduction, the basis has a length of $U_d = \binom{d+3}{3}$ for a sum-of-squares polynomial of degree 2d (and thus a basis up to degree d). Furthermore, for equality we need equality on $K_d = \binom{2d+3}{3}$ sample points. This gives, up to a factor, a computation time of

$$K_d(U_d^2 + 3U_{d-1}^2 + U_{d-2}^2) + K_d^2(U_d + 3U_{d-1} + U_{d-2})\,.$$

With symmetry reduction, we can write the constraint as, with F of degree $2d$,

$$\begin{aligned}-F(u,v,t) = \langle Q_1^0, b_d b_d^T \otimes \Pi_1\rangle + \langle Q_2^0, b_{d-3} b_{d-3}^T \otimes \Pi_2\rangle + \langle Q_3^0, b_{d-2} b_{d-2}^T \otimes \Pi_3\rangle \\ + \langle Q_1^1, s_1 b_{d-1} b_{d-1}^T \otimes \Pi_1\rangle + \langle Q_2^1, s_1 b_{d-4} b_{d-4}^T \otimes \Pi_2\rangle + \langle Q_3^1, s_1 b_{d-3} b_{d-3}^T \otimes \Pi_3\rangle \\ + \langle Q_1^2, s_2 b_{d-2} b_{d-2}^T \otimes \Pi_1\rangle + \langle Q_2^2, s_2 b_{d-5} b_{d-5}^T \otimes \Pi_2\rangle + \langle Q_3^2, s_2 b_{d-4} b_{d-4}^T \otimes \Pi_3\rangle \\ + \langle Q_1^3, s_3 b_{d-3} b_{d-3}^T \otimes \Pi_1\rangle + \langle Q_2^3, s_3 b_{d-6} b_{d-6}^T \otimes \Pi_2\rangle + \langle Q_3^2, s_3 b_{d-5} b_{d-5}^T \otimes \Pi_3\rangle \\ + \langle Q_1^4, s_4 b_{d-2} b_{d-2}^T \otimes \Pi_1\rangle + \langle Q_2^4, s_4 b_{d-5} b_{d-5}^T \otimes \Pi_2\rangle + \langle Q_3^4, s_4 b_{d-4} b_{d-4}^T \otimes \Pi_3\rangle\,.\end{aligned}$$

because $\deg s_1 = 2, \deg s_2 = 4, \deg s_3 = 6$ and $\deg s_4 = 3$, with Π_i as defined before. Recall that Π_1, Π_2 are of size 1 and Π_3 is of size 2. Furthermore, b_d is a basis of the invariant polynomials in variables $u+v+t$, $uv+vt+ut$ and uvt up to total degree d in u, v, t. Such a vector of basis polynomials has length

$$U_d^s = \sum_{i=0}^{d} \sum_{j=0}^{\lfloor (d-i)/2 \rfloor} \sum_{k=0}^{\lfloor \frac{d-i-2j}{3} \rfloor} 1 = \sum_{i=0}^{d} \sum_{j=i}^{d-i} \sum_{k=j}^{d-i-j} 1\,,$$

where the first expression is obtained from considering all possible powers of the variables $u+v+t$, $uv+vt+ut$ and uvt, and the second expression is obtained from considering the unisolvent set of rational points in the simplex with denominator d. Because the basis is symmetric, a point (u,v,t) represents all permutations $\sigma(u,v,t)$, and hence the rational points in the simplex with $u \le v \le t$ are a unisolvent set for the symmetrized case. Empty sums (for example $j > d-i-j$ in the second expression) are understood to evaluate to 0.

Note that, for Q_3^i, the length of the final vectors in CLRS is $2U_d^s$ because Π_3 is of size 2. We also require only $K_d^s = U_{2d}^s$ sample points instead of $K_d = U_{2d}$, due to the symmetry. This gives a computation time, up to the same factor as for the non-symmetric case, of

$$\begin{aligned}K_d^s((U_d^s)^2 + (U_{d-1}^s)^2 + 10(U_{d-2}^s)^2 + 10(U_{d-3}^s)^2 + 17(U_{d-4}^s)^2 + 10(U_{d-5}^s)^2 + (U_{d-6}^s)^2) \\ + (K_d^s)^2(U_d^s + U_{d-1}^s + 10U_{d-2}^s + 10U_{d-3}^s + 17U_{d-4}^s + 10U_{d-5}^s + U_{d-6}^s)\,,\end{aligned}$$

where we grouped together all terms with the same U. The terms with Π_3 where counted 8 times; 2 times because the vectors are twice as long, and 2^2 times because the matrix is of rank 2 and thus requires 4 times the number of computations. Note that this is valid for $d \geq 6$; for lower d not every term has to be taken into account.

