Improved Algorithms for the Approximate k-List Problem in Euclidean Norm

Gottfried Herold, Elena Kirshanova

Faculty of Mathematics
Horst Görtz Institute for IT-Security
Ruhr University Bochum
{gottfried.herold, elena.kirshanova}@rub.de

Abstract. We present an algorithm for the approximate k-List problem for the Euclidean distance that improves upon the Bai-Laarhoven-Stehlé (BLS) algorithm from ANTS'16. The improvement stems from the observation that almost all the solutions to the approximate k-List problem form a particular configuration in n-dimensional space. Due to special properties of configurations, it is much easier to verify whether a k-tuple forms a configuration rather than checking whether it gives a solution to the k-List problem. Thus, phrasing the k-List problem as a problem of finding such configurations immediately gives a better algorithm. Furthermore, the search for configurations can be sped up using techniques from Locality-Sensitive Hashing (LSH). Stated in terms of configuration-search, our LSH-like algorithm offers a broader picture on previous LSH algorithms.

For the Shortest Vector Problem, our configuration-search algorithm results in an exponential improvement for memory-efficient sieving algorithms. For $k = 3$, it allows us to bring down the complexity of the BLS sieve algorithm on an n-dimensional lattice from $2^{0.4812n+o(n)}$ to $2^{0.3962n+o(n)}$ with the same space-requirement $2^{0.1887n+o(n)}$. Note that our algorithm beats the Gauss Sieve algorithm with time resp. space requirements of $2^{0.415n+o(n)}$ resp. $2^{0.208n+o(n)}$, while being easy to implement. Using LSH techniques, we can further reduce the time complexity down to $2^{0.3717n+o(n)}$ while retaining a memory complexity of $2^{0.1887n+o(n)}$.

1 Introduction

The k-List problem is defined as follows: given k lists $L_1, \dots, L_k$ of elements from a set X, find k-tuples $(x_1, \dots, x_k) \in L_1 \times \dots \times L_k$ that satisfy some condition C. For example, Wagner [21] considers $X \subset \{0,1\}^n$, and a tuple $(x_1, \dots, x_k)$ is a solution if $x_1 \oplus \dots \oplus x_n = 0^n$. In this form, the problem has found numerous applications in cryptography [16] and learning theory [6].

For ℓ_2-norm conditions with $X \subset \mathbb{R}^n$ and $k = 2$, the task of finding pairs $(\boldsymbol{x}_1, \boldsymbol{x}_2) \in L_1 \times L_2$, s.t. $\|\boldsymbol{x}_1 + \boldsymbol{x}_2\| < \min\{\|\boldsymbol{x}_1\|, \|\boldsymbol{x}_2\|\}$, is at the heart of certain algorithms for the Shortest Vector Problem (SVP). Such algorithms, called *sieving* algorithms ([1], [19]), are asymptotically the fastest SVP solvers known so far.

Sieving algorithms look at pairs of lattice vectors that sum up to a short(er) vector. Once enough such sums are found, repeat the search by combining these shorter vectors into even shorter ones and so on. It is not difficult to see that in order to find even one pair where the sum is shorter than both the summands, we need an exponential number of lattice vectors, so the memory requirement is exponential. In practice, due to the large memory-requirement, sieving algorithms are outperformed by the asymptotically slower Kannan enumeration [10].

Naturally, the question arises whether one can reduce the constant in the exponent of the memory complexity of sieving algorithms at the expense of running time. An affirmative answer is obtained in the recently proposed k-list sieving by Bai, Laarhoven, and Stehlé [4] (BLS, for short). For constant k, they present an algorithm that, given input lists $L_1, \ldots, L_k$ of elements from the n-sphere S^n with radius 1, outputs k-tuples with the property $\|\boldsymbol{x}_1 + \ldots + \boldsymbol{x}_n\| < 1$. They provide the running time and memory-complexities for $k = 3, 4$.

We improve and generalize upon the BLS k-list algorithm. Our results are as follows:

1. We present an algorithm that on input $L_1, \ldots, L_k$, $L_i \subset \mathsf{S}^n$, outputs k-tuples $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k), \in L_1 \times \ldots \times L_k$, s.t. all *pairs* $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in a tuple satisfy certain inner product constraints. We call this problem the Configuration problem (Def. 3).
2. We give a concentration result on the distribution of scalar products of $\boldsymbol{x}_1, \ldots \boldsymbol{x}_k \in \mathsf{S}^n$ (Thms. 1 and 2), which implies that finding vectors that sum to a shorter vector can be reduced to the above Configuration problem.
3. By working out the properties of the aforementioned distribution, we *prove* the conjectured formula (Eq. (3.2) from [4]) on the input list-sizes (Thm. 3), s.t. we can expect a constant success probability for sieving. We provide closed formulas for the running times for both algorithms: BLS and our Alg. 1 (Thm. 4). Alg. 1 achieves an exponential speed-up compared the BLS algorithm.
4. To further reduce the running time of our algorithm, we introduce the so-called Configuration Extension Algorithm (Alg. 6). It has an effect similar to Locality-Sensitive Hashing as it shrinks the lists in a helpful way. This is a natural generalization of LSH to our framework of configurations. We show how to combine Alg. 1 and the Configuration Extension Algorithm (Alg. 6) in Sect. 7.

Roadmap. Sect. 2 gives basic notations and states the problem we consider in this work. Sect. 3 introduces configurations – a novel tool that aids the analysis in succeeding Sect. 4 and 5 where we present our algorithm for the k-List problem and prove its running time. Our generalization of Locality Sensitive Hashing – Configuration Extension – is described in Sect. 6 and its application to the k-list problem in Sect. 7. We conclude with experimental results confirming our analysis in Sect. 8. We defer some of the proofs and details on the Configuration

Extension Algorithm to the appendices as these are not necessary to understand the main part.

2 Preliminaries

Notations. We denote by $\mathsf{S}^n \subset \mathbb{R}^{n+1}$ the n-dimensional unit sphere. We use soft-$\mathcal{O}$ notation to denote running times: $T = \widetilde{\mathcal{O}}(2^{cn})$ means that we suppress subexponential factors. We use sub-indices $\mathcal{O}_k(.)$ in the $\mathcal{O}$-notation to stress that the asymptotic result holds for k fixed. For any set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ of vectors in some $\mathbb{R}^n$, the *Gram matrix* $C \in \mathbb{R}^{k \times k}$ is given by the set of pairwise scalar products. It is a complete invariant of the $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ up to simultaneous rotation and reflection of all $\boldsymbol{x}_i$'s. For such matrices $C \in \mathbb{R}^{k \times k}$ and $I \subset \{1, \ldots, k\}$, we write $C[I]$ for the appropriate $|I| \times |I|$-submatrix with rows and columns from I.

As we consider distances wrt. the ℓ_2-norm, the approximate k-List problem we consider in this work is the following computational problem:

Definition 1 (Approximate k-List problem). *Let $0 < t < \sqrt{k}$. Assume we are given k lists $L_1, \ldots, L_k$ of equal exponential size, whose entries are iid. uniformly chosen vectors from the n-sphere S^n. The task is to output an $1-o(1)$-fraction of all solutions, where solutions are k-tuples $\boldsymbol{x}_1 \in L_1, \ldots, \boldsymbol{x}_k \in L_k$ satisfying $\|\boldsymbol{x}_1 + \cdots + \boldsymbol{x}_k\|^2 \leq t^2$.*

We consider the case where t, k are constant and the input lists are of size c^n for some constant $\mathsf{c} > 1$. We are interested in the asymptotic complexity for $n \to \infty$. To simplify the exposition, we pretend that we can compute with real numbers; all our algorithms work with sufficiently precise approximations (possibly losing an $o(1)$-fraction of solutions due to rounding). This does not affect the asymptotics. Note that the problem becomes trivial for $t > \sqrt{k}$, since all but an $1 - o(1)$-fraction of k-tuples from $L_1 \times \cdots \times L_k$ satisfy $\|\boldsymbol{x}_1 + \ldots + \boldsymbol{x}_k\|^2 \approx k$ (random $\boldsymbol{x}_i \in \mathsf{S}^n$ are almost orthogonal with high probability, cf. Thm. 1). In the case $t > \sqrt{k}$, we need to ask that $\|\boldsymbol{x}_1 + \ldots + \boldsymbol{x}_k\|^2 \geq t^2$ to get a meaningful problem. Then all our results apply to the case $t > \sqrt{k}$ as well.

In our definition, we allow to drop a $o(1)$-fraction of solutions, which is fine for the sieving applications. In fact, we will propose an algorithm that drops an exponentially small fraction of solutions and our asymptotic improvement compared to BLS crucially relies on dropping more solutions than BLS. For this reason, we are only interested in the case where the expected number of solutions is exponential.

Relation to the approximate Shortest Vector Problem. The main incentive to look at the approximate k-List problem (as in Def. 1) is its straightforward application to the so-called sieving algorithms for the shortest vector problem (SVP) on an n-dimensional lattice (see Sect. 7.2 for a more comprehensive discussion). The complexity of these sieving algorithms is completely determined by the complexity of an approximate k-List solver called as main subroutine. So one can instantiate a lattice sieving algorithm using an approximate k-List solver (the ability to choose k allows a memory-efficient instantiations of such a solver). This is observed and fully explained in [4]. For $k = 3$, the running time for the SVP algorithm presented in [4] is $2^{0.4812n+o(n)}$ requiring $2^{0.1887n+o(n)}$ memory. Running our Alg. 1 instead as a k-List solver within the SVP sieving, one obtains a running time of $2^{0.3962n+o(n)}$ with the same memory complexity $2^{0.1887n+o(n)}$. As explained in Sect. 7.2, we can reduce the running time even further down to $2^{0.3717n+o(n)}$ with no asymptotic increase in memory by using a combination of Alg. 1 and the LSH-like Configuration Extension Algorithm (see Alg. 6 in App. A).

In the applications to sieving, we have $t = 1$ and actually look for solutions $\|\pm \boldsymbol{x}_1 \pm \cdots \pm \boldsymbol{x}_k\| \leq 1$ with arbitrary signs. This is clearly equivalent by considering the above problem separately for each of the $2^k = \mathcal{O}(1)$ choices of signs. Further, the lists $L_1, \ldots, L_k$ can actually be equal. Our algorithm works for this case as well. In these settings, some obvious optimizations are possible, but they do not affect the asymptotics.

Our methods are also applicable to lists of different sizes, but we stick to the case of equal list sizes to simplify the formulas for the running times.

3 Configurations

Whether a given k-tuple $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ is a solution to the approximate k-List problem is invariant under simultaneous rotations/reflections of all $\boldsymbol{x}_i$ and we want to look at k-tuples up to such symmetry by what we call configurations of points. As we are concerned with the ℓ_2-norm, a complete invariant of k-tuples up to symmetry is given by the set of pairwise scalar products and we define configurations for this norm:

Definition 2 (Configuration). *The* configuration $C = \operatorname{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ *of* k *points* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathsf{S}^n$ *is defined as the Gram matrix* $C_{i,j} = \langle \boldsymbol{x}_i\,, \boldsymbol{x}_j \rangle$.

Clearly, the configuration of the k-tuple $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ determines the length of the sum $\|\sum_i \boldsymbol{x}_i\|$:

$$\Big\|\sum_i \boldsymbol{x}_i\Big\|^2 = \sum_{i,j} \langle \boldsymbol{x}_i\,, \boldsymbol{x}_j \rangle = k + 2 \sum_{i<j} \langle \boldsymbol{x}_i\,, \boldsymbol{x}_j \rangle. \tag{1}$$

We denote by

$$\begin{aligned} \mathscr{C} &= \{ C \in \mathbb{R}^{k \times k} \mid C \text{ symmetric positive semi-definite, } C_{i,i} = 1\ \forall i \}, \\ \mathscr{C}_{\leq t} &= \{ C \in \mathscr{C} \mid \textstyle\sum_{i,j} C_{i,j} \leq t^2 \} \subset \mathscr{C} \end{aligned}$$

the spaces of all possible configurations resp. those which give a length of at most t. The spaces $\mathscr{C}$ and $\mathscr{C}_{\leq t}$ are compact and convex. For fixed k, it is helpful from an algorithmic point of view to think of $\mathscr{C}$ as a finite set: for any $\varepsilon > 0$, we can cover $\mathscr{C}$ by finitely many ε-balls, so we can efficiently enumerate $\mathscr{C}$.

In the context of the approximate k-List problem with target length t, a k-tuple $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ is a solution iff $\operatorname{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \in \mathscr{C}_{\leq t}$. For that reason, we call a configuration in $\mathscr{C}_{\leq t}$ *good*. An obvious way to solve the approximate k-List problem is to enumerate over all good configurations and solve the following k-List configuration problem:

Definition 3 (Configuration problem). *On input k exponentially-sized lists $L_1, \ldots, L_k$ of vectors from S^n, a target configuration $C \in \mathscr{C}$ and some $\varepsilon > 0$, the task is to output all k-tuples $\boldsymbol{x}_1 \in L_1, \ldots, \boldsymbol{x}_k \in L_k$, such that $|\langle \boldsymbol{x}_i , \boldsymbol{x}_j \rangle - C_{ij}| \leq \varepsilon$ for all i, j. Such k-tuples are called solutions to the problem.*

Remark 1. Due to $\langle \boldsymbol{x}_i , \boldsymbol{x}_j \rangle$ taking real values, it does not make sense to ask for exact equality to C, but rather we introduce some $\varepsilon > 0$. We shorthand write $C \approx_\varepsilon C'$ for $|C_{i,j} - C'_{i,j}| \leq \varepsilon$. Formally, our analysis will show that for fixed $\varepsilon > 0$, we obtain running times and list sizes of the form $\widetilde{\mathcal{O}}_\varepsilon(2^{(c+f(\varepsilon))n})$ for some unspecified continuous f with $\lim_{\varepsilon \to 0} f(\varepsilon) = 0$. Letting $\varepsilon \to 0$ sufficiently slowly, we absorb $f(\varepsilon)$ into the $\widetilde{\mathcal{O}}(.)$-notation and omit it.

As opposed to the approximate k-List problem, being a solution to the k-List configuration problem is a locally checkable property[13]: it is a conjunction of conditions involving only *pairs* $\boldsymbol{x}_i, \boldsymbol{x}_j$. It is this and the following observation that we leverage to improve on the results of [4].

It turns out that the configurations attained by the solutions to the approximate k-List problem are concentrated around a single good configuration, which is the good configuration with the highest amount of symmetry. So in fact, we only need to solve the configuration problem for this particular good configuration. The following theorem describes the distribution of configurations:

Theorem 1. *Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathsf{S}^n$ be independent, uniformly distributed on the n-sphere, $n > k$. Then the configuration $C = C(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ follows a distribution $\mu_{\mathscr{C}}$ on $\mathscr{C}$ with density given by*

$$\mu_{\mathscr{C}} = W_{n,k} \cdot \det(C)^{\frac{1}{2}(n-k)} \mathrm{d}\mathscr{C} = \widetilde{\mathcal{O}}_k\Big(\det(C)^{\frac{n}{2}}\Big) \mathrm{d}\mathscr{C},$$

where $W_{n,k} = \pi^{-\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n+1-i}{2})} = \mathcal{O}_k\big(n^{\frac{k(k-1)}{4}}\big)$ is a normalization constant that only depends on n and k. Here, the reference measure $\mathrm{d}\mathscr{C}$ is given by $\mathrm{d}\mathscr{C} = \mathrm{d}C_{1,2} \cdots \mathrm{d}C_{(k-1),k}$ (i.e. the Lebesgue measure in a natural parametrization).

Proof. We derive this by an approximate normalization of the so-called Wishart distribution[22]. Observe that we can sample $C \leftarrow \mu_{\mathscr{C}}$ in the following way:
We sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^{n+1}$ iid from spherical $n+1$-dimensional Gaussians, such that the direction of each $\boldsymbol{x}_i$ is uniform over S^n. Note that the lengths of the $\boldsymbol{x}_i$ are not normalized to 1. Then we set $A_{i,j} := \langle \boldsymbol{x}_i , \boldsymbol{x}_j \rangle$. Finally, normalize to $C_{i,j} := \frac{A_{i,j}}{\sqrt{A_{i,i} A_{j,j}}}$.

The joint distribution of the $A_{i,j}$ is (by definition) given by the so-called Wishart distribution.[22] Its density for $n+1 > k-1$ is known to be

$$\rho_{\text{Wishart}} = \frac{e^{-\frac{1}{2}\operatorname{Tr} A} \cdot \det(A)^{\frac{n+1-k-1}{2}}}{2^{\frac{(n+1)k}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n+1-i}{2})} \mathrm{d}A \qquad (2)$$

where the reference density $\mathrm{d}A$ is given by $\mathrm{d}A = \prod_{i \le j} \mathrm{d}A_{i,j}$. We refer to [8] for a relatively simple computation of that density. Consider the change of variables on $\mathbb{R}^{k(k+1)/2}$ given by

$$\begin{aligned}&\Phi\big(A_{1,1}, A_{2,2}, \ldots, A_{k,k}, A_{1,2}, \ldots, A_{k-1,k}\big)\\ &\quad = \Big(A_{1,1}, A_{2,2}, \ldots, A_{k,k}, \tfrac{A_{1,2}}{\sqrt{A_{1,1}A_{2,2}}}, \ldots, \tfrac{A_{k,k-1}}{\sqrt{A_{k-1,k-1}A_{k,k}}}\Big),\end{aligned}$$

i.e. we map the $A_{i,j}$'s to $C_{i,j}$'s while keeping the $A_{i,i}$'s to make the transformation bijective almost everywhere. The Jacobian $D\Phi$ of Φ is a triangular matrix and its determinant is easily seen to be

$$\big|\det\big(D\Phi\big)\big| = \prod_i \frac{1}{\sqrt{A_{i,i}}^{k-1}}.$$

Further, note that $A = TCT$, where T is a diagonal matrix with diagonal $\sqrt{A_{1,1}}, \ldots, \sqrt{A_{k,k}}$. In particular, $\det(A) = \det(C) \cdot \prod_i A_{i,i}$. Consequently, we can transform the Wishart density into $\big(A_{1,1}, \ldots, A_{k,k}, C_{1,2}, \ldots, C_{k-1,k}\big)$-coordinates as

$$\rho_{\text{Wishart}} = \frac{e^{-\frac{1}{2}\sum_i A_{i,i}} \det(C)^{\frac{n-k}{2}} \prod_i A_{i,i}^{\frac{n-k}{2}}}{2^{\frac{(n+1)k}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n-i+1}{2})} \prod_i \sqrt{A_{i,i}}^{k-1} \prod_i \mathrm{d}A_{i,i} \prod_{i<j} \mathrm{d}C_{i,j}.$$

The desired $\mu_{\mathscr{C}}$ is obtained from ρ_{Wishart} by integrating out $\mathrm{d}A_{1,1}\mathrm{d}A_{2,2}\cdots\mathrm{d}A_{k,k}$. We can immediately see that $\mu_{\mathscr{C}}$ takes the form $\mu_{\mathscr{C}} = W_{n,k} \det(C)^{\frac{n-k}{2}} \mathrm{d}\mathscr{C}$ for some constants $W_{n,k}$. We compute $W_{n,k}$ as

$$\begin{aligned}W_{n,k} &= \int\limits_{A_{1,1}} \cdots \int\limits_{A_{k,k}} \frac{e^{-\frac{1}{2}\sum_i A_{i,i}} \prod_i A_{i,i}^{\frac{n-k}{2}}}{2^{\frac{(n+1)k}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n-i+1}{2})} \prod_i \sqrt{A_{i,i}}^{k-1} \prod_i \mathrm{d}A_{i,i}\\
&= \frac{1}{2^{\frac{(n+1)k}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n-i+1}{2})} \Big(\int_{A_{1,1}=0}^{+\infty} A_{1,1}^{\frac{n-1}{2}} e^{-\frac{1}{2}A_{1,1}} \,\mathrm{d}A_{1,1}\Big)^k\\
&= \frac{2^{\frac{(n+1)k}{2}}}{2^{\frac{(n+1)k}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n-i+1}{2})} \Big(\int_{A_{1,1}=0}^{+\infty} \big(\tfrac{A_{1,1}}{2}\big)^{\frac{n+1}{2}-1} e^{-\frac{1}{2}A_{1,1}} \,\tfrac{1}{2}\mathrm{d}A_{1,1}\Big)^k\\
&= \frac{1}{\pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n-i+1}{2})} \Big(\int_{x=0}^{+\infty} x^{\frac{n+1}{2}-1} e^{-x} \,\mathrm{d}x\Big)^k\\
&= \frac{\Gamma(\frac{n+1}{2})^k}{\pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n-i+1}{2})}.\end{aligned}$$

Finally, note that as a consequence of Stirling's formula, we have $\frac{\Gamma(n+z)}{\Gamma(n)} = \mathcal{O}_z(n^z)$ for any fixed z and $n \to \infty$. From this, we get

$$W_{n,k} = \frac{\Gamma(\frac{n+1}{2})^k}{\pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n-i+1}{2})} = \mathcal{O}_k\Big(n^{\sum_{i=0}^{k-1} \frac{i}{2}}\Big) = \mathcal{O}_k\Big(n^{\frac{k(k-1)}{4}}\Big).$$

The configurations C that we care about the most have the highest amount of symmetry. We call a configuration C balanced if $C_{i,j} = C_{i',j'}$ for all $i \neq j$, $i' \neq j'$. To compute the determinant $\det(C)$ for such balanced configurations, we have the following lemma:

Lemma 1.

$$\text{Let } C = \begin{pmatrix} 1 & a & a & \dots & a \\ a & 1 & a & \dots & a \\ a & a & 1 & \dots & a \\ \vdots & & & \ddots & \vdots \\ a & a & a & \dots & 1 \end{pmatrix} \in \mathbb{R}^{k\times k}.$$

Then $\det(C) = (1-a)^{k-1}(1+(k-1)a)$.