The second operation in which a speedup can be seen is in the computation of $Q = (C^{-1}B)^T C^{-1}B$. This involves multiplication of matrices of size $N \times K_j$ and $K_j \times N$, hence the speedup will be proportional to K_d/K_d^s. Note that other processes will be sped up too, such as the Cholesky decomposition of S_j and inverting X, because there are less sample points needed and the largest matrix blocks are reduced in size. The actual speedup depends on the interplay between these speedups and the parts corresponding to other constraints, or which only depend on the number of variables y_i. Experiments show that the Cholesky decomposition of Q becomes the bottleneck for the three-point bound. This is especially the case when using multiple threads, because it cannot be easily parallellized.[1] As the Cholesky decomposition of Q does not depend on the basis or the sample points but purely on the number of free variables, it does not profit from the symmetrization.

To compare the two cases, we plot in Figure 3 the computation time for the non-symmetric case divided by the computation time for the symmetric case versus d, for the most important computations which are sped up. With SDPA-GMP, the highest d for which computations where done was $d = 16$ by Machado and De Oliviera Filho [23], although a slightly different program was used. Therefore we plot it up to $d = 30$, which will not be reached in the near future.

6.7 Experiments concerning sampling with symmetry reduction

To use the solver introduced in Chapter 3 for the computations, one needs to choose bases and sample points for each constraint. In this section, we will consider the multivariate, symmetrized constraints, and show what problems can be encountered. In particular, the case of $n = 3$ and degree $2d$ was considered, using $N = 1$ caps with angle $\theta_1 = \pi/3$. This problem is equivalent to the kissing number τ_n. It is known that $\tau_3 = 12$.

Recall that, for $N = 1$, the multivariate constraints are symmetric in u, v and t. We use symmetry reduction as explained in Section 6.4 to reduce the number of required sample points and the length of the bases. In [23], the basis polynomials used were the polynomials $(u + v + t)^a (uv + vt + ut)^b (uvt)^c$ with $a + 2b + 3c \leq d$. We apply a basis change to obtain the basis with polynomials $P_a^n(u + v + t)P_b^n(uv + vt + ut)P_c^n(uvt)$ with the same requirement for a, b, c, where P_a^n are the Gegenbauer polynomials of degree a with parameter $n/2 - 1$ as before. This in general improves the solution quality. When using coefficient matching, this also reduces the sparsity and thus the speed of the solver SDPA-GMP.

[1] There are parallellized algorithms for the Cholesky factorization, see for example [15]. Such algorithms are not available for arbitrary precision in the standard libraries for Julia, and do not seem often used. In general, such parallellized algorithms are less efficient, and thus a speedup is only seen for a high number of cores and large matrices.