Proof. We have $C = (1-a)\cdot \mathbb{1}_k + a\cdot \mathbf{1}\cdot\mathbf{1}^{\mathrm{t}}$, where $\mathbf{1}\in\mathbb{R}^{k\times 1}$ is an all-ones vector. Sylvester's Determinant Theorem[2] gives

$$\begin{aligned}\det(C) &= (1-a)^k \det\bigl(\mathbb{1}_k + \tfrac{a}{1-a}\mathbf{1}\cdot\mathbf{1}^{\mathrm{t}}\bigr) = (1-a)^k \det\bigl(\mathbb{1}_1 + \tfrac{a}{1-a}\mathbf{1}^{\mathrm{t}}\cdot\mathbf{1}\bigr)\\ &= (1-a)^k(1+\tfrac{a}{1-a}k) = (1-a)^{k-1}(1+(k-1)a).\end{aligned}$$

For fixed k and C, the probability density $\widetilde{\mathcal{O}}\bigl(\det(C)^{\frac{n}{2}}\bigr)$ of $\mu_{\mathscr{C}}$ is exponential in n. Since $C\in\mathscr{C}$ can only vary in a compact space, taking integrals will asymptotically pick the maximum value: in particular, we have for the probability that a uniformly random k-tuple $\boldsymbol{x}_1,\dots,\boldsymbol{x}_k$ is good:

$$\int_{C\text{ good}} \mu_{\mathscr{C}} = \widetilde{\mathcal{O}}\Bigl(\max_{C\text{ good}} \det(C)^{\frac{n}{2}}\Bigr). \tag{3}$$

We now compute this maximum.

Theorem 2. *Let* $0 < t < \sqrt{k}$ *be some target length and consider the subset* $\mathscr{C}_{\leq t}\subset\mathscr{C}$ *of good configurations for target length at most* t*. Then* $\det(C)$ *attains its unique maximum over* $\mathscr{C}_{\leq t}$ *at the balanced configuration* $C_{\mathrm{Bal},t}$*, defined by* $C_{i,j} = \frac{t^2-k}{k^2-k}$ *for all* $i\neq j$ *with maximal value*

$$\det(C)_{\max} = \det(C_{\mathrm{Bal},t}) = \frac{t^2}{k}\Bigl(\frac{k^2-t^2}{k^2-k}\Bigr)^{k-1}.$$

In particular, for $t=1$*, this gives* $C_{i,j} = -\frac{1}{k}$ *and* $\det(C)_{\max} = \frac{(k+1)^{k-1}}{k^k}$*. Consequently, for any fixed* k *and any fixed* $\varepsilon>0$*, the probability that a randomly chosen solution to the approximate* k*-List problem is* ε*-close to* $C_{\mathrm{Bal},t}$ *converges exponentially fast to* 1 *as* $n\to\infty$*.*

Proof. It suffices to show that C is balanced at the maximum, i.e. that all $C_{i,j}$ with $i\neq j$ are equal. Then computing the actual values is straightforward from (1) and Lemma 1. Assume $k\geq 3$, as there is nothing to show otherwise.

For the proof, it is convenient to replace the conditions $C_{i,i}=1$ for all i by the (weaker) condition $\mathrm{Tr}(C)=k$. Let $\mathscr{C}'_{\leq t}$ denote the set of all symmetric, positive semi-definite $C\in\mathbb{R}^{k\times k}$ with $\mathrm{Tr}(C)=k$ and $\sum_{i,j} C_{i,j}\leq t^2$. We maximize $\det(C)$ over $\mathscr{C}'_{\leq t}$ and our proof will show that $C_{i,i}=1$ is satisfied at the maximum.

Let $C \in \mathscr{C}'_{\leq t}$. Since C is symmetric, positive semi-definite, there exists an orthonormal basis $\boldsymbol{v}_1, \dots, \boldsymbol{v}_k$ of eigenvectors with eigenvalues $0 \leq \lambda_1 \leq \dots \leq \lambda_k$.

Clearly, $\sum_i \lambda_i = \mathrm{Tr}(C) = k$ and our objective $\det(C)$ is given by $\det(C) = \prod_i \lambda_i$. We can write $\sum_{i,j} C_{i,j}$ as $\mathbf{1}^{\mathsf{t}} C \mathbf{1}$ for an all-ones vector $\mathbf{1}$. We will show that if $\det(C)$ is maximal, then $\mathbf{1}$ is an eigenvector of C. Since

$$t^2 \geq \mathbf{1}^{\mathsf{t}} C \mathbf{1} \geq \lambda_1 \|\mathbf{1}\|^2 = k\lambda_1, \tag{4}$$

for the smallest eigenvalue λ_1 of C, we have $\lambda_1 \leq \frac{t^2}{k} < 1$. For fixed λ_1, maximizing $\det(C) = \lambda_1 \cdot \prod_{i=2}^k \lambda_i$ under $\sum_{i=2}^k \lambda_i = k - \lambda_1$ gives (via the Arithmetic Mean-Geometric Mean Inequality)

$$\det(C) \leq \lambda_1 \Big(\frac{k - \lambda_1}{k-1}\Big)^{k-1}.$$

The derivative of the right-hand side wrt. λ_1 is $\frac{k(1-\lambda_1)}{k-1}\big(\frac{k-\lambda_1}{k-1}\big)^{k-2} > 0$, so we can bound it by plugging in the maximal $\lambda_1 = \frac{t^2}{k}$:

$$\det(C) \leq \lambda_1 \Big(\frac{k - \lambda_1}{k-1}\Big)^{k-1} \leq \frac{t^2}{k}\Big(\frac{k - \frac{t^2}{k}}{k-1}\Big)^{k-1} = \frac{t^2}{k}\Big(\frac{k^2 - t}{k^2 - k}\Big)^{k-1} \tag{5}$$

The inequalities (5) are satisfied with equality iff $\lambda_2 = \dots = \lambda_k$ and $\lambda_1 = \frac{t^2}{k}$. In this case, we can compute the value of λ_2 as $\lambda_2 = \frac{k^2 - t^2}{k(k-1)}$ from $\mathrm{Tr}(C) = k$. The condition $\lambda_1 = \frac{t^2}{k}$ means that (4) is satisfied with equality, which implies that $\mathbf{1}$ is an eigenvector with eigenvalue λ_1. So wlog. $\boldsymbol{v}_1 = \frac{1}{\sqrt{k}}\mathbf{1}$. Since the $\boldsymbol{v}_i$'s are orthonormal, we have $\mathbb{1}_k = \sum_i \boldsymbol{v}_i {\boldsymbol{v}_i}^{\mathsf{t}}$, where $\mathbb{1}_k$ is the $k \times k$ identity matrix. Since we can write C as $C = \sum_i \lambda_i \boldsymbol{v}_i {\boldsymbol{v}_i}^{\mathsf{t}}$, we obtain

$$C = \sum_i \lambda_i \boldsymbol{v}_i {\boldsymbol{v}_i}^{\mathsf{t}} = (\lambda_1 - \lambda_2)\boldsymbol{v}_1 {\boldsymbol{v}_1}^{\mathsf{t}} + \lambda_2 \sum_{i=1}^k \boldsymbol{v}_i {\boldsymbol{v}_i}^{\mathsf{t}} = \frac{\lambda_1 - \lambda_2}{k}\mathbf{1}\mathbf{1}^{\mathsf{t}} + \lambda_2 \cdot \mathbb{1}_k,$$

for $\det(C)$ maximal. From $C = \frac{\lambda_1 - \lambda_2}{k}\mathbf{1}\mathbf{1}^{\mathsf{t}} + \lambda_2 \cdot \mathbb{1}_k$, we see that all diagonal entries of C are equal to $\lambda_2 + \frac{\lambda_1 - \lambda_2}{k}$ and the off-diagonal entries are all equal to $\frac{\lambda_1 - \lambda_2}{k}$. So all $C_{i,i}$ are equal with $C_{i,i} = 1$, because $\mathrm{Tr}(C) = k$, and C is balanced.

For the case $t > \sqrt{k}$, and $\mathscr{C}_{\leq t}$ replaced by $\mathscr{C}_{\geq t}$, the statement can be proven analogously. Note that we need to consider the largest eigenvalue rather than the smallest in the proof. We remark that for $t = 1$, the condition $\langle \boldsymbol{x}_i \,, \boldsymbol{x}_j \rangle = C_{i,j} = -\frac{1}{k}$ for all $i \neq j$ is equivalent to saying that $\boldsymbol{x}_1, \dots, \boldsymbol{x}_k$ are k points of a regular $k+1$-simplex whose center is the origin. The missing $k+1^{\text{th}}$ point of the simplex is $-\sum_i \boldsymbol{x}_i$, i.e. the negative of the sum (see Fig. (1)).

A corollary of our concentration result is the following formula for the expected size of the output lists in the approximate k-List problem.

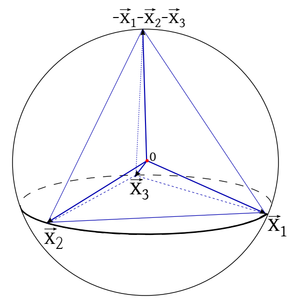


Fig. 1: A regular tetrahedron (3−simplex) represents a balanced configuration for $k = 3$.

Corollary 1. *Let k, t be fixed. Then the expected number of solutions to the approximate k-List problem with input lists of length $|L|$ is*

$$\mathbb{E}[\#solutions] = \widetilde{\mathcal{O}}\left(|L|^k \left(\frac{t^2}{k}\left(\frac{k^2-t^2}{k^2-k}\right)^{k-1}\right)^{\frac{n}{2}}\right). \tag{6}$$

Proof. By Thms. 2 and 1, the probability that any k-tuple is a solution is given by $\widetilde{\mathcal{O}}(\det(C_{\mathrm{Bal},t})^{\frac{n}{2}})$. The claim follows immediately.

In particular, this allows us to prove the following conjecture of [4]:

Theorem 3. *Let k be fixed and $t = 1$. If in the approximate k-List problem, the length $|L|$ of each input list is equal to the expected length of the output list, then $|L| = \widetilde{\mathcal{O}}\left(\left(\frac{k^{\frac{k}{k-1}}}{k+1}\right)^{\frac{n}{2}}\right)$.*

Proof. This follows from simple algebraic manipulation of (6).

Our concentration result shows that it is enough to solve the configuration problem for $C_{\mathrm{Bal},t}$.

Corollary 2. *Let k, t be fixed. Then the approximate k-List problem with target length t can be solved in essentially the same time as the k-List configuration problem with target configuration $C_{\mathrm{Bal},t}$ for any fixed $\varepsilon > 0$.*

Proof. On input $L_1, \ldots, L_k$, solve the k-List configuration problem with target configuration $C_{\mathrm{Bal},t}$. Restrict to those solutions whose sum has length at most t. By Thm. 2, this will find all but an exponentially small fraction of solutions to the approximate k-List problem. Since we only need to output a $1-o(1)$-fraction of the solutions, this solves the problem.

4 Algorithm

In this section we present our algorithm for the Configuration problem (Def. 3). On input it receives k lists $L_1, \ldots, L_k$, a target configuration C in the form of a

Gram matrix $C_{i,j} = \langle \boldsymbol{x}_i \, , \boldsymbol{x}_j \rangle \in \mathbb{R}^{k \times k}$ and a small $\varepsilon > 0$. The algorithm proceeds as follows: it picks an $\boldsymbol{x}_1 \in L_1$ and filters all the remaining lists with respect to the values $\langle \boldsymbol{x}_1 \, , \boldsymbol{x}_i \rangle$ for all $2 \leq i \leq k$. More precisely, $\boldsymbol{x}_i \in L_i$ 'survives' the filter if $|\langle \boldsymbol{x}_1 \, , \boldsymbol{x}_i \rangle - C_{1,i}| \leq \varepsilon$. We put such an $\boldsymbol{x}_i$ into $L_i^{(1)}$ (the superscript indicates how many filters were applied to the original list L_i). On this step, all the k-tuples of the form $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k) \in \{\boldsymbol{x}_1\} \times L_2^{(1)} \times \ldots \times L_k^{(1)}$ with a fixed first component $\boldsymbol{x}_1$ partially match the target configuration: all scalar products involving $\boldsymbol{x}_1$ are as desired. In addition, the lists $L_i^{(1)}$ become much shorter than the original ones.

Next, we choose an $\boldsymbol{x}_2 \in L_2^{(1)}$ and create smaller lists $L_i^{(2)}$ from $L_i^{(1)}$ by filtering out all the $\boldsymbol{x}_i \in L_i^{(1)}$ that do not satisfy $|\langle \boldsymbol{x}_2 \, , \boldsymbol{x}_i \rangle - C_{2,i}| \leq \varepsilon$ for all $3 \leq i \leq k$. A tuple of the form $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_k) \in \{\boldsymbol{x}_1\} \times \{\boldsymbol{x}_2\} \times L_3^{(2)} \times \ldots \times L_k^{(2)}$ satisfies the target configuration $C_{i,j}$ for $i = 1, 2$. We proceed with this list-filtering strategy until we have fixed all $\boldsymbol{x}_i$ for $1 \leq i \leq k$. We output all such k-tuples. Note that our algorithm becomes the trivial brute-force algorithm once we are down to 2 lists to be processed. As soon as we have fixed $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-2}$ and created $L_{k-1}^{(k-2)}, L_k^{(k-2)}$, our algorithm iterates over $L_{k-1}^{(k-2)}$ and checks the scalar product with every element from $L_k^{(k-2)}$.

Our algorithm is detailed in Alg. 1 and illustrated in Fig. (2a).

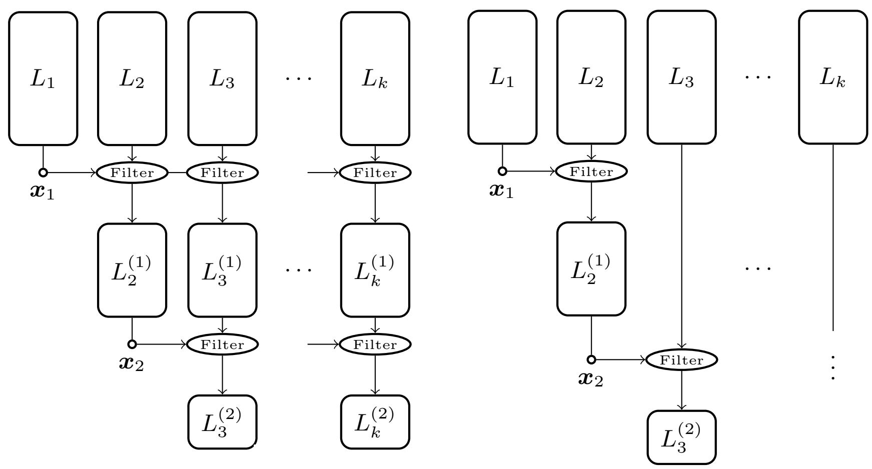
Algorithm 1 k-List for the Configuration Problem

Input: $L_1, \ldots, L_k$ – lists of vectors from S^n. $C_{i,j} = \langle \boldsymbol{x}_i \, , \boldsymbol{x}_j \rangle \in \mathbb{R}^{k \times k}$ – Gram matrix. $\varepsilon > 0$.

Output: L_{out} – list of k-tuples $\boldsymbol{x}_1 \in L_1, \ldots, \boldsymbol{x}_k \in L_k$, s.t. $|\langle \boldsymbol{x}_i \, , \boldsymbol{x}_j \rangle - C_{ij}| \leq \varepsilon$, for all i, j.

$L_{\text{out}} \leftarrow \{\}$
for all $\boldsymbol{x}_1 \in L_1$ **do**
  for all $j = 2 \ldots k$ **do**
    $L_j^{(1)} \leftarrow$ FILTER$(\boldsymbol{x}_1, L_j, C_{1,j}, \varepsilon)$
  for all $\boldsymbol{x}_2 \in L_2^{(1)}$ **do**
    for all $j = 3 \ldots k$ **do**
      $L_j^{(2)} \leftarrow$ FILTER$(\boldsymbol{x}_2, L_j^{(1)}, C_{2,j}, \varepsilon)$
    $\ddots$
      for all $\boldsymbol{x}_k \in L_k^{(k-1)}$ **do**
        $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\boldsymbol{x}_1, \ldots \boldsymbol{x}_k)\}$
return L_{out}

function FILTER$(\boldsymbol{x}, L, c, \varepsilon)$
  $L' \leftarrow \{\}$
  for all $\boldsymbol{x}' \in L$ **do**
    if $|\langle \boldsymbol{x} \, , \boldsymbol{x}' \rangle - c| \leq \varepsilon$ **then**
      $L' \leftarrow L' \cup \{\boldsymbol{x}'\}$
  return L'



(a) Pictorial representation of Alg. 1. At level i, a filter receives as input $\boldsymbol{x}_i$ and a vector $\boldsymbol{x}_j$ from $L_j^{(i-1)}$ (for the input lists, $L = L^{(0)}$). $\boldsymbol{x}_j$ passes through the filter if $|\langle \boldsymbol{x}_i , \boldsymbol{x}_j \rangle - C_{i,j}| \leq \varepsilon$, in which case it is added to $L_j^{(i)}$. The configuration C is a global parameter.

(b) The k-List algorithm given in [4]. The main difference is that a filter receives as inputs $\boldsymbol{x}_i$ and a vector $\boldsymbol{x}_j \in L_j$, as opposed to $\boldsymbol{x}_j \in L_j^{(i-1)}$. Technically, in [4], $\boldsymbol{x}_i$ survives the filter if $|\langle \boldsymbol{x}_i , \boldsymbol{x}_1 + \ldots + \boldsymbol{x}_{i-1} \rangle| \geq c_i$ for some predefined c_i. Due to our concentration results, this description is equivalent to the one given in [4] in the sense that the returned solutions are (up to a subexponential fraction) the same.

Fig. 2: k-List Algorithms for the Configuration Problem. Left: Our Alg. 1. Right: k-tuple sieve algorithm of [4]

5 Analysis

In this section we analyze the complexity of Alg. 1 for the Configuration problem. First, we should mention that the memory complexity is completely determined by the input list-sizes $|L_i|$ (remember that we restrict to constant k) and it does not change the asymptotics when we apply k filters. In practice, all intermediate lists $L_i^{(j)}$ can be implemented by storing pointers to the elements of the original lists.

In the following, we compute the expected sizes of filtered lists $L_i^{(j)}$ and establish the expected running time of Alg. 1. Since our algorithm has an exponential running time of 2^{cn} for some $c = \Theta(1)$, we are interested in determining c (which depends on k) and we ignore polynomial factors, e.g. we do not take into account time spent for computing inner products.

Theorem 4. *Let k be fixed. Alg. 1 given as input k lists $L_1, \ldots, L_k \subset \mathsf{S}^n$ of the same size $|L|$, a target balanced configuration $C_{\mathrm{Bal},t} \in \mathbb{R}^{k\times k}$, a target length $0 < t < \sqrt{k}$, and $\varepsilon > 0$, outputs the list L_{out} of solutions to the Configuration problem. The expected running time of Alg. 1 is*

$$T = \widetilde{\mathcal{O}}\Big(|L| \cdot \max_{1\le i\le k-1} |L|^i \cdot \frac{(k^2-t^2)^i}{(k^2-k)^{i+1}} \cdot \Big(\frac{(k^2-k+(i-1)(t^2-k))^2}{k^2-k+(i-2)(t^2-k)}\Big)^{\frac{n}{2}}\Big). \quad (7)$$

In particular, for $t = 1$ and $|L_{\mathrm{out}}| = |L|$ it holds that

$$T = \widetilde{\mathcal{O}}\Big(\Big(\frac{k^{\frac{1}{k-1}}}{k+1} \cdot \max_{1\le i\le k-1} k^{\frac{i}{k-1}} \cdot \frac{(k-i+1)^2}{k-i+2}\Big)^{\frac{n}{2}}\Big). \quad (8)$$

Remark 2. In the proof below we also show that the expected running time of the k-List algorithm presented in [4] is (see also Fig. (3) for a comparison) for $t = 1, |L_{\mathrm{out}}| = |L|$

$$T_{\mathrm{BLS}} = \widetilde{\mathcal{O}}\Big(\Big(\frac{k^{\frac{k}{k-1}}}{(k+1)^2} \cdot \max_{1\le i\le k-1} \big(k^{\frac{i}{k-1}} \cdot (k-i+1)\big)\Big)^{\frac{n}{2}}\Big). \quad (9)$$

Corollary 3. *For $k = 3$, $t = 1$, and $|L| = |L_{\mathrm{out}}|$ (the most interesting setting for SVP), Alg. 1 has running time*

$$T = 2^{0.3962n+o(n)}, \quad (10)$$

requiring $|L| = 2^{0.1887n+o(n)}$ memory.

Proof (Proof of Thm. 4). The correctness of the algorithm is straightforward: let us associate the lists $L^{(i)}$ with a level i where i indicates the number of filtering steps applied to L (we identify the input lists with the 0^{th} level: $L_i = L_i^{(0)}$). So for executing the filtering for the i^{th} time, we choose an $\boldsymbol{x}_i \in L_i^{(i-1)}$ that satisfies the condition $|\langle \boldsymbol{x}_i , \boldsymbol{x}_{i-1}\rangle - C_{i,i-1}| \le \varepsilon$ (for a fixed $\boldsymbol{x}_{i-1}$) and append to

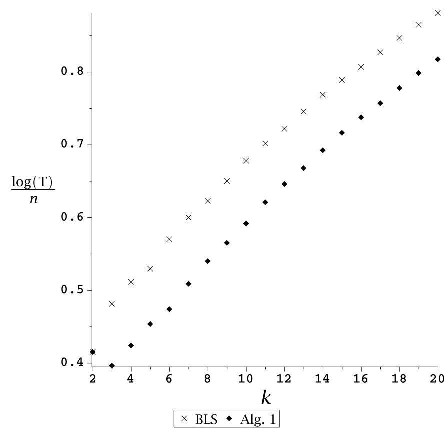


Fig. 3: Running exponents scaled by $1/n$ for the target length $t = 1$. For $k = 2$, both algorithms are the Nguyen-Vidick sieve [20] with $\log(T)/n = 0.415$ (naive brute-force over two lists). For $k = 3$, Algorithm 1 achieves $\log(T)/n = 0.3962$.

a previously obtained $(i-1)$-tuple $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1})$. Thus on the last level, we put into L_{out} a k-tuple $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ that is a solution to the Configuration problem.

Let us first estimate the size of the list $L_i^{(i-1)}$ output by the filtering process applied to the list $L_i^{(i-2)}$ for $i > 1$ (i.e. the left-most lists on Fig. (2a)). Recall that all elements $\boldsymbol{x}_i \in L_i^{(i-1)}$ satisfy $|\langle \boldsymbol{x}_i\,, \boldsymbol{x}_j \rangle - C_{i,j}| \leq \varepsilon,\ 1 \leq j \leq i-1$. Then the *total* number of i-tuples $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_i) \in L_1 \times L_2^{(1)} \times \ldots \times L_i^{(i-1)}$ considered by the algorithm is determined by the probability that in a random i-tuple, all pairs $(\boldsymbol{x}_j, \boldsymbol{x}_{j'}), 1 \leq j, j' \leq i$ satisfy the inner product constraints given by $C_{j,j'}$. This probability is given by Thm. 1 and since the input lists are of the same size $|L|$, we have[1]

$$|L_1| \cdot |L_2^{(1)}| \cdot \ldots |L_i^{(i-1)}| = |L|^i \cdot \det(C[1 \ldots i])^{\frac{n}{2}}, \tag{11}$$

where $\det(C[1 \ldots i])$ denotes the i-th principal minor of C. Using (11) for two consecutive values of i and dividing, we obtain

$$|L_{i+1}^{(i)}| = |L| \cdot \Big(\frac{\det(C[1 \ldots i+1])}{\det(C[1 \ldots i])} \Big)^{\frac{n}{2}}. \tag{12}$$

Note that these expected list sizes can be smaller than 1. This should be thought of as the inverse probability that the list is not empty. Since we target a balanced configuration $C_{\text{Bal},t}$, the entries of the input Gram matrix are specified by Thm. 2 and, hence, we compute the determinants in the above quotient by applying Lemma 1 for $a = \frac{t^k - k}{k^2 - k}$. Again, from the shape of the Gram matrix $C_{\text{Bal},t}$ and the equal-sized input lists, it follows that the filtered list on each level are of the same size: $|L_{i+1}^{(i)}| = |L_{i+2}^{(i)}| = \ldots = |L_k^{(i)}|$. Therefore, for all filtering levels $0 \leq j \leq k-1$ and for all $j+1 \leq i \leq k$,

$$\big| L_i^{(j)} \big| = |L| \cdot \Big(\frac{k^2 - t^2}{k^2 - k} \cdot \frac{k^2 - k + j(t^2 - k)}{k^2 - k + (j-1)(t^2 - k)} \Big)^{\frac{n}{2}}. \tag{13}$$

[1] Throughout this proof, the equations that involve list-sizes $|L|$ and running time T are assumed to have $\widetilde{\mathcal{O}}(\cdot)$ on the right-hand side. We omit it for clarity.

Now let us discuss the running time. Clearly, the running time of Alg. 1 is (up to subexponential factors in n)

$$T = |L_1^{(0)}| \cdot (|L_2^{(0)}| + |L_2^{(1)}| \cdot (|L_3^{(1)}| + |L_3^{(2)}| \cdot (\ldots \cdot (|L_k^{(k-2)}| + |L_k^{(k-1)}|))).$$

Multiplying out and observing that $|L_k^{(k-2)}| > |L_k^{(k-1)}|$, so we may ignore the very last term, we deduce that the total running time is (up to subexponential factors) given by

$$T = |L| \cdot \max_{1 \le i \le k-1} |L^{(i-1)}| \cdot \prod_{j=1}^{i-1} |L^{(j)}|, \tag{14}$$

where $|L^{(j)}|$ is the size of any filtered list on level j (so we omit the subscripts). Consider the value $i_{\max}$ of i where the maximum is attained in the above formula. The meaning of $i_{\max}$ is that the total cost over all loops to create the lists $L_j^{(i_{\max})}$ is dominating the running time. At this level, the lists $L_j^{(i_{\max})}$ become small enough such that iterating over them (i.e. creation of $L_j^{(i_{\max}+1)}$) does not contribute asymptotically. Plugging in Eqns. (11) and (12) into (14), we obtain

$$T = |L| \cdot \max_{1 \le i \le k-1} |L|^i \Big(\frac{(\det C[1 \ldots i])^2}{\det C[1 \ldots (i-1)]} \Big)^{\frac{n}{2}}. \tag{15}$$

Using Lemma 1, we obtain the desired expression for the running time.

For the case $t = 1$ and $|L_{\text{out}}| = |L|$, the result of Thm. 3 on the size of the input lists $|L|$ yields a compact formula for the filtered lists:

$$\big|L_i^{(j)}\big| = \Big(k^{\frac{1}{k-1}} \cdot \frac{k-j}{k-j+1} \Big)^{\frac{n}{2}}. \tag{16}$$

Plugging this into either (14) or (15), the running time stated in (8) easily follows.

It remains to show the complexity of the BLS algorithm [4], claimed in Rmk. 2. We do not give a complete description of the algorithm but illustrate it in Fig. (2b). We change the presentation of the algorithm to our configuration setting: in the original description, a vector $\boldsymbol{x}_i$ survives the filter if it satisfies $|\langle \boldsymbol{x}_i \,, \boldsymbol{x}_1 + \ldots + \boldsymbol{x}_{i-1} \rangle| \ge c_i$ for a predefined c_i (a sequence $(c_1, \ldots, c_{k-1}) \in \mathbb{R}^{k-1}$ is given as input to the BLS algorithm). Our concentration result (Thm. 1) also applies here and the condition $|\langle \boldsymbol{x}_i \,, \boldsymbol{x}_1 + \ldots + \boldsymbol{x}_{i-1} \rangle| \ge c_i$ is equivalent to a pairwise constraint on the $\langle \boldsymbol{x}_i \,, \boldsymbol{x}_j \rangle$ up to losing an exponentially small fraction of solutions. The optimal sequence of c_i's corresponds to the balanced configuration $C_{\text{Bal},t}$ derived in Thm. 2. Indeed, Table 1 in [4] corresponds exactly to $C_{\text{Bal},t}$ for $t = 1$. So we may rephrase their filtering where instead of shrinking the list L_i by taking inner products with the sum $\boldsymbol{x}_1 + \ldots + \boldsymbol{x}_{i-1}$, we filter L_i gradually by considering $\langle \boldsymbol{x}_i \,, \boldsymbol{x}_j \rangle$ for $1 \le j \le i-1$.

It follows that the filtered lists $L^{(i)}$ on level i are of the same size (in leading order) for both our and BLS algorithms. In particular, Eq. (12) holds for the expected list-sizes of the BLS algorithm. The crucial difference lies in the construction of these lists. To construct the list $L_i^{(i-1)}$ in BLS, the filtering procedure is applied not to $L_i^{(i-2)}$, but to a (larger) input-list L_i. Hence, the running time is (cf. (14)), ignoring subexponential factors

$$\begin{aligned} T_{\text{BLS}} &= |L_1| \cdot (|L_2| + |L_2^{(1)}| \cdot (|L_3| + |L_3^{(2)}| \cdot (\ldots \cdot (|L_k| + |L_k^{(k-1)}|))) \\ &= |L|^2 \cdot \max_{1\le i\le k-1} \cdot \prod_{j=1}^{i-1} |L^{(j)}|. \end{aligned}$$

The result follows after substituting (16) into the above product.

6 Configuration Extension

For $k = 2$, the asymptotically best algorithm with running time $T = \left(\frac{3}{2}\right)^{\frac{n}{2}}$ for $t = 1$ is due to [5], using techniques from Locally Sensitive Hashing. We generalize this to what we call Configuration Extension. To explain the LSH technique, consider the (equivalent) approximate 2-List problem with $t = 1$, where we want to bound the norm of the difference $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 \le 1$ rather than the sum, i.e. we want to find points that are close. The basic idea is to choose a family of hash functions $\mathscr{H}$, such that for $h \in \mathscr{H}$, the probability that $h(\boldsymbol{x}_1) = h(\boldsymbol{x}_2)$ is large if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are close, and small if they are far apart. Using such an $h \in \mathscr{H}$, we can bucket our lists according to h and then only look for pairs $\boldsymbol{x}_1, \boldsymbol{x}_2$ that collide under h. Repeat with several $h \in \mathscr{H}$ as appropriate to find all/most solutions. We may view such an $h \in \mathscr{H}$ as a collection of preimages $D_{h,z} = h^{-1}(z)$ and the algorithm first determines which elements $\boldsymbol{x}_1, \boldsymbol{x}_2$ are in some given $D_{h,z}$ (filtering the list using $D_{h,z}$) and then searches for solutions only among those. Note that, conceptually, we only really need the $D_{h,z}$ and not the functions h. Indeed, there is actually no need for the $D_{h,z}$ to be a partition of S^n for given h, and h need not even exist. Rather, we may have an arbitrary collection of sets $D^{(r)}$, with r belonging to some index set. The existence of functions h would help in efficiency when filtering. However, [5] (and also [18], stated for the ℓ_1-norm) give a technique to efficiently construct and apply filters $D^{(r)}$ without such an h in an amortized way.

The natural choice for $D^{(r)}$ is to choose all points with distance at most d for some $d > 0$ from some reference point $\boldsymbol{v}^{(r)}$ (that is typically not from any L_i). This way, a random pair $\boldsymbol{x}_1, \boldsymbol{x}_2 \in D^{(r)}$ has a higher chance to be close to each other than uniformly random points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathsf{S}^n$. Notationally, let us call (a description of) $D^{(r)}$ together with the filtered lists an *instance*, where $1 \le r \le R$ and R is the number of instances.

In our situation, we look for small sums rather than small differences. The above translates to asking that $\boldsymbol{x}_1$ is close to $\boldsymbol{v}^{(r)}$ and that $\boldsymbol{x}_2$ is far apart from $\boldsymbol{v}^{(r)}$ (or, equivalently, that $\boldsymbol{x}_2$ is close to $-\boldsymbol{v}^{(r)}$). In general, one may (for $k > 2$) consider not just a single $\boldsymbol{v}^{(r)}$ but rather several *related* $\boldsymbol{v}_1^{(r)}, \ldots, \boldsymbol{v}_m^{(r)}$. So an instance consists of m points $\boldsymbol{v}_1^{(r)}, \ldots, \boldsymbol{v}_m^{(r)}$ and shrunk lists $L_i'^{(r)}$ where $L_i'^{(r)} \subset L_i$ is obtained by taking those $x_i \in L_i$ that have some prescribed distances $d_{i,j}$

to $\boldsymbol{v}_j^{(r)}$. Note that the $d_{i,j}$ may depend on i and so need not treat the lists symmetrically. As a consequence, it does no longer make sense to think of this technique in terms of hash collisions in our setting.

We organize all the distances between $\boldsymbol{v}$'s and $\boldsymbol{x}$'s that occur into a single matrix C (i.e. a configuration) that governs the distances between $\boldsymbol{v}$'s and $\boldsymbol{x}$'s: the $\langle \boldsymbol{v}_j , \boldsymbol{v}_{j'} \rangle$-entries of C describe the relation between the $\boldsymbol{v}$'s and the $\langle \boldsymbol{x}_i , \boldsymbol{v}_j \rangle$-entries of C describe the $d_{i,j}$. The $\langle \boldsymbol{x}_i , \boldsymbol{x}_{i'} \rangle$-entries come from the approximate k-List problem we want to solve. While not relevant for constructing actual $\boldsymbol{v}_j^{(r)}$'s and $L_i'^{(r)}$'s, the $\langle \boldsymbol{x}_i , \boldsymbol{x}_{i'} \rangle$-entries are needed to choose the number R of instances.

For our applications to sieving, the elements from the input list L_i may possibly be not uniform from all of S^n due to previous processing of the lists. Rather, the elements $\boldsymbol{x}_i$ from L_i have some prescribed distance $d_{i,j}$ to (known) $\boldsymbol{v}_j$'s: e.g. in Alg. 1, we fix $\boldsymbol{x}_1 \in L_1$ that we use to filter the remaining $k-1$ lists; we model this by taking $\boldsymbol{x}_1$ as one of the $\boldsymbol{v}_j$'s (and reducing k by 1). Another possibility is that we use configuration extension on lists that are the output of a previous application of configuration extension.

In general, we consider "old" points $\boldsymbol{v}_j$ and wish to create "new" points $\boldsymbol{v}_\ell$, so we have actually three different types of rows/columns in C, corresponding to the list elements, old and new points.

Definition 4 (Configuration Extension). *Consider a configuration matrix C. We consider C as being indexed by disjoint sets $I_{\text{lists}}, I_{\text{old}}, I_{\text{new}}$. Here, $|I_{\text{lists}}| = k$ corresponds to the input lists, $|I_{\text{old}}| = m_{\text{old}}$ corresponds to the "old" points, $|I_{\text{new}}| = m_{\text{new}}$ corresponds to the "new" points. We denote appropriate square submatrices by $C[I_{\text{lists}}]$ etc. By configuration extension, we mean an algorithm* $\mathsf{ConfExt}$ *that takes as input k exponentially large lists $L_i \subset \mathsf{S}^n$ for $i \in I_{\text{lists}}$, m_{old} "old" points $\boldsymbol{v}_j \in \mathsf{S}^n$, $j \in I_{\text{old}}$ and the matrix C. Assume that each input list separately satisfies the given configuration constraints wrt. the old points: $\mathrm{Conf}(\boldsymbol{x}_i, (\boldsymbol{v}_j)_{j \in I_{\text{old}}}) \approx C[i, I_{\text{old}}]$ for $i \in I_{\text{lists}}$, $\boldsymbol{x}_i \in L_i$.*

It outputs R instances, where each instance consists of m_{new} points $\boldsymbol{v}_\ell$, $\ell \in I_{\text{new}}$ and shrunk lists $L_i' \subset L_i$, where $\mathrm{Conf}((\boldsymbol{v}_j)_{j \in I_{\text{old}}}, (\boldsymbol{v}_\ell)_{\ell \in I_{\text{new}}}) \approx C[I_{\text{old}}, I_{\text{new}}]$ and each $\boldsymbol{x}_i' \in L_i'$ satisfies

$$\mathrm{Conf}(\boldsymbol{x}_i', (\boldsymbol{v}_j)_{j \in I_{\text{old}}}, (\boldsymbol{v}_\ell)_{\ell \in I_{\text{old}}}) \approx C[i, I_{\text{old}}, I_{\text{new}}].$$

The instances are output one-by-one in a streaming fashion. This is important, since the total size of the output usually exceeds the amount of available memory.

The naive way to implement configuration extension is as follows: independently for each instance, sample uniform $\boldsymbol{v}_\ell$'s conditioned on the given constraints and then make a single pass over each input list L_i to construct L_i'. This would require $\widetilde{\mathcal{O}}(\max_i |L_i| \cdot R)$ time. However, using the block coding / stripe techniques of [5,18], one can do much better. The central observation is that if we subdivide the coordinates into blocks, then a configuration constraint on all coordinates is (up to losing a subexponential fraction of solutions) equivalent to independent configuration constraints on each block. The basic idea is then to construct the $\boldsymbol{v}_\ell$'s in a block-wise fashion such that an exponential number of instances have the same $\boldsymbol{v}_\ell$'s on a block of coordinates. We can then amortize the construction of the L_i''s among such instances, since we can first construct some intermediate $L_i'' \subset L_i$ that is compatible with the $\boldsymbol{v}_\ell$'s on the shared block of coordinates. To actually construct $L_i' \subset L_i''$, we only need to pass over L_i'' rather than L_i. Of course, this foregos independence of the $\boldsymbol{v}_\ell$'s across different instances, but one can show that they are still independent enough to ensure that we will find most solutions if the number of instances is large enough.

Adapting these techniques of [5,18] to our framework is straightforward, but extremely technical. We work out the details in Appendix A.