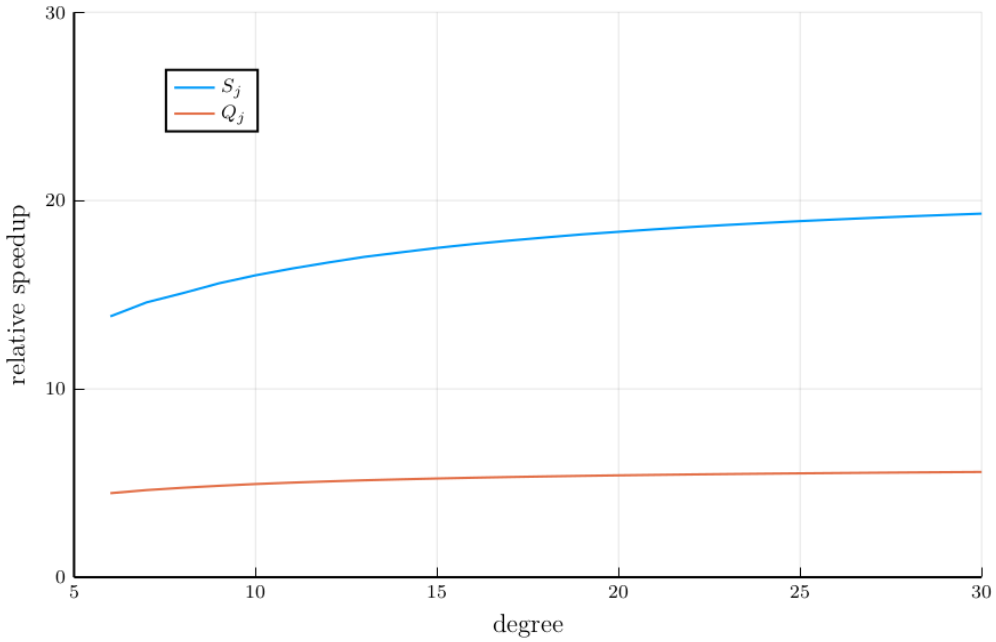

Figure 3: The theoretical relative speedup for the computation of S_j due to symmetry reduction, corresponding to bases of degree d, and hence a program with a maximum degree of $2d$.

We tried several methods to choose sample points, because we noted that the choice of the sample points can greatly influence the value of the solution. Some of the solutions are clearly not solutions to the initial problem, as the value does not upper bound the kissing number. We give a number of the methods with their results.

6.7.1 Rational points in the simplex

As mentioned in Section 4.6.5, one method to obtain sample points is to take the rational points in the unit simplex with denominator $2d$, where the polynomials have degree $2d$. To symmetrize this, only the points (u, v, t) with $u \leq v \leq t$ were taken into account for the solving. This gives for $n = 3$ and $d = 6$ a value of 5.685, although the number of spherical caps with angle $\theta = \pi/3$ is known to be 12 for $n = 3$.

6.7.2 Approximately maximizing the Vandermonde matrix

We tried several methods using the greedy maximization of the Vandermonde matrix, with different starting sets. The basis of Gegenbauer polynomials in the symmetrized variables was used in each case.

Chebyshev points with increasing degree

Following the Padua points, we chose a subset of $C_{2d+1} \times C_{2d+2} \times C_{2d+3}$ which approximately maximizes the Vandermonde matrix. In this case, the Chebyshev points were taking in the interval $[-1, \cos(\theta)]$. Although this often gives a good unisolvent set for polynomial interpolation, it was not possible to compute the bound for this set of sample points. Because every point was of the form (u, v, t) with $u \neq v \neq t \neq u$, the set represented a too large number of points in the non-symmetrized basis. This seemed to introduce implicit linear dependencies. In case of linear dependencies, the Schur complement matrix S becomes singular. This in general means that it has a small negative eigenvalue due to the use of floating point numbers, which makes the Cholesky decomposition impossible.

Chebyshev points with the same degree

As points of the form (u, u, u) and (u, u, v) were needed, the second set we considered was C^3_{4d+1} with the requirements as explained in Section 4.6.5. Note that the size was also increased, to have more possible good sample points. This gives a value of 12.36 for $d = 6$. Although this is an upper bound, this is lower than the bound given by [23] for $d = 16$, and thus cannot be expected to be the value for $d = 6$.

Combining Chebyshev points of the same degree with increasing degree

We combined the two previous methods; the points with $u = v = t$ which were needed were chosen from C^3_{4d+1}, the points with $u = v \neq t$ were chosen from $C^2_{4d+1} \times C_{4d+2}$ and the points with u, v and t distinct were chosen from $C_{4d+1} \times C_{4d+2} \times C_{4d+3}$. This further increased the value to 12.51, although computing the value for the bound with SDPA-GMP and coefficient matching gives 12.71 for $d = 6$.

6.7.3 Results for higher degrees d

We used the last mentioned method of combining Chebyshev points of multiple degrees with degrees starting from $3d + 1$ to obtain values for $d = 6, \ldots, 10$. This gave results as stated in Table 3. Note that, even though the bound should be decreasing, the value increases when increasing d from for example 6 to 7. The exact reason for this is unclear; a possible cause is the bad conditioning of the Vandermonde matrices, i.e., a determinant of 10^{-130} to 10^{-900} in absolute value.

d	6	7	8	9	10
value	12.11	12.47	12.40	12.52	12.59

Table 3: Optimal values returned by CLRS when using bases of length d. Sample points were chosen as subset of C^3_{3d+1}, $C^2_{3d+1} \times C_{3d+2}$ or $C_{3d+1} \times C_{3d+2} \times C_{3d+3}$ resembling the weight distribution of the symmetrized rational points in the simplex with denominator $2d$.

6.7.4 Further research

Although these results are not important in the context of spherical cap packing (the results are not expected to be correct), they show what has to investigated before the techniques can be used with confidence in the multivariate case. First of all, in what sense does taking sample points in the symmetrized setting, representing more sample points than needed for n-variate polynomials of degree d, introduce linear dependencies? This is not entirely clear, because the solver only 'sees' the symmetrized basis. Invariance theory might be able to provide answers to this and related questions. Secondly, and more practical, what is a good set of sample points to choose or a good heuristic to choose sample points in multivariate polynomial optimization, when considering invariant polynomials? This is intertwined with the problem of choosing a good basis, as the combination of basis and sample points determine the conditioning of the Vandermonde matrix.

7. Conclusion and recommendations

This thesis focuses mainly on two parts. In the first part, we introduced the clustered, low-rank semidefinite program, together with the solver CLRS. CLRS is based on SDPB [29] and exploits the structure of the problem for faster computations similar to, but more general than, SDPB. This was applied to multivariate polynomial (matrix) programs, which give such a structure. The time complexity results are promising, showing the expected speedup.

The solver CLRS currently uses the C library Arb [17] for most of the computations, through the Julia library Arblib. It does not seem to be possible to explicitely use multiple threads for operations such as matrix multiplication, which may be possible using other arbitrary precision arithmetic libraries. Furthermore, Arb uses ball-arithmetic, which is unstable for operations such as Cholesky decompositions on near to singular matrices. For most other operations, it is possible to use versions without error bounds.

In the second part, we give two applications of the multivariate polynomial matrix programs. We compute the Cohn-Elkies bound on the sphere packing density for several polynomial degrees to compare the speed of SDPA-GMP, SDPB and CLRS. This shows that, the solver can compete with and surpass SDPA-GMP in the type of problems considered with one thread. In addition, the more general Julia solver CLRS is slightly faster than the C++ solver SDPB for one thread, although it uses extra threads less efficiently. For the example problem, CLRS and SDPB are clearly faster than SDPA-GMP when using multiple threads, which is not possible to do with SDPA-GMP.

Furthermore, the problem on spherical cap packing with N sizes of caps is considered. We give a new three-point bound for N caps, extending the three-point bound for equal sized caps of Bachoc and Vallentin [1, 2] and the two-point bound for N sizes of caps of de Laat et al. [10].

The three-point bound can be solved faster with symmetry reduction, which leads to the question how multivariate sampling combined with semidefinite programming can be done in the setting of invariance theory. It is unclear how a good unisolvent set of sample points can be chosen without introducing implicit linear dependencies. Although it has been encountered in this thesis for S_3 symmetries, other group symmetries may have similar issues. Further research is needed for the combination of sampling for semidefinite programming and invariance theory.

As a last remark, it would be interesting if other applications beside polynomial optimization can be found for the solver.

A. Proof of Theorem 4.1

In this appendix we will give a detailed proof of Theorem 4.1:

Theorem 4.1 ([27, 19])**.** *Let $f \in \mathbb{R}[x]^{m\times m}$, $G \subseteq \mathbb{R}[x]$. Suppose M_G is archimedean. If $f \succ 0$ on S_G, then $f \in M_G$.*

We follow the proof of [19]. Let $L : \mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m}) \to \mathbb{R}$ be a linear functional. L is called a state on $(\mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m}), M_G)$ if $L(M_G) \subseteq \mathbb{R}_+$, and $L(I) = 1$. L is a pure state if it is a state and it is an extreme point of the convex set of all states. Furthermore, we recall the definition of an algebraic interior point of a convex set A in a vector space X [3]: $a \in A \subseteq X$ is an algebraic interior point of A if $\forall x \in X\, \exists \varepsilon > 0 : a + \varepsilon x \in A$.