A rough summary of the properties of our Algorithm ConfExt from Alg. 6 in App. A is given by the following:

Theorem 5. *Use notation as in Def. 4. Assume that $C, k, m_{\mathrm{old}}, m_{\mathrm{new}}$ do not depend on n. Then our algorithm* ConfExt*, given as input $C, k, m_{\mathrm{old}}, m_{\mathrm{new}}$, old points $\boldsymbol{v}_j$ and exponentially large lists $L_1, \dots, L_k$ of points from S^n, outputs*

$$R = \widetilde{\mathcal{O}}\left(\frac{\det(C[I_{\mathrm{old}}, I_{\mathrm{new}}]) \cdot \det(C[I_{\mathrm{lists}}, I_{\mathrm{old}}])}{\det(C[I_{\mathrm{lists}}, I_{\mathrm{old}}, I_{\mathrm{new}}]) \cdot \det(C[I_{\mathrm{old}}])}\right)^{\frac{n}{2}} \tag{17}$$

instances, where each output instance consists of m_{new} points $\boldsymbol{v}_\ell$ and sublists $L_i' \subset L_i$. In each such output instance, the new points $(\boldsymbol{v}_\ell)_{\ell \in I_{\mathrm{new}}}$ are chosen uniformly conditioned on the constraints (but not independent across instances). Consider solution k-tuples, i.e. $\boldsymbol{x}_i \in L_i$ with $\mathrm{Conf}((\boldsymbol{x}_i)_{i \in I_{\mathrm{lists}}}) \approx C[I_{\mathrm{lists}}]$. With overwhelming probability, for every solution k-tuple $(\boldsymbol{x}_i)_{i \in I_{\mathrm{lists}}}$, there exists at least one instance such that all $\boldsymbol{x}_i \in L_i'$ for this instance, so we retain all solutions. Assume further that the elements from the input lists L_i, $i \in I_{\mathrm{lists}}$ are iid uniformly distributed conditioned on the configuration $\mathrm{Conf}(\boldsymbol{x}_i, (\boldsymbol{v}_j)_{j \in I_{\mathrm{old}}})$ for $\boldsymbol{x}_i \in L_i$, which is assumed to be compatible with C. Then the expected size of the output lists per instance is given by

$$\mathbb{E}[|L_i'|] = |L_i| \cdot \widetilde{\mathcal{O}}\left(\left(\frac{\det(C[i, I_{\mathrm{old}}, I_{\mathrm{new}}]) \cdot \det(C[I_{\mathrm{old}}])}{\det(C[I_{\mathrm{old}}, I_{\mathrm{new}}]) \cdot \det(C[i, I_{\mathrm{old}}])}\right)^{n/2}\right).$$

Assume that all these expected output list sizes are exponentially increasing in n (rather than decreasing). Then the running time of the algorithm is given by $\widetilde{\mathcal{O}}(R \cdot \max_i \mathbb{E}[|L_i'|])$ (essentially the size of the output) and the memory complexity is given by $\widetilde{\mathcal{O}}(\max_i |L_i|)$ (essentially the size of the input).

Proof. See App. A. Note for notational consistency that our more formal definition there also takes $\varepsilon > 0$ as input. This parameter $\varepsilon \to 0$ specifies the approximation quality in the $\approx$-statements used in Def. 4 above.

7 Improved k-List Algorithm with Configuration Extension

Now we explain how to use the Configuration Extension Algorithm 6 within the k-List algorithm Alg. 1 to speed-up the search for configurations. In fact, there is a whole family of algorithms obtained by combining Filter from Alg. 1 and the configuration extension algorithm ConfExt from Alg. 6.

Recall that Alg. 1 takes as inputs k lists $L_1, \dots, L_k$ of equal size and processes the lists in several levels (cf. Fig. 2a). The lists $L_j^{(i)}$ for $j \geq i$ at the i^{th} level (where the input lists correspond to the 0^{th} level) are obtained by brute-forcing over $\boldsymbol{x}_i \in L_i^{(i-1)}$ and running Filter on $L_j^{(i-1)}$ and $\boldsymbol{x}_i$.

We can use ConfExt in the following way: before using Filter on $L_j^{(i-1)}$, we run ConfExt to create R instances with smaller sublists $L_j'^{(i-1)} \subset L_j^{(i-1)}$. We then apply Filter to each of these $L_j'^{(i-1)}$ rather than to $L_j^{(i-1)}$. The advantage is that for a given instance, the $L_j'^{(i-1)}$ are dependent (over the choice of j), so we expect a higher chance to find solutions.

In principle, one can use ConfExt on any level, i.e. we alternate between using ConfExt and Filter. Note that the $\boldsymbol{x}_i$'s that we brute-force over in order to apply Filter become "old" $\boldsymbol{v}_j$'s in the context of the following applications of ConfExt.

It turns out that among the variety of potential combinations of Filter and ConfExt, some are more promising than others. From the analysis of Alg. 1, we know that the running time is dominated by the cost of filtering (appropriately multiplied by the number of times we need to filter) to create lists at some level $i_{\max}$. The value of $i_{\max}$ can be deduced from Eq. (14), where the individual contribution $|L| \cdot |L^{(i-1)}| \cdot \prod_{j=1}^{i-1} |L^{(j)}|$ in that formula exactly corresponds to the total cost of creating all lists at the i-th level.

It makes sense to use ConfExt to reduce the cost of filtering at this critical level. This means that we use ConfExt on the lists $L_j^{(i_{\max}-1)}$, $j \geq i_{\max} - 1$. Let us choose $m_{\text{new}} = 1$ new point $\boldsymbol{v}_\ell$. The lists $L_j^{(i_{\max}-1)}$ are already reduced by enforcing configuration constraints with $\boldsymbol{x}_1 \in L_1, \dots, \boldsymbol{x}_{i_{\max}-1} \in L_{i_{\max}-1}$ from previous applications of Filter. This means that the $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{i_{\max}-1}$ take the role of "old" $\boldsymbol{v}_j$'s in ConfExt. The configuration $C^{ext} \in \mathbb{R}^{(k+1)\times(k+1)}$ for ConfExt is obtained as follows: The $C^{ext}[I_{\text{lists}}, I_{\text{old}}]$-part is given by the target configuration. The rest (which means the last row/column corresponding to the single "new" point) can be chosen freely and is subject to optimization. Note that the optimization problem does not depend on n.

This approach is taken in Alg. 2. Note that for levels below $i_{\max}$, it does not matter whether we continue to use our Filter approach or just brute-force: if $i_{\max} = k$, there are no levels below. If $i_{\max} < k$, the lists are small from this level downward and brute-force becomes cheap enough not to affect the asymptotics.

Let us focus on the case where the input list sizes are the same as the output list sizes, which is the relevant case for applications to Shortest Vector sieving. It turns out (numerically) that in this case, the approach taken by Alg. 2 is optimal for most values of k. The reason is as follows: Let T be the contribution to the running time of Alg. 1 from level $i_{\max}$, which is asymptotically the same as the total running time. The second-largest contribution, denoted T' comes from level $i_{\max} - 1$. The improvement in running time from using ConfExt to reduce T decreases with k and is typically not enough to push it below T'. Consequently, using ConfExt between other levels will not help. We also observed that choosing $m_{\text{new}} = 1$ was usually optimal for k up to 10. Exceptions to these observations occur when T and T' are very close (this happens, e.g. for $k = 6$) or when k is small and the benefit from using ConfExt is large (i.e. $k = 3$).

Since the case $k = 3$ is particularly interesting for the Shortest Vector sieving (see Sect. 7.2), we present the 3-List algorithm separately in Sect. 7.1.

Algorithm 2 k-List with Configuration Extension

Input: $L_1, \ldots, L_k$ – input lists. $C \in \mathbb{R}^{k \times k}$ – target configuration. $\varepsilon > 0$ – measure of closeness.
Output: L_{out} – list of k-tuples $\boldsymbol{x}_1 \in L_1, \ldots, \boldsymbol{x}_k \in L_k$, s.t. $|\langle \boldsymbol{x}_i , \boldsymbol{x}_j \rangle - C_{ij}| \leq \varepsilon$, for all i, j.

$i_{\max}, C^{ext} = \textsc{Preprocess}(k, C_{i,j} \in \mathbb{R}^{k \times k})$
$L_{\text{out}} \leftarrow \{\}$
for all $\boldsymbol{x}_1 \in L_1$ **do**
  $L_j^{(1)} \leftarrow \textsc{Filter}(\boldsymbol{x}_1, L_j, C_{1,j}, \varepsilon)$ ▷ $j = 2, \ldots, k$
  $\ddots$
    for all $\boldsymbol{x}_{i_{\max}-1} \in L_{i_{\max}-1}^{(i_{\max}-2)}$ **do**
      $L_j^{(i_{\max}-1)} \leftarrow \textsc{Filter}(\boldsymbol{x}_{i_{\max}-1}, L_j^{i_{\max}-2}, C_{i_{\max}-1,j}, \varepsilon)$ ▷ $j = i_{\max}, \ldots, k$
      $I_{\text{old}} \leftarrow \{1, \ldots, i_{\max} - 1\}$, $I_{\text{lists}} \leftarrow \{i_{\max}, \ldots, k\}$, $I_{\text{new}} \leftarrow \{k + 1\}$.
      $m_{\text{old}} \leftarrow i_{\max} - 1, k' \leftarrow k + 1 - i_{\max}, m_{\text{new}} \leftarrow 1$.
      $\boldsymbol{v}_j \leftarrow \boldsymbol{x}_j$ for $j \in I_{\text{old}}$.
      Call $\mathsf{ConfExt}(n, k', m_{\text{old}}, m_{\text{new}}, C^{ext}, L_{i_{\max}}^{(i_{\max}-1)}, \ldots, L_k^{(i_{\max}-1)}, (\boldsymbol{v}_j)_{j \in I_{\text{old}}}, \varepsilon)$
      for all output instances $\boldsymbol{w}$, $L_{i_{\max}}'^{(i_{\max}-1)}, \ldots, L_k'^{(i_{\max}-1)}$ **do** ▷ Output is streamed
        for all $\boldsymbol{x}_{i_{\max}} \in L_j'^{(i_{\max}-1)}$ **do**
          $L_j^{(i_{\max})} \leftarrow \textsc{Filter}(\boldsymbol{x}_{i_{\max}}, L_j'^{(i_{\max}-1)}, C_{i_{\max},j}, \varepsilon)$ ▷ $j = i_{\max} + 1 \ldots k$
        Brute-force over $L_j^{(i_{\max})}$ to obtain $\boldsymbol{x}_{i_{\max}+1}, \ldots, \boldsymbol{x}_k$ compatible with C
        $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)\}$
return L_{out}

procedure $\textsc{Preprocess}(k,\ C \in \mathbb{R}^{k \times k})$
  Determine $i_{\max}$ using Eq. (14)
  Set $C^{ext}[\{1, \ldots, k\}] \leftarrow C$.
  Determine optimal $C_{i,k+1}^{ext} = C_{k+1,i}^{ext}$ by numerical optimization.
  return $i_{\max}$, $C^{ext} \in \mathbb{R}^{(k+1)\times(k+1)}$

function $\textsc{Filter}(\boldsymbol{x}, L, c, \varepsilon)$: See Algorithm 1

7.1 Improved 3-List Algorithm

The case $k = 3$ stands out from the above discussion as one can achieve a faster algorithm running the Configuration Extension Algorithm 6 on *two* points $\boldsymbol{v}_1, \boldsymbol{v}_2$. This case is also interesting in applications to lattice sieving, so we detail on it below.

From Eq. (14) we have $i_{\text{max}} = 2$, or more precisely, the running time of the 3-List algorithm (without Configuration Extension) is $T = |L_1| \cdot |L_2^{(1)}| \cdot |L_3^{(1)}|$. So we start shrinking the lists right from the beginning which corresponds to $m_{\text{old}} = 0$. For the balance configuration as the target, we have $C[I_{\text{lists}}] = -1/3$ on the off-diagonals. With the help of an optimization solver, we obtain the optimal values for $\langle \boldsymbol{x}_i , \boldsymbol{v}_j \rangle$ for $i = \{1, 2, 3\}$ and $j = \{1, 2\}$, and for $\langle \boldsymbol{v}_1 , \boldsymbol{v}_2 \rangle$ (there are 7 values to optimize for), so the input to Alg. 6 is determined. The target configuration is of the form

$$C = \begin{pmatrix} 1 & -1/3 & -1/3 & 0.47 & -0.15 \\ -1/3 & 1 & -1/3 & -0.17 & 0.26 \\ -1/3 & -1/3 & 1 & -0.19 & -0.14 \\ 0.47 & -0.17 & -0.19 & 1 & -0.26 \\ -0.15 & 0.26 & -0.14 & -0.26 & 1 \end{pmatrix} \tag{18}$$

and the number of instances is given by $R = \widetilde{\mathcal{O}}(1.4038^n)$ according to (17). The algorithm runs in a streamed fashion: the lists L_1', L_2', L_3' in line 2 of Alg. 3 are obtained instance by instance and, hence, lines 3 to 9 are repeated R times.

Algorithm 3 3-List with Configuration Extension

Input: L_1, L_2, L_3 – input lists of vectors from S^n, $|L| = 2^{0.1887n+o(n)}$
$C \in \mathbb{R}^{5\times 5}$ as in Eq. (18), $\rho = 1.4038$, $\varepsilon > 0$
Output: $L_{\text{out}} \subset L_1 \times L_2 \times L_3$, s.t. $|\langle \boldsymbol{x}_i , \boldsymbol{x}_j \rangle - C_{ij}| \le \varepsilon$, for all $1 \le i, j \le 3$.

1: $L_{\text{out}} \leftarrow \{\}$
2: $L_1', L_2', L_3' \leftarrow \mathsf{ConfExt}(k = 3, m_{\text{old}} = 0, m_{\text{new}} = 2, C \in \mathbb{R}^{5\times 5}, \varepsilon, L_1, L_2, L_3, (n_1, \dots, n_t))$
3:   **for all** $\boldsymbol{x}_1 \in L_1'$ **do**
4:     $L_2^{(1)} \leftarrow \text{Filter}(\boldsymbol{x}_1, L_2', -1/3, \varepsilon)$
5:     $L_3^{(1)} \leftarrow \text{Filter}(\boldsymbol{x}_1, L_3', -1/3, \varepsilon)$
6:     **for all** $\boldsymbol{x}_2 \in L_2^{(1)}$ **do**
7:       **for all** $\boldsymbol{x}_3 \in L_3^{(1)}$ **do**
8:         **if** $|\langle \boldsymbol{x}_2 , \boldsymbol{x}_3 \rangle + 1/3| \le \varepsilon$ **then**
9:           $L_{\text{out}} \leftarrow (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$
10: **return** L_{out}
1: **function** $\text{Filter}(\boldsymbol{x}, L, c, \varepsilon)$: See Algorithm 1

From Thm. 3, it follows that if the input lists satisfy $|L| = 2^{0.1887n+o(n)}$, then we expect $|L_{\text{out}}| = |L|$. Also from Eq. (8), it follows that the 3-List Algorithm 1 (i.e. without combining with the Configuration Extension Algorithm) has running time of $2^{0.3962n+o(n)}$. The above Alg. 3 brings it down to $2^{0.3717n+o(n)}$.

7.2 Application to the Shortest Vector Problem

In this section we briefly discuss how certain shortest vector algorithms can benefit from our improvement for the approximate k-List problem. We start by stating the approximate shortest vector problem.

On input, we are given a full-rank lattice $\mathcal{L}(B)$ described by a matrix $B \in \mathbb{R}^{n\times n}$ (with polynomially-sized entries) whose columns correspond to basis vectors, and some constant $\mathsf{c} \geq 1$. The task is to output a nonzero lattice vector $\boldsymbol{x} \in \mathcal{L}(B)$, s.t. $\|\boldsymbol{x}\| \leq \mathsf{c}\lambda_1(B)$ where $\lambda_1(B)$ denotes the length of the shortest nonzero vector in $\mathcal{L}(B)$. $\boldsymbol{x}$ is a solution to the approximate shortest vector problem.

The AKS sieving algorithm (introduced by Ajtai, Kumar, and Sivakumar in [1]) is currently the best (heuristic) algorithm for the approximate shortest vector problem: for an n-dimensional lattice, the running time and memory are of order 2^n. Sieving algorithms have two flavours: the Nguyen-Vidick sieve [20] and the Gauss sieve [19]. Both make polynomial in n number of calls to the approximate 2-List solver. Without LSH-techniques, the running time both the Nguyen-Vidick and the Gauss sieve is the running time of the approximate 2-List algorithm: $2^{0.415n+o(n)}$ with $2^{0.208n+o(n)}$ memory. Using our 3-List Algorithm 1 instead, the running time can be reduced to $2^{0.3962n+o(n)}$ (with only $2^{0.1887n+o(n)}$ memory) introducing essentially no polynomial overhead. Using Algorithm 3, we achieve even better asymptotics: $2^{0.3717n+o(n)}$, but it might be too involved for practical speed-ups due very large polynomial overhead for too little exponential gain in realistic dimensions.

Now we describe the Nguyen-Vidick sieve that uses a k-List solver as a main subroutine (see [4] for a more formal description). We start by sampling lattice-vectors $\boldsymbol{x} \in \mathcal{L}(B) \cap \mathsf{B}_n(2^{O(n)} \cdot \lambda_1(B))$, where $\mathsf{B}_n(R)$ denotes an n-dimensional ball of radius R. This can be done using, for example, Klein's nearest plane procedure [11]. In the k-List Nguyen-Vidick for $k > 2$, we sample many such lattice-vectors, put them in a list L, and search for k-tuples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in L \times \ldots \times L$ s.t. $\|\boldsymbol{x}_1 + \ldots + \boldsymbol{x}_k\| \leq \gamma \cdot \max_{1\leq i\leq k} \boldsymbol{x}_i$ for some $\gamma < 1$. The sum $\boldsymbol{x}_1 + \ldots + \boldsymbol{x}_k$ is put into L_{out}. The size of L is chosen in a way to guarantee that $|L| \approx |L_{\mathrm{out}}|$. The search for short k-tuples is repeated over the list L_{out}. Note that since with each new iteration we obtain vectors that are shorter by a constant factor γ, starting with $2^{\mathcal{O}(n)}$ approximation to the shortest vector (this property is guaranteed by Klein's sampling algorithm applied to an LLL-reduced basis), we need only linear in n iterations to find the desired $\boldsymbol{x} \in \mathcal{L}(B)$.

Naturally, we would like to apply our approximate k-List algorithm to k copies of the list L to implement the search for short sums. Indeed, we can do so by making a commonly used assumption: we assume the lattice-vectors we put into the lists lie uniformly on a spherical shell (on a very thin shell, essentially a sphere). The heuristic here is that it does not affect the behaviour of the algorithm. Intuitively, the discreteness of a lattice should not be "visible" to the algorithm (at least not until we find the approximate shortest vector).

We conclude by noting that our improved k-List Algorithm can as well be used within the Gauss sieve, which is known to perform faster in practice than the Nguyen-Vidick sieve. An iteration of the original 2-Gauss sieve as described in [19], searches for pairs $(\boldsymbol{p}, \boldsymbol{v})$, s.t. $\|\boldsymbol{p}+\boldsymbol{v}\| < \max\{\|\boldsymbol{p}\|, \|\boldsymbol{v}\|\}$, where $\boldsymbol{p} \in \mathcal{L}(B)$ is *fixed*, $\boldsymbol{v} \in L \subset \mathcal{L}(B)$, and $\boldsymbol{p} \neq \boldsymbol{v}$. Once such a pair is found and $\|\boldsymbol{p}\| > \|\boldsymbol{v}\|$, we set $\boldsymbol{p}' \leftarrow \boldsymbol{p}+\boldsymbol{v}$ and proceed with the search over $(\boldsymbol{p}', \boldsymbol{v})$, otherwise if $\|\boldsymbol{p}\| < \|\boldsymbol{v}\|$, we delete $\boldsymbol{v} \in L$ and store the sum $\boldsymbol{p}+\boldsymbol{v}$ as $\boldsymbol{p}$-input point for the next iteration. Once no pair is found, we add $\boldsymbol{p}'$ to L. On the next iteration, the search is repeated with another $\boldsymbol{p}$ which is obtained either by reducing some deleted $\boldsymbol{v} \in L$ before, or by sampling from $\mathcal{L}(B)$. The idea is to keep only those vectors in L that *cannot* form a pair with a shorter sum. Bai, Laarhoven, and Stehlé in [4], generalize it to k-Gauss sieve by keeping only those vectors in L that do not form a shorter k-sum. In the language of configuration search, we look for configurations $(\boldsymbol{p}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1}) \in \{\boldsymbol{p}\} \times L \times \ldots \times L$ where the first point is fixed, so we apply our Alg. 1 on $k-1$ (identical) lists.

Unfortunately, applying LSH / configuration extension-techniques for the Gauss Sieve is much more involved than for the Nguyen-Vidick Sieve. For $k=2$, [14] applies LSH techniques, but this requires an exponential increase in memory (which runs counter to our goal). We do not know whether these techniques extend to our setting. At any rate, since the gain from LSH / Configuration Extension techniques decreases with k (with the biggest jump from $k=2$ to $k=3$, while the overhead increases, gaining a practical speed-up from LSH / Configuration Extension within the Gauss sieve for $k \geq 3$ seems unrealistic.