We also state Theorem 9 of [19], which we will use in the proof of Theorem 4.1.

Theorem A.1 ([19, Theorem 9])**.** *Suppose $G \subseteq \mathbb{R}[\bar{x}]^{m\times m}$, and M_G is archimedean. For each pure state L on $(\mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m}), M_G)$, there exists $x \in S_G$ and a unit vector $v \in \mathbb{R}^m$ such that*

$$L(p) = v^T p(x) v \qquad \text{for all } p \in \mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m})\,.$$

Proof of Theorem 4.1. Suppose M_G is archimedean, and $f \notin M_G$. We will prove that I is an interior point of M_G in order to separate $f\mathbb{R}_{>0}$ and M_G. Let $p \in \mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m})$. As M_G is archimedean, there is an $N \in \mathbb{N}$ such that $NI - p^2 \in M_G$. M_G is a convex cone, which implies that

$$(NI - p^2) + (p + I)^2 = NI + 2p + I \in M_G\,.$$

So $I + \frac{2}{N+1} p \in M_G$, i.e. I is an algebraic interior point of M_G.

By separation Theorem III.1.7 of Barvinok's *A Course in Convexity* [3], there is a linear functional L with $L(M_G) \geq 0$ and $L(f\mathbb{R}_{>0}) \leq 0$. By scaling, L is a state on $(\mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m}), M_G)$.

We will prove that the set of such states is weak* compact. Note that, because p is symmetric with real coefficients, we have $L((p \pm I)^T (p \pm I)) = L(p^T p) \pm 2L(p) + L(I) \geq 0$, hence $|L(p)| \leq \frac{1}{2}(L(p^T p) + 1)$. Because M_G is archimedean, there is an $N \in \mathbb{N}$ such that $N - p^T p \in M_G$ for every $p \in \mathbb{R}[\bar{x}]^{m\times m}$. Hence $L(N - p^T p) \geq 0$, which gives the bound $L(p^T p) \leq N$ using the linearity and the definition of a state. Define $\mathcal{I}_p = [-\frac{1}{2}(N_p + 1), \frac{1}{2}(N_p + 1)]$ and

$$C \coloneqq \prod_{p \in \mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m})} \mathcal{I}_p\,,$$

where $N_p \in \mathbb{N}$ denotes the smallest constant, only dependent on p, for which $N_p - p^T p \in M_G$. As a direct product of compact intervals, C is compact in the product topology by Tikhonov's theorem. We can identify the states on $(\mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m}), M_G)$ with a closed subset of C; every linear functional L can be identified with a vector $(L(p))_p$, and the set of such vectors is sequentially closed because of continuity of the linear functionals L. Note that, because the product topology is the weakest topology where projections

T_p are continuous, this identification defines a bijection between the weak* topology on $(\mathrm{Sym}(\mathbb{R}[\bar{x}]^{m\times m}), M_G)$ (open sets are unions of finite intersections of the basic open sets $U(p,\alpha,\beta) = \{L \mid \alpha < L(p) < \beta\}$), and the product topology.

The set of states with $L(f) \leq 0$ is a closed subset of C, which is thus also compact in the product topology. The product topology is locally convex, hence we can use the Krein-Milmann Theorem [3, Theorem III.4.1] to choose L to be pure (i.e. an extreme point). By Theorem A.1, $L(f) = v^T f(x) v \leq 0$ for some $x \in S_G$ and unit vector $v \in \mathbb{R}^m$, hence $f \not\succ 0$. □

B. Improvements of Theorem 4.1 in the univariate case

In this appendix we consider the univariate case of Theorem 4.1, with G consisting of polynomials rather than polynomial matrices. This allows for equivalence between positive semidefinite polynomial matrices on both compact and (semi) infinite intervals. Starting with $S_G = \mathbb{R}$, we have the following theorem, which is equivalent to the Biform Theorem of [7, Theorem 7.1] by considering the homogenized version of $x^T f(y)x$.

Theorem B.1. *Let $f \in \mathbb{R}[x]^{m\times m}$. Then $f \succeq 0$ on $\mathbb{R}$ if and only if $f = S(x)^T S(x)$ for some $S(x) \in \mathbb{R}[x]^{t\times m}$ with $t \leq 2m$.*

Note that the degree of S is at most $\lfloor \frac{\deg(f)}{2} \rfloor$, because terms of the highest degree cannot cancel on the diagonal, and $f(x) \succeq 0$ when $x \to \infty$.