Open questions. We present all our algorithms for a *fixed* k, and in the analysis, we suppress all the prefactors (in running time and list-sizes) for fixed k in the $\mathcal{O}_k(.)$ notation. Taking a closer look at how these factors depend on k, we notice (see, for example, the expression for $W_{n,k}$ in Thm. 1) that exponents of the polynomial prefactors depend on k. It prevents us from discussing the case $k \to \infty$, which is an interesting question especially in light of SVP. Another similar question is the optimal choice of ε and how it affects the pre-factors.

8 Experimental results

We implement the 3-Gauss sieve algorithm in collaboration with S. Bai [3]. The implementation is based on the program developed by Bai, Laarhoven, and Stehlé in [4], making the approaches comparable.

Lattice bases are generated by the SVP challenge generator [15]. It produces a lattice generated by the columns of the matrix

$$B = \begin{pmatrix} p & x_1 & \ldots & x_{n-1} \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix},$$

where p is a large prime, and $x_i < p$ for all i. Lattices of this type are random in the sense of Goldstein and Mayer [9].

For all the dimensions except 80, the bases are preprocessed with BKZ reduction of block-size 20. For $n = 80$, the block-size is 30. For our input lattices, we

	2-sieve	BLS 3-sieve	Alg. 1 for $k=3$			
			$\varepsilon = 0.0$	$\varepsilon = 0.015$	$\varepsilon = 0.3$	$\varepsilon = 0.4$
n	T, $\|L\|$	T, $\|L\|$	T, $\|L\|$	T, $\|L\|$	T, $\|L\|$	T, $\|L\|$
60	1.38e3, 13257	1.02e4, 4936	1.32e3, 7763	1.26e3, 7386	1.26e3, 6751	**1.08e3, 6296**
62	2.88e3, 19193	1.62e4, 6239	2.8e3, 10356	3.1e3, 9386	**1.8e3, 8583**	2.2e3, 8436
64	8.64e3, 24178	5.5e4, 8369	5.7e3, 13573	3.6e3, 12369	**3.36e3, 11142**	4.0e4, 10934
66	1.75e4, 31707	9.66e4, 10853	1.5e4, 17810	1.38e4, 16039	**9.1e3, 14822**	1.2e4, 14428
68	3.95e4, 43160	2.3e5, 14270	2.34e4, 24135	2.0e4, 21327	**1.68e4, 19640**	1.86e4, 18355
70	6.4e4, 58083	6.2e5, 19484	6.21e4, 32168	3.48e5, 26954	**3.3e4, 25307**	3.42e4, 24420
72	2.67e5, 77984	1.2e6, 25034	7.6e4, 40671	7.2e4, 37091	**6.16e4, 34063**	6.35e4, 34032
74	3.45e5, 106654	—	2.28e5, 54198	2.08e5, 47951	**2.02e5, 43661**	2.03e5, 40882
76	4.67e5, 142397	—	3.58e5, 71431	2.92e5, 64620	**2.42e5, 56587**	2.53e5, 54848
78	9.3e5, 188905	—	—	—	**4.6e5, 74610**	4.8e5, 70494
80	—	—	—	—	**9.47e5, 98169**	9.9e5, 98094

Table 1: Experimental results for k-tuple Gauss sieve. The running times T are given in seconds, $|L|$ is the maximal size of the list L. ε is the approximation parameter for the subroutine Filter of Alg. 1. The best running-time per dimension is type-set bold.

do not know their minimum λ_1. The algorithm terminates when it finds many linearly dependent triples $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$. We set a counter for such an event and terminate the algorithm once this counter goes over a pre-defined threshold. The intuition behind this idea is straightforward: at some point the list L will contain very short basis-vectors and the remaining list-vectors will be their linear combinations. Trying to reduced the latter will ultimately produce the zero-vector. The same termination condition was already used in [17], where the authors experimentally determine a threshold of such “zero-sum” triples.

Up to $n = 64$, the experiments are repeated 5 times (i.e. on 5 random lattices), for the dimensions less than 80, 3 times. For the running times and the list-sizes presented in the table below, the average is taken. For $n = 80$, the experiment was performed once.

Our tests confirm a noticeable speed-up of the 3-Gauss sieve when our Configuration Search Algorithm 1 is used. Moreover, as the analysis suggests (see Fig. 3), our algorithm outperforms the naive 2-Gauss sieve while using much less memory. The results can be found in Table 1.

Another interesting aspect of the algorithm is the list-sizes when compared with BLS. Despite the fact that, asymptotically, the size of the list $|L|$ is the same for our and for the BLS algorithms, in practice our algorithm requires a longer list (cf. the right numbers in each column). This is due to the fact that we filter out a larger fraction of solutions. Also notice that increasing ε – the approximation to the target configuration, we achieve an additional speed-up. This becomes obvious once we look at the Filter procedure: allowing for a smaller inner-product throws away less vectors, which in turn results in a shorter list L. For the range of dimensions we consider, we experimentally found $\varepsilon = 0.3$ to be a good choice.

Acknowledgments. We would like to thank the authors of [4], Shi Bai, Damien Stehlé, and Thijs Laarhoven for constructive discussions. A similar (unpublished) algorithm to ours was used by Thorsten Kleinjung[12] to break the $n = 116$ SVP instance of the Darmstadt lattice challenge[15].

Elena Kirshanova was supported by UbiCrypt, the research training group 1817/1 funded by the DFG. Gottfried Herold was funded by ERC grant 307952 (acronym FSC).

References

1. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of STOC*, pages 601–610, 2001.
2. A. G. Akritas, E. K. Akritas, and G. I. Malaschonok. Sybolic computation, new trends and developments various proofs of sylvester's (determinant) identity. *Mathematics and Computers in Simulation*, 42(4):585 – 593, 1996.
3. S. Bai, August 2016. Personal Communication.
4. S. Bai, T. Laarhoven, and D. Stehlé. Tuple Lattice Sieving. In *ANTS-XII (to appear)*, 2016.
5. A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In R. Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24. SIAM, 2016.
6. A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, pages 506–519, 2003.
7. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
8. M. Ghosh and B. K. Sinha. A simple derivation of the wishart distribution. *The American Statistician*, 56(2):100–101, 2002.
9. D. Goldstein and A. Mayer. On the equidistribution of hecke points. *Forum Mathematicum*, 15(3):165–189, 01 2006.
10. R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of STOC*, pages 193–206, 1983.
11. P. Klein. Finding the closest lattice vector when it's unusually close. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2000, pages 937–941, 2000.
12. T. Kleinjung, January 2017. Personal Communication.

13. O. Kupferman, Y. Lustig, and M. Y. Vardi. *On Locally Checkable Properties*, pages 302–316. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
14. T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 3–22, Santa Barbara, CA, USA, Aug. 16–20, 2015. Springer, Heidelberg, Germany.
15. R. Lindner, M. Rückert, P. Baumann, and L. Nobach. SVP challenge generator. `http://latticechallenge.org/svp-challenge`.
16. V. Lyubashevsky. On random high density subset sums. In *APPROX-RANDOM, LNCS 3624*, pages 378–389, 2005.
17. A. Mariano, T. Laarhoven, and C. Bischof. Parallel (probable) lock-free hash sieve: A practical sieving algorithm for the svp. In *44th International Conference on Parallel Processing (ICPP)*, pages 590–599, September 2015.
18. A. May and I. Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 203–228, Sofia, Bulgaria, Apr. 26–30, 2015. Springer, Heidelberg, Germany.
19. D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2010, pages 1468–1480, 2010.
20. P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. In *Journal of Mathematical Cryptology*, volume 2, pages 181–207, 2008.
21. D. Wagner. Generalized birthday problem. In *Proceedings of CRYPTO 2002, LNCS 2442*, pages 288–304, 2002.
22. J. Wishart. The generalized product moment distribution in samples from a normal multivariate population. *Biometrika*, 20A(1-2):32–52, 1928.

A Details on Configuration Extension

A.1 Outline

In this section, we describe our algorithm ConfExt for configuration extension from Sect. 6, achieving what was claimed in Thm. 5. We will first give an informal outline of our algorithm. For technical reasons, we need to briefly discuss conditional densities and probabilities that will appear later (and also implicitly appear in Thm. 5, depending on how one interprets the $\approx$-notion there) in Sec. A.2. Our algorithm ConfExt relies on a concentration result for configurations, namely that configurations look locally the same as globally. This is discussed in section A.3.

Following that, we formally define ConfExt and give an extensive analysis in Thm. 7. This theorem will directly imply Thm. 5 from Sect. 6, which is a toned-down version of Thm. 7.

Let us start with the informal outline.

Recall that we want to solve the following problem: we are given m_{old} points $\boldsymbol{v}_j$, $j \in I_{\mathrm{old}}$ with given configuration and k lists $L_1, \ldots, L_k$, where the elements from each L_i have prescribed distance to each $\boldsymbol{v}_j$. We want to extend this situation by adding more $\boldsymbol{v}_\ell$'s, $\ell \in I_{\mathrm{new}}$, where the full configurations of all old and new $\boldsymbol{v}_j$'s resp. $\boldsymbol{v}_\ell$'s are prescribed. Furthermore, we want to shrink the lists L_i

to L'_i. The lists L_i are shrunk because we now also have prescribed distances to the newly added points. The various prescribed distances are organized in a single configuration C, giving the scalar products between old $\boldsymbol{v}_j$, $j \in I_{\text{old}}$, new $\boldsymbol{v}_\ell$, $\ell \in I_{\text{new}}$ and list elements $\boldsymbol{x}_i$, $i \in I_{\text{lists}}$.

The obvious way to solve this problem is as follows: sample some $\boldsymbol{v}_\ell$'s, conditioned on the prescribed configuration $C[I_{\text{old}}, I_{\text{new}}]$. This is relatively easy to do: an algorithm to sample such $\boldsymbol{v}_\ell$'s is given in Alg. 5 below. Then shrink the lists L_i according to the prescribed distances to the newly constructed points, obtaining L'_i. To actually shrink the list L_i to L'_i for given $\boldsymbol{v}_\ell$'s, we just iterate over L_i and check each configuration. We call this *trimming*[2] the list L_i.

If we use the simple approach from above, then trimming will cost $\mathcal{O}(|L_i|)$ each time we shrink a list. This is too costly: the result we want to prove only allots a running time of $\widetilde{\mathcal{O}}(|L'_i|)$ to each time we shrink a list.

What helps us is that we do not wish to create a single set of $\boldsymbol{v}_\ell$'s, but rather an exponential number R of instances $\boldsymbol{v}'^{(r)}_\ell$ and $L'^{(r)}_i$, $1 \leq r \leq R$ and we will amortize the cost of trimming the list among these R instances. Indeed, we can bring the amortized cost down to essentially (up to subexponential factors) the output size. To do so, we use the block coding / stripe technique of [5,18] and do not trim the list in one go. Rather, we trim the lists in t iterative steps, such that after each step, we only have to operate on smaller lists. To make this work out, we need that an exponentially large number of r's share the initial steps (which are costly, since they operate on large lists, but we amortize them over a large number of instances). For this, we will choose the $\boldsymbol{v}'^{(r)}_\ell$ in a structured, block-wise way. Notably, subdivide the $n+1$ coordinates of points from $\mathsf{S}^n \subset \mathbb{R}^{n+1}$ into t blocks as $n+1 = n_1 + \ldots n_t$. Then we first choose $R_1 \ll R$ possible $\boldsymbol{v}^{(r_1)}_{\ell,1}$'s for $1 \leq r_1 \leq R$, where each $\boldsymbol{v}^{(r_1)}_{\ell,1}$ only has n_1 coordinates. For each r_1, we further choose $R_2 \ll R$ possible $\boldsymbol{v}^{(r_2)}_{\ell,2}$ on the next n_2 coordinates and so on, until $R = R_1 \cdots R_t$.

The crucial observation is that we can enforce our configuration constraints on each such subset of coordinates separately. Indeed, if the subdivisions $n+1 = n_1 + \ldots + n_t$ are chosen such that t is constant and for each $1 \leq s \leq t$ the ratio $\frac{n_s}{n} \xrightarrow{n\to\infty} c_s$ with c_s constant, then up to subexponential factors, any configuration constraint $\mathrm{Conf}(\boldsymbol{x}_1, \ldots) = C$ on all $n+1$ coordinates also holds (after rescaling) locally on any such block of n_s coordinates and vice versa. This means that we may start trimming the lists L_i once we know $\boldsymbol{v}^{(r_1)}_{\ell,1}$, i.e. only n_1 coordinates, hereby amortizing cost.

To give a formal analysis, we first introduce such local configurations on subdivisions of coordinates. To formalize that we can switch from global to local configurations, we prove a concentration result in the spirit of Thm. 2. We then give the algorithm and analyze its cost and properties.

[2] Conceptually, trimming and filtering (as in Filter from Alg. 1 from Sect. 4) are the same. The only difference is that trimming will work block-wise and we call the algorithm Trim for disambiguation, since the parameters passed to it are slightly different from Filter.

A.2 Conditional Densities

In this section, we will give a definition of expressions of the form "$\boldsymbol{x}_i$ are uniform, conditioned on their configuration $\mathrm{Conf}(\ldots)$ and $\boldsymbol{v}_j$'s, with $\mathrm{Conf}(\ldots) \approx_\varepsilon C'$" that will appear in this section, and that also imlicitly already appeared in the statement of Thm. 5. Furthermore, we prove a lemma that allows us to compute how much lists (that already have been shrunk by imposing configuration constraints) shrink when imposing further configuration constraints. The latter Lemma 2 is the fundamental tool that allows us to compute list sizes and the necessary number of instances/repetitions R in our ConfExt-algorithm.

Recall that, since we deal with real numbers, we had to introduce some $\varepsilon > 0$ and ask for ε-closeness rather than equality of reals. Our approach is to analyze the complexity of configuration extension for fixed ε. If we then let $\varepsilon \to 0$, we can hide the effect of ε in the subexponential factors suppressed in the $\widetilde{\mathcal{O}}(.)$-notation. Conceptually, this means that we really are only interested in the behavior for $\varepsilon \to 0$ and it would be nice to be able to talk only about the limiting case in the first place. The only comparisons we ever do are between configurations of points, not points themselves (or, more generally following our block-wise strategy, configurations and lengths on blocks of coordinates). In particular, directly talking about the limiting case involves talking about distributions of points, where we condition on the configuration. This means, we want to be able to talk about the uniform distribution of k points $\boldsymbol{x}_i \in \mathsf{S}^n$, conditioned on $\mathrm{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) \approx_\varepsilon C'$ resp. $\mathrm{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = C'$ for given C'. The issue with the latter is that this is, strictly speaking, ill-defined, since a given configuration occurs only with probability 0. We still write this as an abuse of notation. The reader may think of this as implicit limit of $\varepsilon \to 0$ and the ill-definedness can actually be safely ignored.

Actually, it is possible to condition on a well-behaved set of measure zero, but the result depends on some additional choices. We only give a very rough sketch of how this can be done here for the interested reader. This paragraph can be safely skipped (until "ε-terms") and we do not assume the reader to be familiar with the methods used herein for the rest of this paper.

It is possible to condition a distribution defined on a v-dimensional V with continuous density ρ_V on a set $W = \{g_1 = \cdots = g_\kappa = 0\} \subset V$ of dimension $w = v - \kappa < v$ (and, hence, measure zero), defined as the zero-set of well-behaved[3] functions $g_1, \ldots, g_\kappa$. The most straight-forward way is to condition on $W_\varepsilon = \{x \in V \mid |g_1(x)| \leq \varepsilon, \ldots, |g_\kappa(x)| \leq \varepsilon\}$ and letting $\varepsilon \to 0$. This will converge in distribution to the "correct" object. However, the result will depend on the choice of functions $g_1, \ldots, g_\kappa$ rather than just on W. In a more infinitesimal way of thinking, we may think of ρ_V (locally) as a differential v-form $\rho_V \mathrm{d}V$. Then a conditional density $\rho_W \mathrm{d}W$ should satisfy the property that $\rho_V \mathrm{d}V$ is (up to scaling) the product of the conditional distribution $\rho_W \mathrm{d}W$ on W and some transversal[4] $(v-w)$-form ν. The $(v-w)$-form ν should be thought of as

[3] We assume that g_i's are smooth and their Jacobian has full rank on W.

[4] We require that along W, the kernel of ν is exactly the tangent space TW of W. One may think of ν as being a density "orthogonal" to $\mathrm{d}W$. However, we have no

corresponding to the distribution of $\boldsymbol{g}(x)$, so the condition $\rho_V \mathrm{d}V = (\rho_W \mathrm{d}W) \cdot \nu$ corresponds to the defining property of conditional distributions/probabilities. Indeed, this can be used as a definition of $\rho_W \mathrm{d}W$, which depends on the choice of ν. If we choose ν via the differentials of the g_i, these approaches[5] coincide.

In our case, the natural defining functions $g_i(.)$ are usually of the form $f_i(\mathrm{Conf}(.))$ and the prior distribution is uniform on (k-fold products of) spheres S^n. This has two important consequences: Firstly, the choice of f_i's (for given W) does not matter asymptotically, provided the choice of f_i's is independent from n. The reason is that the f_i's are only defined on a compact set and any change in f_i's (and their derivatives) can therefore only be by a bounded function, independent from n. Secondly, and more importantly, all distributions we look at will be invariant under simultaneous rotations of all points. Since $\mathrm{Conf}(.)$ is unchanged under such simultaneous rotations, this invariance property of the involved distributions is preserved when conditioning. As a consequence, we can obtain the distribution of k-tuples $(\boldsymbol{x}_i)_{1\le i\le k}$'s from the distribution of $\mathrm{Conf}\big((\boldsymbol{x}_i)_{1\le i\le k}\big)$ by taking any one $(\boldsymbol{x}_i)_{1\le i\le k}$ with the prescribed configuration and applying a random rotation $\mathbf{A} \in \mathbb{O}_n$ to each $\boldsymbol{x}_i$. Here, the orthogonal group $\mathbb{O}_n$ is equipped with its Haar measure (left and right Haar measure are equal for $\mathbb{O}_n$). Consequently, the uniform distribution of $(\boldsymbol{x}_i)_{1\le i\le k}$ conditioned on their configuration, satisfies the property that for any $\mathbf{B} \in \mathbb{O}_n$, the distribution of $(\mathbf{B}\boldsymbol{x}_i)_{1\le i\le k}$ is the same as the distribution of $(\boldsymbol{x}_i)_{1\le i\le k}$.

ε-terms. In the following sections, we will often include ε-terms in our statements and avoid talking about the limit-case only. The main reason is that we need ε-terms for the transition to configuration defined on blocks of coordinates. A limit-case only concentration result for such blocks would become awkward to state for the purely technical reason that in our formal proofs we need to care about speed of convergence. Furthermore, these ε-terms have some consequences for the design of the algorithms, so a treatment including those feels more complete. Surprisingly, it turns out that we can design ConfExt in such a way that the ε in the output is the same as the ε in the input, so we can avoid error propagation. Still, the reader is safe to ignore any ε-terms.

---

meaningful notion of orthogonality or a scalar product here. If such a notion was available, we could indeed define ν via the orthogonal complement of (the tangent space of) W, which would give rise to a uniquely defined conditional density. This is how the w-dimensional Lebesgue measure on w-dimensional subspaces of $\mathbb{R}^v$ may be obtained from the v-dimensional Lebesgue measure on $\mathbb{R}^v$. Note that since the kernel of ν is TW, we may also view ν as a nonwhere vanishing section in $\Lambda^{v-w}(TV/TW)^*$. Since this is one-dimensional, any two different choices of ν differ multiplicatively by a nowhere vanishing function h. A change of ν then changes $\rho_V \mathrm{d}V$ by a factor h^{-1}, up to rescaling.

[5] The first approach may feel more natural and is the point of view we shall take; the second one avoids limits and the question of whether these limits exist and are finite/well-behaved. It is easy to verify using the calculus of differential forms that both approaches coincide and the second approach serves for us as a justification of the first one.