This can be extended to $S_G = \mathbb{R}_+$, or more general $S_G = [a, \infty)$ by a translation.

Theorem B.2 ([29, Theorem 2.1])**.** *Let $f \in \mathbb{R}[x]^{m\times m}$. $f \succeq 0$ on $\mathbb{R}_+$ if and only if $f = s_0 + xs_1$, where $s_0, s_1 \in M_\emptyset$. Furthermore,* $\deg(s_0) \leq \deg(f)$ *and* $\deg(xs_1) \leq \deg(f)$.

Proof. The 'if' direction is clear. The converse follows from Theorem B.1 by taking $x = \hat{x}^2$. Define $\hat{f}(x) = f(x^2)$. Then $\hat{f} \succeq 0$ on $\mathbb{R}$, so by Theorem B.1 we have

$$\hat{f}(x) = S(x)^T S(x) = \sum_{i=1}^{N} s_i(x)^T s_i(x) = \sum_{i=1}^{N} (s_i^0(x) + xs_i^1(x))^T (s_i^0(x) + xs_i^1(x)) \,,$$

where $s_i^j(x)$ has only even powers. One such a product gives the terms

$$s_i^0(x)^T s_i^0(x) + x^2 s_i^1(x)^T s_i^1(x) + x(s_i^0(x)^T s_i^1(x) + s_i^1(x)^T s_i^0(x)) \,.$$

Note that, because $\hat{f}(x) = f(x^2)$ and f is a polynomial matrix, $\hat{f}$ has only even powers of x. As all powers in the cross term are odd, we obtain

$$\sum_{i=1}^{N} x(s_i^0(x)^T s_i^1(x) + s_i^1(x)^T s_i^0(x)) = 0 \,.$$

Hence we have

$$f(x) = \sum_{i=1}^{N} \left(s_i^0(\sqrt{x})^T s_i^0(\sqrt{x}) + xs_i^1(\sqrt{x})^T s_i^1(\sqrt{x}) \right) = S_0(x)^T S_0(x) + xS_1(x)^T S_1(x) \,.$$

The bound on the degree is clear from the fact that terms on the diagonal with the highest degree cannot cancel, and the term with the highest degree is on the diagonal because $f \succeq 0$ on $\mathbb{R}_+$. □

Theorem B.3. *Let $f \in \mathbb{R}[x]^{m\times m}$. Then $f \succeq 0$ on $[a, b]$ if and only if*

- $f(x) = s_0(x)+(b-x)(x-a)s_1(x)$ *with* $\deg(s_0(x)), \deg((b-x)(x-a)s_1(x)) \leq \deg(f)$ *resp.* $\deg(f)+1$ *when* $\deg(f)$ *is even resp. odd.*
- *if* $\deg(f)$ *is odd,* $f = (x-a)s_0(x) + (b-x)s_1(x)$ *with* $\deg((x-a)s_0(x)), \deg((b-x)s_1(x)) \leq \deg(f)$

where $s_0, s_1 \in M_\emptyset$.

Note that this is Lukács Theorem for polynomial matrices. We will prove it in a similar fashion as the polynomial version in [21].

Proof. The 'if' direction is clear, as s_0 and s_1 are positive semi-definite on $\mathbb{R}$.

Conversely, suppose $f \succeq 0$ on $[a,b]$. By an affine transformation, we may assume that $[a,b] = [-1,1]$. Define the matrix Goursat transform as:

$$G(f)(x) = (1+x)^d f(\frac{1-x}{1+x}),$$

where $d = \deg(f)$ and $f \in \mathbb{R}[x]^{m\times m}$. Similar to the normal Goursat transform for polynomials, $G(f) \succeq 0$ on $\mathbb{R}_+$ if and only if $f \succeq 0$ on $[-1,1]$, and $G(G(f)) = 2^d f$. Both properties are easily verified. As $f \succeq 0$ on $[-1,1]$ by assumption, $G(f) \succeq 0$ on $\mathbb{R}_+$. Hence $G(f) = S_0(x)^T S(x) + xS_1(x)^T S_1(x)$ for some polynomial matrices S_1 and S_0 by Theorem B.2. Taking the Goursat transform, we obtain