One case where we actually do make use of conditioning on lower-dimensional sets is the following: we use expressions of the form

"uniform $(\boldsymbol{x}_i)_{i\in I_{\text{lists}}}$, conditioned on $\boldsymbol{v}_j$'s with $j \in I_{\text{old}}$ and on $\mathrm{Conf}((\boldsymbol{x}_i)_{1\leq i\leq k}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}})$, where $\mathrm{Conf}((\boldsymbol{x}_i)_{1\leq i\leq k}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}}) \approx_\varepsilon C'$"

where C' is some given configuration. By the above discussion, this means the following:

We take points $\boldsymbol{x}_i$, where we assume that $\mathrm{Conf}((\boldsymbol{x}_i)_{1\leq i\leq k}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}}) \approx_\varepsilon C'$, but we make no further assumption about the actual distribution of the configurations $\mathrm{Conf}((\boldsymbol{x}_i)_{1\leq i\leq k}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}})$. For the distribution of the points $\boldsymbol{x}_i, \boldsymbol{v}_j$ it means the following: for any $\mathbf{B} \in \mathbb{O}_n$ that fixes all $\boldsymbol{v}_j$'s, the rotated $(\mathbf{B}\boldsymbol{x}_i)_{i\in I_{\text{lists}}}$ follow the same joint distribution as $(\boldsymbol{x}_i)_{i\in I_{\text{lists}}}$.

For the analysis of list sizes in our algorithm, the central tool is the following lemma, which we only state in an $\varepsilon = 0$-variant.

Lemma 2 (Shrinking Lemma). *Let $k, m_{\text{old}}, m_{\text{new}} > 0$ and consider disjoint index sets $I_{\text{lists}}, I_{\text{old}}, I_{\text{new}}$ with $|I_{\text{lists}}| = k, |I_{\text{old}}| = m_{\text{old}}, |I_{\text{new}}| = m_{\text{new}}$. We will treat $I_{\text{lists}}, I_{\text{old}}, I_{\text{new}}$ as index sets for a given non-singular configuration $C' \in \mathbb{R}^{(k+m_{\text{old}}+m_{\text{new}})\times(k+m_{\text{old}}+m_{\text{new}})}$. These data shall not depend on n.*

Let $\boldsymbol{x}_i \in \mathsf{S}^n$ for $i \in I_{\text{lists}}$ and $\boldsymbol{v}_j \in \mathsf{S}^n$ with $\mathrm{Conf}((\boldsymbol{x}_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}}) = C'[I_{\text{lists}}, I_{\text{old}}]$. We sample a uniformly random $(\boldsymbol{w}_\ell) \in \mathsf{S}^n$, conditioned on the $\boldsymbol{v}_j$'s and $\mathrm{Conf}((\boldsymbol{v}_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}) = C'[I_{\text{old}}, I_{\text{new}}]$. Then we have

$$\begin{aligned}&\Pr\Big[\mathrm{Conf}\big((\boldsymbol{x}_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}}\big) = C'[I_{\text{lists}}, I_{\text{old}}]\Big]\\ =&\widetilde{\mathcal{O}}\bigg(\Big(\frac{\det C'[I_{\text{lists}}, I_{\text{old}}, I_{\text{new}}]\det C'[I_{\text{old}}]}{\det C'[I_{\text{lists}}, I_{\text{old}}]\det C'[I_{\text{old}}, I_{\text{new}}]}\Big)^{\frac{n}{2}}\bigg),\end{aligned}$$

where the equality in $\Pr\big[\mathrm{Conf}(\dots) = C'[\dots]\big]$ means that we take the appropriate density wrt. the reference density $\prod_{i\in I_{\text{lists}},\ell\in I_{\text{new}}} \mathrm{d}C'_{i,\ell}$.

Proof. The proof is conceptually a rather simple computation with conditional probabilities. The hard part is actually rigorously justifying the computation.

We abused notation and wrote $=$ when we meant a density. To compute the density, we may consider the probability p_ε of $\mathrm{Conf}(\dots) \approx_\varepsilon C'[\dots]$ and take the limit $\lim_{\varepsilon\to 0} \frac{p_\varepsilon}{g(\varepsilon)}$ for some $g(\varepsilon)$. The function $g(\varepsilon)$ depends on the reference density with respect to which we compute the density and its meaning is the volume of the set of C's with $C \approx_\varepsilon C'$. In our case, $g(\varepsilon) = (2\varepsilon)^d$, where $d = km_{\text{new}}$ is the dimension of the space where the density is defined.

Conditioning on an equality is defined by a similar limit (without such a $g(\varepsilon)$). In our computation, we will get several different $\varepsilon_1, \varepsilon_2, \dots$-terms. The central observation is that the limits of $\varepsilon_i \to 0$ all commute and we may replace the different ε_i's by just a single ε.

Write $A^\varepsilon(I_{\text{lists}})$ for the statement $\mathrm{Conf}((\boldsymbol{x}_i)_{i\in I_{\text{lists}}}) \approx_\varepsilon C'[I_{\text{lists}}]$. Similarly, $A^\varepsilon(I_{\text{lists}}, I_{\text{old}})$ is shorthand for $\mathrm{Conf}((\boldsymbol{x}_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}}) \approx_\varepsilon C'[I_{\text{lists}}, I_{\text{old}}]$. Other statements $A^\varepsilon(I_\alpha, \dots)$ are defined analogously. We write $A(\dots)$ without superscript for the statement with $=$ rather than $\approx_\varepsilon$.

The statement only depends on the configuration of $\boldsymbol{x}_i$, not on the actual points. So we may imagine that the $(\boldsymbol{x}_i)_{i\in I_{\text{lists}}}$ and $(\boldsymbol{v}_j)_{j\in I_{\text{old}}}$ are sampled uniformly from S^n, conditioned on the configuration being $C'[I_{\text{lists}}, I_{\text{old}}]$. We can then compute

$$\begin{aligned}\Pr_{(\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}}&\Big[\mathrm{Conf}\big((\boldsymbol{x}_i)_{i\in I_{\text{lists}}},(\boldsymbol{v}_j)_{j\in I_{\text{old}}}\big)=C'[I_{\text{lists}},I_{\text{old}}]\Big]\\ &=\Pr\big[A(I_{\text{lists}},I_{\text{new}})\ \big|\ A(I_{\text{lists}},I_{\text{old}}),A(I_{\text{old}},I_{\text{new}})\big]\\ =\lim_{\varepsilon_1\to 0}\lim_{\varepsilon_2,\varepsilon_3\to 0}&\ \frac{\Pr\big[A^{\varepsilon_1}(I_{\text{lists}},I_{\text{new}})\ \big|\ A^{\varepsilon_2}(I_{\text{lists}},I_{\text{old}}),A^{\varepsilon_3}(I_{\text{old}},I_{\text{new}})\big]}{g(\varepsilon_1)}\end{aligned}$$

Since these limits commute, this is equal to

$$\begin{aligned}&\lim_{\varepsilon\to 0}\frac{\Pr\big[A^{\varepsilon}(I_{\text{lists}},I_{\text{new}})\ \big|\ A^{\varepsilon}(I_{\text{lists}},I_{\text{old}}),A^{\varepsilon}(I_{\text{old}},I_{\text{new}})\big]}{g(\varepsilon)}\\ &=\lim_{\varepsilon\to 0}\frac{\Pr\big[A^{\varepsilon}(I_{\text{lists}},I_{\text{old}},I_{\text{new}})\big]}{\Pr\big[A^{\varepsilon}(I_{\text{lists}},I_{\text{old}}),A^{\varepsilon}(I_{\text{old}},I_{\text{new}})\big]g(\varepsilon)}\\ &=\lim_{\varepsilon\to 0}\frac{\Pr\big[A^{\varepsilon}(I_{\text{lists}},I_{\text{old}},I_{\text{new}})\big]}{\Pr\big[A^{\varepsilon}(I_{\text{lists}})\ \big|\ A^{\varepsilon}(I_{\text{old}})\big]\cdot\Pr\big[A^{\varepsilon}(I_{\text{new}})\ \big|\ A^{\varepsilon}(I_{\text{old}})\big]\cdot\Pr\big[A^{\varepsilon}(I_{\text{old}})\big]g(\varepsilon)}\\ &=\lim_{\varepsilon\to 0}\frac{\Pr\big[A^{\varepsilon}(I_{\text{lists}},I_{\text{old}},I_{\text{new}})\big]\cdot\Pr\big[A^{\varepsilon}(I_{\text{old}})\big]}{\Pr\big[A^{\varepsilon}(I_{\text{lists}},I_{\text{old}})\big]\cdot\Pr\big[A^{\varepsilon}(I_{\text{old}},I_{\text{new}})\big]g(\varepsilon)}.\end{aligned}$$

Here, we used that, conditioned on $A^\varepsilon(I_{\text{old}})$, the statements $A^\varepsilon(I_{\text{old}}, I_{\text{new}})$ and $A^\varepsilon(I_{\text{lists}}, I_{\text{old}})$ are independent. The probability space is over uniform $\boldsymbol{x}_i, \boldsymbol{v}_j, \boldsymbol{w}_\ell$ unless specified otherwise. Plugging in Thm. 1 into the last expression directly gives the desired density: we have

$$\Pr[A^\varepsilon(I_1,\dots)]=\int_{C[I_1,\dots]\approx_\varepsilon C'[I_1,\dots]}\mu_{\mathscr{C}}(C[I_1,\dots])\mathrm{d}C[I_1,\dots],$$

where $\mu_{\mathscr{C}}(C[I_1,\dots]) = \widetilde{\mathcal{O}}(\det(C[I_1,\dots])^{\frac{n}{2}})$. Note that the $g(\varepsilon)$-factor is (by construction) nothing but the quotient of the volume over which we integrate in the numerator and the volume over which we integrate in the denominator. Consequently, taking $\varepsilon\to 0$, the integration just gives the value of the integrand. This is exactly what we wanted to show.

A.3 Local Configurations

Definition 5. *Let $n+1 = n_1+\ldots+n_t$ and $\boldsymbol{x}_1,\dots,\boldsymbol{x}_k\in\mathsf{S}^n\subset\mathbb{R}^{n+1}$. Write $\boldsymbol{x}_i$ as $\boldsymbol{x}_i = \boldsymbol{x}_{i,1},\dots,\boldsymbol{x}_{i,t}$ for $\boldsymbol{x}_{i,s}\in\mathbb{R}^{n_s}$. We denote by $\mathrm{L}_s(\boldsymbol{x}_i) := \|\boldsymbol{x}_{i,s}\|^2$ and $\mathrm{L}(\boldsymbol{x}_i) := \big(\mathrm{L}_1(\boldsymbol{x}_i),\dots,\mathrm{L}_s(\boldsymbol{x}_i)\big)$ the square-length distribution of $\boldsymbol{x}_i$ wrt. the decomposition $n+1 = n_1+\ldots+n_t$. We define the s^{th} local configuration of $\boldsymbol{x}_1,\dots,\boldsymbol{x}_k$ wrt. the decomposition $n+1=n_1+\ldots+n_t$ to be*

$$\mathrm{Conf}_s(\boldsymbol{x}_1,\dots,\boldsymbol{x}_k) := C\in\mathbb{R}^{k\times k},\quad C_{i,j}=\frac{\langle\boldsymbol{x}_{i,s}\ ,\boldsymbol{x}_{j,s}\rangle}{\|\boldsymbol{x}_{i,s}\|\|\boldsymbol{x}_{j,s}\|}.$$

The total *configuration of the k-tuple $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ is defined as the sequence of all* $\mathrm{Conf}_s(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ *for* $1 \leq s \leq t$ *together with the sequence of square-length distributions* $\mathrm{L}(\boldsymbol{x}_1), \ldots, \mathrm{L}(\boldsymbol{x}_k)$*. By contrast, we call* $\mathrm{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ *the* global *configuration of* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ *if we want to emphasize that we take all coordinates.*

Note that local configurations are undefined if some $\boldsymbol{x}_{i,s} = 0$*, but this only happens on a set of measure 0.*

It is easy to see that the total configuration determines the global configuration. Indeed, for $n+1 = n_1 + \ldots + n_t$, we have

$$\mathrm{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = D_1 \mathrm{Conf}_1(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) D_1 + \ldots + D_t \mathrm{Conf}_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) D_t, \tag{19}$$

where the D_s are diagonal matrices with entries $\|\boldsymbol{x}_{1,s}\|, \ldots, \|\boldsymbol{x}_{k,s}\|$.

The proof of our concentration result uses the well-known fact that $\log\det$ is a concave function on positive definite symmetric matrices:

Lemma 3. *Let $k > 0$ and consider the space $\mathscr{S}_k$ of all positive definite symmetric $k \times k$ matrices (i.e. $\mathscr{S}_k$ is the interior of $\mathscr{C}$). Then the map* $\log\det \colon \mathscr{S}_k \to \mathbb{R}$ *is strictly concave.*

Proof. This result can be found in a lot of textbooks. See e.g. [7, p.73].

We now state our concentration result for total vs. global configurations.

Theorem 6. *Let $\varepsilon > 0$. Let t, k be fixed and consider the decomposition $n+1 = n_1 + \ldots + n_t$. For the asymptotics with $n \to \infty$, we assume that each $\frac{n_s}{n+1}$ converges to a constant $\neq 0, 1$. Consider a fixed global non-singular configuration $C \in \mathbb{R}^{k \times k}$. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathsf{S}^n$ be uniform, conditioned on $\mathrm{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = C$. Consider their total configuration $\overline{C} = (C_1, \ldots, C_t, \mathrm{L}(\boldsymbol{x}_1), \ldots, \mathrm{L}(\boldsymbol{x}_k))$.*

Then with probability exponentially close to 1, we simultaneously have $C_s \approx_\varepsilon C$ for all s and $\left|\mathrm{L}_s(\boldsymbol{x}_i) - \frac{n_s}{n}\right| \leq \varepsilon$ for all i, s.

Proof. We will compare the probability densities for global and total configurations in this proof. Notably, our proof will essentially show that the probability density for total configurations $\overline{C}$ is of the form $\widetilde{\mathcal{O}}(f(\overline{C})^n)$ for some function f. As this is exponential, the conditional probability for $\overline{C}$ conditioned on some global configuration C, is determined by the maximum of f among all $\overline{C}$ giving rise to the prescribed global configuration. We will see that the unique maximum is attained when the square-length distribution is as expected and all local configurations are equal.

Let us now start with analyzing the probability densities for global and total configurations, without conditioning on C yet. In order to get better independence properties between the blocks, we consider vectors $\boldsymbol{y}_i$ following a Gaussian distribution on $\mathbb{R}^{n+1}$ instead of vectors $\boldsymbol{x}_i \in \mathsf{S}^n$. More precisely, let $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k \in \mathbb{R}^{n+1}$, where each coordinate follows an independent continuous Gaussian with variance $\frac{1}{n+1}$ per coordinate (so the expected value of $\|\boldsymbol{y}_i\|^2$ is 1). This way, the directions $\boldsymbol{x}_i := \frac{\boldsymbol{y}_i}{\|\boldsymbol{y}_i\|}$ are uniform. The lengths $\|\boldsymbol{y}_i\|$ are independent from the directions $\boldsymbol{x}_i$ and, by definition, $(n+1) \cdot \|\boldsymbol{y}_i\|^2$ follows a

χ^2-distribution with $n+1$ degrees of freedom. Let $\mathbf{A}$ be the Gram matrix of the $\boldsymbol{y}_1,\ldots,\boldsymbol{y}_k$. The diagonal entries of $\mathbf{A}$ are the squared lengths $\|\boldsymbol{y}_i\|^2$ and we have $\langle \boldsymbol{x}_i\,,\boldsymbol{x}_j\rangle = \frac{A_{i,j}}{\sqrt{A_{i,i}A_{j,j}}}$. So $\mathbf{A}$ determines $C=\mathrm{Conf}(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_k)$.

We denote the individual blocks, according to the decomposition $n+1 = n_1+\ldots+n_t$, of the the $\boldsymbol{y}_i$ by $\boldsymbol{y}_{i,s}$ for $1\le s\le t$. The local Gram matrices of the $\boldsymbol{y}_{1,s},\ldots,\boldsymbol{y}_{k,s}$ are denoted by $\mathbf{A}_s$, accordingly. By definition, the $\boldsymbol{y}_{i,s}$ are all independent and so are the local Gram matrices. We have $\mathbf{A}=\sum_s \mathbf{A}_s$ and the set of local Gram matrices fully determines the total configuration of the $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_k$.

Let us normalize to $\mathbf{B}\coloneqq\mathbf{A},\mathbf{B}_s\coloneqq\frac{n+1}{n_s}\mathbf{A}_s$, such that all the expected values on the diagonals of $\mathbf{B},\mathbf{B}_s$ are equal to 1.

The distribution of $\mathbf{B}$ is given by the Wishart distribution (cf. (2)). Its probability density is given by

$$\mu_{\mathbf{B}} = \frac{(n+1)^{\frac{k(n+1)}{2}}(\det\mathbf{B})^{\frac{n-k}{2}}e^{-\frac{n+1}{2}\,\mathrm{Tr}\,\mathbf{B}}}{2^{\frac{k(n+1)}{2}}\pi^{\frac{k(k-1)}{4}}\prod_{i=0}^{k-1}\Gamma(\frac{n+1-i}{2})}\mathrm{d}\mathbf{B}, \tag{20}$$

where the reference density is $\mathrm{d}\mathbf{B}=\prod_{i\le j}\mathrm{d}B_{i,j}$. Note that this differs slightly from (2) due to the normalization of $\|\boldsymbol{y}\|^2$ to expected value 1. Using Stirling's formula, we compute

$$\begin{aligned}
\mu_{\mathbf{B}} &= \frac{(n+1)^{\frac{k(n+1)}{2}}(\det\mathbf{B})^{\frac{n-k}{2}}e^{-\frac{n+1}{2}\,\mathrm{Tr}\,\mathbf{B}}}{2^{\frac{k(n+1)}{2}}\pi^{\frac{k(k-1)}{4}}\prod_{i=0}^{k-1}\Gamma(\frac{n+1-i}{2})}\mathrm{d}\mathbf{B}\\
&= \mathcal{O}_k\left(\frac{n^{\frac{kn}{2}}n^{\frac{k}{2}}(\det\mathbf{B})^{\frac{n-k}{2}}e^{-\frac{n+1}{2}\,\mathrm{Tr}\,\mathbf{B}}}{2^{\frac{kn}{2}}\prod_{i=0}^{k-1}\left(\frac{4\pi}{n+1-i}\right)^{\frac{1}{2}}\left(\frac{n+1-i}{2e}\right)^{\frac{n+1-i}{2}}}\mathrm{d}\mathbf{B}\right)\\
&= \mathcal{O}_k\left(\frac{n^{\frac{kn}{2}}n^{\frac{k}{2}}(\det\mathbf{B})^{\frac{n-k}{2}}e^{-\frac{n+1}{2}\,\mathrm{Tr}\,\mathbf{B}}}{2^{\frac{kn}{2}}n^{\frac{k}{2}}(2e)^{-\frac{nk}{2}}n^{\frac{2nk+3k-k^2}{4}}}\mathrm{d}\mathbf{B}\right)\\
&= \mathcal{O}_k\left(\frac{(\det\mathbf{B})^{\frac{n-k}{2}}e^{-\frac{n+1}{2}\,\mathrm{Tr}\,\mathbf{B}}}{e^{-\frac{nk}{2}}n^{\frac{3k-k^2}{4}}}\mathrm{d}\mathbf{B}\right)\\
&= \mathcal{O}_k\left(n^{-\frac{3}{4}k(k-1)}(\det\mathbf{B})^{\frac{n-k}{2}}e^{-\frac{n+1}{2}(\mathrm{Tr}\,\mathbf{B}-k)}\mathrm{d}\mathbf{B}\right)\\
&= \widetilde{\mathcal{O}}_k\left((\det\mathbf{B})^{\frac{n}{2}}e^{-\frac{n}{2}(\mathrm{Tr}\,\mathbf{B}-k)}\right).
\end{aligned}$$

The distribution of $\mathbf{B}_s$ is obtained from that of $\mathbf{B}$ by replacing $n+1$ by $n_s=\Theta(n)$. Since the $\mathbf{B}_s$ are independent, the distribution $\mu_{\mathbf{B}}$ of $\mathbf{B}_1,\ldots,\mathbf{B}_t$ is then given by

$$
\begin{aligned}
\mu_{\mathbf{B}} &= \prod_{s=1}^{t} \frac{n_s^{\frac{kn_s}{2}} (\det \mathbf{B}_s)^{\frac{n_s-k-1}{2}} e^{-\frac{n_s}{2}\operatorname{Tr}\mathbf{B}_s}}{2^{\frac{kn_s}{2}} \pi^{\frac{k(k-1)}{4}} \prod_{i=0}^{k-1} \Gamma(\frac{n_s-i}{2})} \mathrm{d}\mathbf{B}_s \\
&= \mathcal{O}_{k,t}\Big(n^{-\frac{3}{4}tk(k-1)} e^{-\sum_s \frac{n_s}{2}(\operatorname{Tr}\mathbf{B}_s - k)} \prod_s (\det \mathbf{B}_s)^{\frac{n_s-k-1}{2}} \mathrm{d}\mathbf{B}_s \\
&= \mathcal{O}_{k,t}\Big(n^{-\frac{3}{4}tk(k-1)} e^{\frac{n+1}{2}(\operatorname{Tr}\mathbf{B} - k)} \prod_s (\det \mathbf{B}_s)^{\frac{n_s-k-1}{2}} \mathrm{d}\mathbf{B}_s\Big) \\
&= \widetilde{\mathcal{O}}_{k,t}\Big(e^{-\frac{n+1}{2}(\operatorname{Tr}\mathbf{B} - k)} \prod_s (\det \mathbf{B}_s)^{\frac{n_s}{2}} \mathrm{d}\mathbf{B}_s\Big),
\end{aligned}
$$

where we used that $\sum_s n_s \mathbf{B}_s = (n+1)\mathbf{B}$. Now, the density depends exponentially on $\mathbf{B}_1, \ldots, \mathbf{B}_s$ in the form $\mu_{\mathbf{B}} = \widetilde{\mathcal{O}}(f(\mathbf{B}_1, \ldots, \mathbf{B}_s)^{\frac{n}{2}} \mathrm{d}\mathbf{B})$ for some function f. Since the set of $\mathbf{B}_1, \ldots, \mathbf{B}_s$ for given $\mathbf{B} = \sum_s \frac{n_s}{n+1} \mathbf{B}_s$ is compact, we deduce that the conditional distribution of $\mathbf{B}_1, \ldots, \mathbf{B}_s$, conditioned on $\mathbf{B}$, is concentrated where f attains its maximum.