$$\begin{aligned} 2^d f(x) &= (1+x)^d (S_0(\frac{1-x}{1+x})^T S_0(\frac{1-x}{1+x}) + \frac{1-x}{1+x} S_1(\frac{1-x}{1+x})^T S_1(\frac{1-x}{1+x})), \\ &= (1+x)^{d-2d_0} G(S_0)^T G(S_0) + (1-x)(1+x)^{d-2d_1-1} G(S_1)^T G(S_1), \end{aligned}$$

where $d_i = \deg(S_i)$.

Suppose d is even. Then this clearly admits the required form, with

$$\begin{aligned} s_0(x) &= \left((1+x)^{d/2-d_0} G(S_0)\right)^T \left((1+x)^{d/2-d_0} G(S_0)\right), \\ s_1(x) &= \left((1+x)^{d/2-d_1-1} G(S_1)\right)^T \left((1+x)^{d/2-d_1-1} G(S_1)\right). \end{aligned}$$

Note that the degree bound is satisfied because $\deg(s_0) \leq 2(d/2 - d_0 + d_0) = d$ and similarly $\deg((1-x^2)s_1) \leq d$.

Suppose d is odd. Then the first term is $(1+x)$ times a sum of squares of polynomial matrices, and the second term is $(1-x)$ times a sum of squares, giving the second decomposition with the degree bound. Moreover, using the identities

$$\begin{aligned} 1+x &= \frac{(1+x)^2}{2} + \frac{1-x^2}{2} \\ 1-x &= \frac{(1-x)^2}{2} + \frac{1-x^2}{2} \end{aligned}$$

this decomposition can be rewritten to the first form with the required degree bound. □

References

[1] C. Bachoc and F. Vallentin. New upper bounds for kissing numbers from semidefinite programming. *arXiv:math/0608426*, Oct. 2007. arXiv: math/0608426.

[2] C. Bachoc and F. Vallentin. Semidefinite programming, multivariate orthogonal polynomials, and codes in spherical caps. *European Journal of Combinatorics*, 30(3):625–637, Apr. 2009. arXiv: math/0610856.

[3] A. Barvinok. *A Course in Convexity*, volume 54 of *Graduate Studies in Mathematics*. American Mathematical Society, 2002.

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, Jan. 2017. Publisher: Society for Industrial and Applied Mathematics.

[5] S. Bochner. Hilbert Distances and Positive Definite Functions. *Annals of Mathematics*, 42(3):647–656, 1941. Publisher: Annals of Mathematics.

[6] M. Caliari, M. Vianello, A. Sommariva, and S. De Marchi. Padua2DM: Fast interpolation and cubature at the Padua points in Matlab/Octave. *Numerical Algorithms*, 56(1):45–60, Mar. 2011.

[7] M.-D. Choi, T.-Y. Lam, and B. Reznick. Real zeros of positive semidefinite forms. I. *Mathematische Zeitschrift*, 171(1):1–26, Feb. 1980.

[8] H. Cohn and N. Elkies. New upper bounds on sphere packings I. *arXiv:math/0110009*, Sept. 2003. arXiv: math/0110009.

[9] D. de Laat. Moment methods in energy minimization: New bounds for Riesz minimal energy problems. *Transactions of the American Mathematical Society*, 373(2):1407–1453, Oct. 2019. arXiv: 1610.04905.

[10] D. de Laat, F. M. de Oliveira Filho, and F. Vallentin. Upper bounds for packings of spheres of several radii. *Forum of Mathematics, Sigma*, 2:e23, Sept. 2014.

[11] D. de Laat, F. C. Machado, F. M. de Oliveira Filho, and F. Vallentin. k-point semidefinite programming bounds for equiangular lines. *arXiv:1812.06045 [math]*, Nov. 2019. arXiv: 1812.06045.

[12] P. Delsarte, J. M. Goethals, and J. J. Seidel. Spherical codes and designs. *Geometriae Dedicata*, 6(3):363–388, Sept. 1977.

[13] M. Dostert, D. de Laat, and P. Moustrou. Exact semidefinite programming bounds for packing problems. *arXiv:2001.00256 [math]*, July 2020. arXiv: 2001.00256.

[14] K. Gatermann and P. A. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. *arXiv:math/0211450*, Nov. 2002. arXiv: math/0211450.

[15] A. George, M. T. Heath, and J. Liu. Parallel Cholesky factorization on a shared-memory multiprocessor. *Linear Algebra and its Applications*, 77:165–187, May 1986. Publisher: North-Holland.

[16] D. Hilbert. Ueber die Darstellung definiter Formen als Summe von Formenquadraten. *Mathematische Annalen*, 32(3):342–350, Sept. 1888.

[17] F. Johansson. Arb: Efficient Arbitrary-Precision Midpoint-Radius Interval Arithmetic. *IEEE Transactions on Computers*, 66(8):1281–1292, Aug. 2017. Conference Name: IEEE Transactions on Computers.

[18] L. Khachiyan. On the Complexity of Approximating Extremal Determinants in Matrices. *Journal of Complexity*, 11(1):138–153, Mar. 1995.

[19] I. Klep and M. Schweighofer. Pure states, positive matrix polynomials and sums of hermitian squares. *Indiana University Mathematics Journal*, 59(3):857–874, 2010. arXiv: 0907.2260 version: 3.

[20] W. Landry and D. Simmons-Duffin. Scaling the semidefinite program solver SDPB. *arXiv:1909.09745 [hep-th]*, Sept. 2019.

[21] M. Laurent. Sums of Squares, Moment Matrices and Optimization Over Polynomials. In M. Putinar and S. Sullivant, editors, *Emerging Applications of Algebraic Geometry*, volume 149, pages 157–270. Springer New York, New York, NY, 2009. Series Title: The IMA Volumes in Mathematics and its Applications.

[22] J. Lofberg and P. Parrilo. From coefficients to samples: a new approach to SOS optimization. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, pages 3154–3159, Nassau, 2004. IEEE.

[23] F. C. Machado and F. M. de Oliviera Filho. Improving the semidefinite programming bound for the kissing number by exploiting polynomial symmetry. *arXiv:1609.05167 [math]*, Sept. 2016. arXiv: 1609.05167.

[24] S. Mehrotra. On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization*, 2(4):575–601, Nov. 1992. Publisher: Society for Industrial and Applied Mathematics.

[25] D. Papp and S. Yıldız. Sum-of-squares optimization without semidefinite programming. *arXiv:1712.01792 [math]*, Dec. 2018.

[26] M. Putinar. Positive Polynomials on Compact Semi-algebraic Sets. *Indiana University Mathematics Journal*, 42(3):969–984, 1993. Publisher: Indiana University Mathematics Department.

[27] C. W. Scherer and C. W. J. Hol. Matrix Sum-of-Squares Relaxations for Robust Semi-Definite Programs. *Mathematical Programming*, 107(1):189–211, June 2006.

[28] I. J. Schoenberg. Positive definite functions on spheres. *Duke Mathematical Journal*, 9(1):96–108, Mar. 1942. Publisher: Duke University Press.

[29] D. Simmons-Duffin. A Semidefinite Program Solver for the Conformal Bootstrap. *arXiv:1502.02033 [cond-mat, physics:hep-th]*, Feb. 2015.

[30] A. Sommariva and M. Vianello. Computing approximate Fekete points by QR factorizations of Vandermonde matrices. *Computers & Mathematics with Applications*, 57(8):1324–1336, Apr. 2009.

[31] K.-C. Toh, M. J. Todd, and R. H. Tütüncü. On the Implementation and Usage of SDPT3 – A Matlab Software Package for Semidefinite-Quadratic-Linear Programming, Version 4.0. In M. F. Anjos and J. B. Lasserre, editors, *Handbook on Semidefinite, Conic and Polynomial Optimization*, volume 166, pages 715–754. Springer US, Boston, MA, 2012. Series Title: International Series in Operations Research & Management Science.

[32] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and Evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0). *Optimization Methods and Software*, 18(4 II):491–505, Sept. 2002.

[33] M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto. A high-performance software package for semidefinite programs: SDPA 7. Research Report B-460, Dept. of Mathematical and Computing Science, Tokyo Institute of Technology, Tokyo, Japan, Jan. 2010.