To see that the unique maximum is attained at $\mathbf{B}_1 = \ldots = \mathbf{B}_t = \mathbf{B}$, we use that

$$
\tfrac{1}{n+1} \log\Big(\prod_s \det \mathbf{B}_s^{n_s}\Big) = \sum_s \tfrac{n_s}{n+1} \log\det \mathbf{B}_s \leq \log\det\Big(\sum_s \tfrac{n_s}{n+1} \mathbf{B}_s\Big) = \log\det \mathbf{B}
$$

by Lemma 3. Equality holds iff all $\mathbf{B}_s$ are equal. It follows that $\prod_s (\det \mathbf{B})^{\frac{n_s}{2}} \leq (\det \mathbf{B})^{\frac{n}{2}}$ with equality for $\mathbf{B}_1 = \ldots = \mathbf{B}_t$. This implies for the conditional probabilities, conditioned on fixed $\mathbf{B}$, that for any $\varepsilon' > 0$

$$
1 - \Pr\big[\mathbf{B}_s \approx_{\varepsilon'} \mathbf{B}\ \forall s \bigm| \mathbf{B} \text{ is fixed}\big] \leq 2^{-\gamma n} \tag{21}
$$

for some $\gamma > 0$. Note that γ may depend on $\varepsilon', \mathbf{B}, t$ and the values of $\lim_{n\to\infty} \frac{n_s}{n}$.

The result for conditioning on the configuration $C = \operatorname{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k), \boldsymbol{x}_i = \frac{\boldsymbol{y}_i}{\|\boldsymbol{y}_i\|}$ rather than conditioning on $\mathbf{B}$ can be deduced as follows: Since the lengths $\|\boldsymbol{y}_i\|$ are independent from C, we have for any $\varepsilon'' > 0$:

$$
\begin{aligned}
&1 - \Pr\big[C_s \approx_\varepsilon C, \mathrm{L}_s(\boldsymbol{x}_i) \approx_\varepsilon \tfrac{n_s}{n}\ \forall s, i \bigm| \operatorname{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = C\big] \\
\leq& \big(1 - \Pr\big[C_s \approx_\varepsilon C, \mathrm{L}_s(\boldsymbol{x}_i) \approx_\varepsilon \tfrac{n_s}{n}\ \forall s, i \bigm| \operatorname{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = C, \|\boldsymbol{y}_s\| \approx_{\varepsilon''} 1\ \forall s\big]\big) \\
&\cdot \Pr\big[\|\boldsymbol{y}_s\| \approx_{\varepsilon''} 1\ \forall s\big] + (1 - \Pr\big[\|\boldsymbol{y}_s\| \approx_{\varepsilon''} 1\ \forall s\big])
\end{aligned}
$$

Now, $(1 - \Pr\big[\|\boldsymbol{y}_s\| \approx_{\varepsilon''} 1\ \forall s\big])$ is exponentially small by Chernoff bounds (or tail bounds for the χ^2-distribution). By definition, $\|\boldsymbol{y}_i\| \approx_{\varepsilon''} 1$ implies $C \approx_{\varepsilon'} \mathbf{B}$ with $\varepsilon' \to 0$ as $\varepsilon'' \to 0$. Together with $\mathbf{B}_s \approx_{\varepsilon'} \mathbf{B}$ (cf. (21)), this implies $\mathbf{B}_s \approx_{2\varepsilon'} C$. Further, $\mathbf{B}_s \approx_{2\varepsilon'} C$ implies $\mathrm{L}_s(\boldsymbol{x}_i) \approx_\varepsilon \frac{n_s}{n}$ and $C_s \approx_\varepsilon C$ for all s, i with $\varepsilon \to 0$ as $\varepsilon' \to 0$. Consequently, Eq. (21) for conditioning on $\mathbf{B}$ implies $1 - \Pr\big[C_s \approx_\varepsilon C, \mathrm{L}_s(\boldsymbol{x}_i) \approx_\varepsilon \frac{n_s}{n}\ \forall s, i \bigm| \operatorname{Conf}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = C, \|\boldsymbol{y}_s\| \approx_\varepsilon 1\ \forall s\big] \leq 2^{-\gamma' n}$ for some $\gamma' > 0$. Note that we use (21) for several possible $\mathbf{B}$ with $C \approx_{\varepsilon'} \mathbf{B}$ and γ' is the minimum of γ in (21) over these choices of $\mathbf{B}$. Since γ can be made to depend continuously on ε' and $\mathbf{B} \approx_{\varepsilon'} C$ restricts $\mathbf{B}$ to a compact set, we know that $\gamma' > 0$, finishing the proof.

Sampling Points with Prescribed Configuration. Thm. 6 allows us to perform our configuration extension in a blockwise way. To write down our Algorithm ConfExt for configuration extension more concisely, we consider two sub-algorithms Trim and SamplePoints. The algorithm SamplePoints is used to sample points $\boldsymbol{w}_\ell$ on a block of n_s coordinates, subject to a prescribed (local) configuration wrt. the old points $(\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}$. We then build up the $\boldsymbol{v}_\ell$'s from such blocks $\boldsymbol{w}_\ell$. The algorithm Trim checks for consistency of the $\boldsymbol{w}_\ell$'s with the list elements from L_i on the s^{th} block of coordinates. The algorithms are detailed in Alg. 4 and Alg. 5. Formally, the properties of Trim and SamplePoints are as follows:

Proposition 1 (Properties of Trim). *Algorithm* $\mathsf{Trim}\big(\boldsymbol{n}, s, L, C, (\boldsymbol{w}_\ell)_{\ell\in I_{\mathrm{new}}}, \varepsilon\big)$ *described in Alg. 4 takes as input*

- *a vector* $\boldsymbol{n}$*, describing the decomposition* $n+1 = n_1 + \ldots + n_t$.
- *an index* $1 \le s \le t$ *describing the block we are considering.*
- *a list* $L \subset \mathsf{S}^n$.
- *target configuration* $C \in \mathbb{R}^{(1+m_{\mathrm{new}})\times(1+m_{\mathrm{new}})}$*. Wlog. consider* C *as indexed by* $\{1\} \cup I_{\mathrm{new}}$*, with* $\{1\}$ *corresponding to the elements from the list* L *and* $I_{\mathrm{new}} = \{2,\ldots,1+m_{\mathrm{new}}\}$ *corresponding to the* $\boldsymbol{w}_\ell$*'s.*
- m_{new} *points* $\boldsymbol{w}_\ell \in \mathbb{R}^{n_s}$, $\ell \in I_{\mathrm{new}}$*. Note that the points* $\boldsymbol{w}_\ell$ *are only defined on* n_s *coordinates.*
- *a measure of closeness* $\varepsilon > 0$.

It outputs $L' \subset L$ *consisting of all* $\boldsymbol{x} \in L$ *with* $\mathrm{Conf}_s\big(\boldsymbol{x}, (\boldsymbol{w}_\ell)_{\ell\in I_{\mathrm{new}}}\big) \approx_\varepsilon C$.

Here, $\mathrm{Conf}_s(\boldsymbol{x}, (\boldsymbol{w}_\ell)_{\ell\in I_{\mathrm{new}}})$ *means we take the* s^{th} *block of* $\boldsymbol{x}$ *and all of the* $\boldsymbol{w}_\ell$*'s (as the* $\boldsymbol{w}_\ell$ *only have* n_s *coordinates). For an exponentially large* $|L|$*, the running time and memory are both* $\widetilde{\mathcal{O}}(|L|)$.

Proof. The algorithm just makes a single pass over L. If L is exponential, the (polynomial) cost to check a single $\boldsymbol{x} \in L$ is hidden in the $\widetilde{\mathcal{O}}(.)$-notation.

Proposition 2 (Properties of SamplePoints).
The algorithm $\mathsf{SamplePoints}(\boldsymbol{n}, s, C, I_{\mathrm{old}}, I_{\mathrm{new}}, (\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}})$ *described in Alg. 5 takes as input*

- *a vector* $\boldsymbol{n}$*, describing the decomposition* $n+1 = n_1 + \ldots + n_t$.
- *an index* $1 \le s \le t$ *describing the block we are considering.*
- *non-singular target configuration* C*, indexed by* $I_{\mathrm{old}} \cup I_{\mathrm{new}}$.
- $m_{\mathrm{old}} = |I_{\mathrm{old}}|$ *old points* $\boldsymbol{v}_j \in \mathsf{S}^n$*, expected to satisfy* $\mathrm{Conf}_s((\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}) \approx C$.

It outputs $m_{\mathrm{new}} = |I_{\mathrm{new}}|$ *points* $\boldsymbol{w}_\ell \in \mathbb{R}^{n_s}$, $\ell \in I_{\mathrm{new}}$ *scaled to* $\|\boldsymbol{w}\|^2 = \frac{n_s}{n+1}$*. For any* $\varepsilon > 0$ *s.t. any symmetric* C' *with* $C' \approx_\varepsilon C$ *is still non-singular and provided* $\mathrm{Conf}_s((\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}) \approx_\varepsilon C$*, the output points satisfy* $\mathrm{Conf}_s((\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\mathrm{new}}}) \approx_\varepsilon C[I_{\mathrm{old}}, I_{\mathrm{new}}]$.
The distribution of the $\boldsymbol{w}_\ell$ *is uniform, conditioned on* $\mathrm{Conf}_s((\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\mathrm{new}}})$ *and the input points* $(\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}$*. The running time and memory usage of the algorithm* SamplePoints *is polynomial.*[6]

[6] As the algorithm is stated, the complexity is only polynomial in a model where we can compute with reals. More precisely, it also depends on the desired numerical precision. Note that the algorithm, as stated, chooses the new points such that the

Proof. We may sample the points $\boldsymbol{w}_\ell$ one by one. Previously sampled $\boldsymbol{w}_\ell$'s then take the roles of $\boldsymbol{v}_j$'s. Formally, we fill in the missing coordinates of $\boldsymbol{v}_j$ with zeros to make dimensions match; the algorithm will only consider the s^{th} block of $\boldsymbol{v}_j$'s anyway. This is done in lines 2–7 via recursion. So assume we are in the case where $I_{\text{new}} = \{\ell\}$ consists of a single element. We consider the s^{th} block $\boldsymbol{v}'_j$ of the $\boldsymbol{v}_j$. We claim that the algorithm samples $\boldsymbol{w}_\ell$ such that $\frac{\langle \boldsymbol{w}_\ell, \boldsymbol{v}'_j\rangle}{\|\boldsymbol{w}_\ell\|\|\boldsymbol{v}'_j\|} = C_{\ell,j}$ for all j. This implies that the output of SamplePoints satisfies $\operatorname{Conf}_s((\boldsymbol{v}_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}) \approx_\varepsilon C[I_{\text{old}}, I_{\text{new}}]$, where the ε-deviation actually comes purely from any deviation already present in the input.

For the claim that $\frac{\langle \boldsymbol{w}_\ell, \boldsymbol{v}'_j\rangle}{\|\boldsymbol{w}_\ell\|\|\boldsymbol{v}'_j\|} = C_{\ell,j}$, consider the subspace $V \subset \mathbb{R}^{n_s}$ spanned by the $\boldsymbol{v}'_j$'s. Under our assumptions, the Gram matrix of the $\boldsymbol{v}'_j$ is positive definite, hence V has dimension $|I_{\text{old}}|$. Let us split $\boldsymbol{w}_\ell$ as $\boldsymbol{w}_\ell = \boldsymbol{w}^\parallel + \boldsymbol{w}^\perp$, where $\boldsymbol{w}^\parallel \in V$ and $\boldsymbol{w}^\perp \in V^\perp$. The main observation is that (for $\|\boldsymbol{w}_\ell\|$ fixed) a configuration constraint on $\boldsymbol{w}_\ell$ of the given form is actually equivalent to prescribing both $\boldsymbol{w}^\parallel$ and $\|\boldsymbol{w}^\perp\|$. Indeed, writing $\boldsymbol{w}^\parallel = \sum_{j'} \alpha_{j'} \boldsymbol{v}'_{j'}$, the condition $\frac{\langle \boldsymbol{w}_\ell, \boldsymbol{v}'_j\rangle}{\|\boldsymbol{w}_\ell\|\|\boldsymbol{v}'_j\|} = C_{\ell,j}$ is equivalent to

$$\sum_{j'} \alpha_{j'} \langle \boldsymbol{v}'_{j'}, \boldsymbol{v}'_j \rangle = \|\boldsymbol{w}_\ell\|\|\boldsymbol{v}_j\| C_{\ell,j}.$$

This is exactly the system of linear equations we solve in line 10. To obtain $\boldsymbol{w}^\perp$, we sample a random direction $\boldsymbol{w}'^\perp$ orthogonal to V by orthogonally projecting a Gaussian. $\boldsymbol{w}^\perp$ is obtained by appropriate normalization.

Note that our algorithm ensures that the resulting configuration of all $|I_{\text{new}}|$ outputs satisfies $\operatorname{Conf}_s((\boldsymbol{v}_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\text{old}}}) = C'$, where C' is obtained from C by replacing $C[I_{\text{old}}]$ with the actual configuration of the $\boldsymbol{v}_j$'s. By assumption, C' is still positive definite. This ensures that the linear system actually has a (unique) solution and that the normalization of $\boldsymbol{w}^\perp$ works out (i.e. the argument of the square root is not negative): otherwise, this would show that a symmetric positive definite matrix with ones on the diagonal cannot appear as a non-singular configuration.

The fact that $\boldsymbol{w}'^\perp$ has uniform direction in $V^\perp$ gives the statement about the distribution of the output. The statement about the running time is clear (in the model where we can compute with reals and sample continuous Gaussians).

$\langle \boldsymbol{w}_\ell, \boldsymbol{w}_{\ell'}\rangle$ and the $\langle \boldsymbol{v}_j, \boldsymbol{w}_\ell\rangle$ *exactly* match the target configuration C. The ε-deviation of $\operatorname{Conf}_s((\boldsymbol{v}_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}})$ from C comes purely from the $\langle \boldsymbol{v}_j, \boldsymbol{v}_{j'}\rangle$-terms present in the input. If we are given ε as an argument (as one can do), then we may relax matching the target configuration exactly. Rather, we also allow an ε-deviation there. This allows to use finite-precision numerics, giving polynomial (in n) complexity for any fixed given $\varepsilon > 0$ and fixed non-singular C. For fixed C, the running time still stays polynomial if $\varepsilon \to 0$ sufficiently slowly.

Algorithm 4 Algorithm Trim

Input: $\boldsymbol{n}$ – decomposition of $n+1$. s – block we are working on.
L – input list. $C \in \mathbb{R}^{(1+m_{\text{new}})\times(1+m_{\text{new}})}$ – target configuration on n_s coordinates.
$(\boldsymbol{w}_\ell)_{\ell \in I_{\text{new}}}$ – $|I_{\text{new}}| = m_{\text{new}}$ new points, defined on n_s coordinates. ε – measure of closeness.

Output: $L' \subset L$ list of points locally compatible with C wrt. $\boldsymbol{w}_\ell$'s.

$L' := \emptyset$.
for all $\boldsymbol{x} \in L$ **do**
  if $|\text{Conf}_s(\boldsymbol{x}, \boldsymbol{w}_\ell) - C[1,\ell]| \leq \varepsilon$ for all $\ell \in I_{\text{new}}$ **then** ▷ $\boldsymbol{n}$ and s define Conf_s. We use
    $L' := L' \cup \{\boldsymbol{x}\}$. ▷ all n_s coordinates of the $\boldsymbol{w}_\ell$'s.
return L'

Algorithm 5 Algorithm SamplePoints

Input: $\boldsymbol{n}$ – decomposition of $n+1$. s – block we are working on.
C – target configuration, indexed by $I_{\text{old}} \cup I_{\text{new}}$.
$I_{\text{old}}, I_{\text{new}}$ – index sets with $|I_{\text{new}}| \geq 1$
$\boldsymbol{v}_j$ for $j \in I_{\text{old}}$ – $m_{\text{old}} = |I_{\text{old}}|$ old points.

Output: $\boldsymbol{w}_\ell \in \mathbb{R}^{n_s}$ for $\ell \in I_{\text{new}}$ – new points (with n_s coordinates) locally compatible with C.

function $\mathsf{SamplePoints}(\boldsymbol{n}, s, C, I_{\text{old}}, I_{\text{new}}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}})$
  if $|I_{\text{new}}| > 1$ **then** ▷ Sample points one by one
    Pick first element $\ell' \in I_{\text{new}}$.
    $\boldsymbol{w}_{\ell'} \leftarrow \mathsf{SamplePoints}(\boldsymbol{n}, s, C[I_{\text{old}} \cup \{\ell'\}], I_{\text{old}}, \{\ell'\}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}})$
    $I_{\text{old}} := I_{\text{old}} \cup \{\ell'\}$, $I_{\text{new}} := I_{\text{new}} \setminus \{\ell'\}$.
    Set $\boldsymbol{v}_{\ell'} \in \mathbb{R}^{n+1}$ such that its s^{th} block is $\boldsymbol{w}_{\ell'}$ (by extending with 0's).
    return $\boldsymbol{w}_{\ell'}, \mathsf{SamplePoints}(\boldsymbol{n}, s, C, I_{\text{old}}, I_{\text{new}}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}})$
  else ▷ $|I_{\text{new}}| = 1$, sample single point $\boldsymbol{w}$
    Denote $\{\ell\} := I_{\text{new}}$, let $\boldsymbol{v}'_j \in \mathbb{R}^{n_s}$ be the s^{th} block of $\boldsymbol{v}_j$ for all $j \in I_{\text{old}}$.
    Obtain α_j's by solving $\sum_{j\in I_{\text{old}}} \langle \boldsymbol{v}'_j, \boldsymbol{v}'_{j'} \rangle \alpha_j = \sqrt{\frac{n_s}{n+1}} \|\boldsymbol{v}'_{j'}\| C_{\ell,j'}$ for all $j' \in I_{\text{old}}$.
    Set $\boldsymbol{w}^{\parallel} := \sum_j \alpha_{j\in I_{\text{old}}} \boldsymbol{v}'_j$
    Sample $\boldsymbol{w}' \in \mathbb{R}^{n_s}$ Gaussian
    Project $\boldsymbol{w}'$ to $\boldsymbol{w}'^{\perp} \in \text{Span}\big((\boldsymbol{v}_j)_{j\in I_{\text{old}}}\big)^{\perp}$ ▷ via Gram-Schmidt-Orthogonalization
    Set $\boldsymbol{w}^{\perp} := \sqrt{\frac{n_s}{n+1} - \|\boldsymbol{w}^{\parallel}\|^2} \frac{\boldsymbol{w}'^{\perp}}{\|\boldsymbol{w}'^{\perp}\|}$. ▷ of the ordered set $\{(\boldsymbol{v}_j)_{j\in I_{\text{old}}}, \boldsymbol{w}'\}$
    return $\boldsymbol{w}^{\parallel} + \boldsymbol{w}^{\perp}$.

A.4 Our ConfExt Algorithm

We use algorithms SamplePoints and Trim to describe our algorithm ConfExt, given in Alg. 6.

The properties of ConfExt are described in Thm. 7.

Theorem 7 (Configuration Extension). *Let $\varepsilon > 0$, $C, I_{\mathrm{new}}, I_{\mathrm{old}}, k$ and $0 < c_s < 1$ be fixed. Assume that any configuration C' with $C' \approx_{2\varepsilon} C$ is non-singular. Assume that* ConfExt *chooses ρ and t independent from n and its choice of decomposition $n+1 = n_1 + \ldots + n_t$ is such that $\frac{n_s}{n+1} \xrightarrow{n\to\infty} c_s$.*

Let $\boldsymbol{v}_j \in \mathsf{S}^n$ for $j \in I_{\mathrm{old}}$ be points with $\mathrm{Conf}((\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}) \approx_\varepsilon C[I_{\mathrm{old}}]$. Let L_i be lists of exponential size for $1 \le i \le k$ whose elements $\boldsymbol{x}_i \in L_i$ satisfy $\mathrm{Conf}(\boldsymbol{x}_i, (\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}) \approx_\varepsilon C[i, I_{\mathrm{old}}]$.

Then ConfExt*, run with these parameters satisfies the following:*

- ***Memory:*** *The memory required by* ConfExt *is $\widetilde{\mathcal{O}}(\max |L_i|)$.*
- ***#Instances:*** *The number of instances output is $\widetilde{\mathcal{O}}(\rho^n)$.*
- ***Correctness:*** *For every instance $(\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell\in I_{\mathrm{new}}}, L_1'^{(\boldsymbol{r})}, \ldots, L_k'^{(\boldsymbol{r})}$ that we output and any $\boldsymbol{x}_i \in L_i'^{(\boldsymbol{r})}$, we have $\mathrm{Conf}\big(\boldsymbol{x}_i, (\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}, (\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell\in I_{\mathrm{new}}}\big) \approx_\varepsilon C[i, I_{\mathrm{old}}, I_{\mathrm{new}}]$.*
- ***Output distribution:*** *For any individual instance output, the joint distribution of the $(\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell\in I_{\mathrm{new}}}$ point is uniform, conditioned on the configuration $\mathrm{Conf}((\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}, (\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell\in I_{\mathrm{new}}})$. Note that we do not have (or need) independence for different $\boldsymbol{r}$'s.*
- ***List sizes:*** *The expected size of $L_i'^{(\boldsymbol{r})}$ in output instances can be bounded by*

$$\mathbb{E}[|L_i'^{(\boldsymbol{r})}|] = |L_i| \cdot \widetilde{\mathcal{O}}(\kappa_i^n) \text{ with}$$
$$\kappa_i = \max_{C'\approx_{2\varepsilon} C}\left(\frac{\det(C'[i, I_{\mathrm{old}}, I_{\mathrm{new}}]) \cdot \det(C'[I_{\mathrm{old}}])}{\det(C'[I_{\mathrm{old}}, I_{\mathrm{new}}]) \cdot \det(C'[i, I_{\mathrm{old}}])}\right)^{1/2}.$$

- ***Concentration:*** *With probability exponentially close to 1, the length of every list whose length is exponential is at most twice its expected value.*
- ***Solution preservation:*** *Let*

$$\rho' := \max_{C'\approx_\varepsilon C}\left(\frac{\det C'[I_{\mathrm{lists}}, I_{\mathrm{old}}] \det C'[I_{\mathrm{old}}, I_{\mathrm{new}}]}{\det C'[I_{\mathrm{lists}}, I_{\mathrm{old}}, I_{\mathrm{new}}] \det C'[I_{\mathrm{old}}]}\right)^{\frac{1}{2}}$$

  Call a k-tuple $(\boldsymbol{x}_i)_{i\in I_{\mathrm{lists}}}$ a solution if $\mathrm{Conf}((\boldsymbol{x}_i)_{i\in I_{\mathrm{lists}}}, (\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}}) \approx_\varepsilon C[I_{\mathrm{lists}}, I_{\mathrm{old}}]$. If $\rho > \rho'$, then with overwhelming probability, for every solution $(\boldsymbol{x}_i)_{i\in I_{\mathrm{lists}}}$, there exists an instance $r_0, \ldots, r_t$ such that the solution is still present in the output: $\boldsymbol{x}_i \in L_i'^{(r_0,\ldots,r_t)}$.
- ***Running time:*** *If $\rho > \rho'$ and if all $|L_i| \cdot \kappa_i^n$ are exponentially increasing, the expected running time T of* ConfExt *can be upper bounded by*

$$T = \widetilde{\mathcal{O}}\big(\rho^n \max_s \rho^{n_s} \max_i |L_i|\, \kappa_i^n\big).$$

Note that in cases where the algorithm does not output anything for a given $\boldsymbol{r}$ due to lines 7 or 9, we may treat $L_i'^{(\boldsymbol{r})}$ as the empty set. Further, note that the solution

Algorithm 6 Algorithm $\mathsf{ConfExt}$ for configuration extension

Input: k – number of lists. m_{old} – number of old $\boldsymbol{v}$'s. m_{new} – number of new $\boldsymbol{v}$'s.
$C \in \mathbb{R}^{(k+m_{\text{old}}+m_{\text{new}})\times(k+m_{\text{old}}+m_{\text{new}})}$ – configuration.
(Notationally, we consider index sets $I_{\text{new}}, I_{\text{old}}, I_{\text{lists}}$ to index into C.)
L_i for $i \in I_{\text{lists}}$ – k lists of points from S^n.
($\boldsymbol{x}_i \in L_i$ are promised to satisfy $\langle \boldsymbol{x}_i\, , \boldsymbol{v}_j\rangle \approx_\varepsilon C[i,j]$ for $\boldsymbol{x}_i \in L_i, j \in I_{\text{old}}$.)
$\boldsymbol{v}_j$ for $j \in I_{\text{old}}$ – old $\boldsymbol{v}$'s , promised to satisfy $\mathrm{Conf}((\boldsymbol{v}_j)_{j\in I_{\text{old}}}) \approx_\varepsilon C[I_{\text{old}}]$.
$\varepsilon > 0$ – measure of closeness.

Output: $\boldsymbol{v}_\ell^{(\boldsymbol{r})}$ for $\ell \in I_{\text{new}}$ and $R = \widetilde{\Theta}(\rho^n)$ many different $\boldsymbol{r}$'s – new $\boldsymbol{v}$'s
$L_i'^{(\boldsymbol{r})}$ for $i \in I_{\text{lists}}$ – shrunk lists.
(Output is streamed for one value of $\boldsymbol{r}$ after another.)

Choose ρ and a decomposition $n+1 = n_1 + \ldots + n_t$. ▷ See analysis
Set $R_s \leftarrow \lceil \rho^{n_s} \rceil$ for $s \in \{1,\ldots,t\}$ ▷ Number of Repetitions per block
for r_0 from 1 to n^2 **do** ▷ Outermost loop. Its purpose is to rerandomize
    Sample random rotation $\mathbf{A} \in \mathbb{O}_{n+1}$. ▷ for global-local transitions.
    Apply $\mathbf{A}$ to all elements in each L_i to obtain $L_i'^{(r_0)}$.
    Apply $\mathbf{A}$ to all $\boldsymbol{v}_j$ to obtain $\boldsymbol{v}_j'$ for all $j \in I_{\text{old}}$.
    if $\left|\|\boldsymbol{v}'_{j,s}\| - \sqrt{\frac{n_s}{n+1}}\right| > \frac{\varepsilon}{\sqrt{t}}$ for any $j \in I_{\text{old}}, 1 \le s \le t$ **then** ▷ Bad lengths
        continue; (as in the C language)
    if $\mathrm{Conf}_s((\boldsymbol{v}_j')_{j\in I_{\text{old}}}) \not\approx_{2\varepsilon} C[I_{\text{old}}]$ for some $1 \le s \le t$ **then** ▷ Bad local config
        continue;
    Remove any $\boldsymbol{x}_i \in L_i'^{(r_0)}$ from $L_i'^{(r_0)}$ with $\mathrm{Conf}_s(\boldsymbol{x}_i, (\boldsymbol{v}_j')_{j\in I_{\text{old}}}) \not\approx_{2\varepsilon} C[i, I_{\text{old}}]$.
    For each $\ell \in I_{\text{new}}$, initialize $\boldsymbol{v}_\ell^{(r_0)} := () \in \mathbb{R}^0$ with an empty list.
    for r_1 from 1 to R_1 **do**
        $(\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}} \leftarrow \mathsf{SamplePoints}(\boldsymbol{n}, 1, C, I_{\text{old}}, I_{\text{new}}, (\boldsymbol{v}_j')_{j\in I_{\text{old}}})$.
        Let $\boldsymbol{v}_\ell^{(r_0,r_1)}$ be the concatenation of $\boldsymbol{v}_\ell^{(r_0)}$ and $\boldsymbol{w}_\ell$ for all ℓ ▷ Note: $\boldsymbol{v}^{(r_0)}$ is empty.
        $L_i'^{(r_0,r_1)} \leftarrow \mathsf{Trim}(\boldsymbol{n}, 1, L_i'^{(r_0)}, C[i, I_{\text{new}}], (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}, 2\varepsilon)$ for each i.
        for r_2 from 1 to R_2 **do**
            $(\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}} \leftarrow \mathsf{SamplePoints}(\boldsymbol{n}, 2, C, I_{\text{old}}, I_{\text{new}}, (\boldsymbol{v}_j')_{j\in I_{\text{old}}})$.
            Let $\boldsymbol{v}_\ell^{(r_0,r_1,r_2)}$ be the concatenation of $\boldsymbol{v}_\ell^{(r_0,r_1)}$ and $\boldsymbol{w}_\ell$ for all ℓ
            $L_i'^{(r_0,r_1,r_2)} \leftarrow \mathsf{Trim}(\boldsymbol{n}, 2, L_i'^{(r_0,r_1)}, C[i, I_{\text{new}}], (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}, 2\varepsilon)$ for each i.
            for r_3 from 1 to R_3 **do**
                ...
                for r_t from 1 to R_t **do**
                    $(\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}} \leftarrow \mathsf{SamplePoints}(\boldsymbol{n}, t, C, I_{\text{old}}, I_{\text{new}}, (\boldsymbol{v}_j')_{j\in I_{\text{old}}})$.
                    Let $\boldsymbol{v}_\ell^{(r_0,\ldots,r_t)}$ be the concatenation of $\boldsymbol{v}_\ell^{(r_0,\ldots,r_{t-1})}$ and $\boldsymbol{w}_\ell$.
                    $L_i'^{(r_0,\ldots,r_t)} \leftarrow \mathsf{Trim}(\boldsymbol{n}, t, L_i'^{(r_0,\ldots,r_{t-1})}, C[i, I_{\text{new}}], (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}, 2\varepsilon)$.
                    Apply $\mathbf{A}^{-1}$ to all elements from $L_i'^{(r_0,\ldots,r_t)}$ for each i. ▷ Undo rerandomization
                    Apply $\mathbf{A}^{-1}$ to $\boldsymbol{v}_\ell^{(r_0,\ldots,r_t)}$ for each $\ell \in I_{\text{new}}$.
                    Remove all elements $\boldsymbol{x}_i \in L'^{(r_0,\ldots,r_t)}$ that do not satisfy
                        $\left|\langle \boldsymbol{x}_i\, , \boldsymbol{v}_\ell^{(r_0,\ldots,r_t)}\rangle - C_{i,\ell}\right| \le \varepsilon$
                    Output the instance consisting of new points $(\boldsymbol{v}_\ell^{(r_0,\ldots,r_t)})_{\ell\in I_{\text{new}}}$ and lists $L_1'^{(r_0,\ldots,r_t)}, \ldots, L_k'^{(r_0,\ldots,r_t)}$.
End Algorithm

preservation property actually holds irrespective of the actual distribution on the L_i's (still, we need that the elements satisfy the configuration constraints).

Before we give the proof of Thm. 7, let us discuss how it implies Thm. 5 from Sect. 6 and why we care about the fact that the Solution Preservation property holds irrespective of the input distribution.

The analysis above is for $t, \varepsilon, \rho, \boldsymbol{n}$ fixed. We may assume that the algorithm chooses $\rho \approx_\varepsilon \rho'$. If we let $\varepsilon \to 0$ sufficiently slowly and $t \to \infty$ sufficiently slowly, Thm. 7 implies as a corollary Thm. 5, which we repeat here:

Corollary 4 (Thm. 5). *Use notation as in Def. 4. Assume that $C, k, m_{\text{old}}, m_{\text{new}}$ do not depend on n. Then our algorithm* ConfExt *outputs*

$$R = \widetilde{\mathcal{O}}\left(\frac{\det(C[I_{\text{old}}, I_{\text{new}}]) \cdot \det(C[I_{\text{lists}}, I_{\text{old}}])}{\det(C[I_{\text{lists}}, I_{\text{old}}, I_{\text{new}}]) \cdot \det(C[I_{\text{old}}])}\right)^{\frac{n}{2}} \tag{22}$$

instances. In each output instance, the new points $(\boldsymbol{v}_\ell)_{\ell \in I_{\text{new}}}$ are chosen uniformly (conditioned on the constraints). Consider solution k-tuples, i.e. $\boldsymbol{x}_i \in L_i$ with $\mathrm{Conf}((\boldsymbol{x}_i)_{i \in I_{\text{lists}}}) \approx C[I_{\text{lists}}]$. With overwhelming probability, for every solution k-tuple $(\boldsymbol{x}_i)_{i \in I_{\text{lists}}}$, there exists at least one instance such that all $\boldsymbol{x}_i \in L'_i$ for this instance, so we retain all solutions. Assume further that the elements from the input lists L_i, $i \in I_{\text{lists}}$ are iid uniformly distributed (conditioned on the configuration $\mathrm{Conf}(\boldsymbol{x}_i, (\boldsymbol{v}_j)_{j \in I_{\text{old}}})$ for $\boldsymbol{x}_i \in L_i$, which is assumed to be compatible with C). Then the expected size of the output lists per instance is given by

$$\mathbb{E}[|L'_i|] = |L_i| \cdot \widetilde{\mathcal{O}}\left(\left(\frac{\det(C[i, I_{\text{old}}, I_{\text{new}}]) \cdot \det(C[I_{\text{old}}])}{\det(C[I_{\text{old}}, I_{\text{new}}]) \cdot \det(C[i, I_{\text{old}}])}\right)^{n/2}\right).$$

Assuming all these expected list sizes are exponentially increasing in n, the running time of the algorithm is given by $\widetilde{\mathcal{O}}(R \cdot \max_i \mathbb{E}[|L'_i|])$ (essentially the size of the output) and the memory complexity is given by $\widetilde{\mathcal{O}}(\max_i |L_i|)$ (essentially the size of the input).

Proof. We assume $\rho \approx \rho'$. Set $c_s = \frac{1}{t}$, i.e. $\frac{n_s}{n+1} \to \frac{1}{t}$. Then this is a simple consequence of letting $\varepsilon \to 0$ and $t \to \infty$ sufficiently slowly: ρ' and κ_i converge to the correct terms. For the running time, note that $\max_s \rho^{n_s} = \sqrt[t]{\rho}^n$ and this contribution can be absorbed in the $\widetilde{\mathcal{O}}$-notation. So the running time becomes $\widetilde{\mathcal{O}}(R \cdot \max_i |L_i| \, \kappa_i^n)$ with $\kappa_i^n \, |L_i| = \widetilde{\mathcal{O}}(\mathbb{E}[L'_i])$.

Remark 3 (Correctness of Composition). The solution preservation property in Thm. 7 holds irrespective of the actual distribution of the inputs. This has the following consequence: consider an output instance $(\boldsymbol{v}_\ell)_{\ell \in I_{\text{new}}}$ with shrunk lists L'_i for $i \in I_{\text{lists}}$. Our definition of configuration extension tell us that all $\boldsymbol{x}'_i \in L'_i$ satisfy some configuration constraints wrt. $\boldsymbol{v}_\ell$, given by

$$\mathrm{Conf}(\boldsymbol{x}_i, (\boldsymbol{v}_\ell)_{\ell \in I_{\text{new}}}) \approx C[i, I_{\text{new}}]. \tag{23}$$

But the definition does not tell us that L'_i actually contains all such $\boldsymbol{x}_i \in L_i$ that satisfy the constraint (23). In particular, the theorem gives us no guarantee for the distribution on the L'_i, which might be input to the next algorithm. However, this is not a problem: The reason is that our algorithms still work in this case (this includes Alg. 1). We only need the distribution on the input for statements about the running time. However, the running time can only get better if we input a sublist. Of course, the list size of the output L'_i is (for $\varepsilon \to 0$, in leading order) the same as the size of all $\boldsymbol{x}_i \in L_i$ that satisfy (23), so we do not really "gain" anything here. Essentially, this means that we may pretend that the distribution on the outputs lists is as if we would output all solutions of (23).

We now prove Thm. 7.

Proof. We will prove the items in the order they appear in the theorem's statement, starting with the memory requirement:

Memory. Due to the fact that lists are only ever shrunk, the algorithm can clearly be implemented in $\widetilde{\mathcal{O}}(\max|L_i|)$ memory: note that we only need to store $\boldsymbol{v}_\ell^{(r'_0,\dots)}$ and $L_i'^{(r'_0,\dots)}$ for $(r'_0,\dots)$ being a prefix of the current $r_0,\dots$ under consideration.

Number of Instances. The statement about the number of instances is obvious.

Correctness. Let us turn our attention towards the correctness statement. We need to show $\mathrm{Conf}\big(\boldsymbol{x}_i,(\boldsymbol{v}_j)_{j\in I_{\mathrm{old}}},(\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell\in I_{\mathrm{new}}}\big) \approx_\varepsilon C[i,I_{\mathrm{old}},I_{\mathrm{new}}]$. Note that this is a statement about scalar products between vectors, where each vector is either $\boldsymbol{x}_i$, one of the old $\boldsymbol{v}_j$'s or one of the new $\boldsymbol{v}_\ell^{(\boldsymbol{r})}$'s. This means that we need to consider 6 different types of pairs. If both vectors come from the input of the algorithm, there is nothing to show. For $\langle \boldsymbol{x}_i\,,\boldsymbol{v}_\ell^{(\boldsymbol{r})}\rangle$, line 29 ensures correctness. So we need to consider $\langle \boldsymbol{v}_\ell^{(\boldsymbol{r})}\,,\boldsymbol{v}_{\ell'}^{(\boldsymbol{r})}\rangle$ and $\langle \boldsymbol{v}_\ell^{(\boldsymbol{r})}\,,\boldsymbol{v}_j\rangle$. We have

$$\begin{aligned}\langle \boldsymbol{v}_\ell^{(\boldsymbol{r})}\,,\boldsymbol{v}_{\ell'}^{(\boldsymbol{r})}\rangle &= \sum_s \|\boldsymbol{v}_{\ell,s}^{(\boldsymbol{r})}\|\cdot\|\boldsymbol{v}_{\ell',s}^{(\boldsymbol{r})}\|\cdot\frac{\langle \boldsymbol{v}_{\ell,s}^{(\boldsymbol{r})}\,,\boldsymbol{v}_{\ell',s}^{(\boldsymbol{r})}\rangle}{\|\boldsymbol{v}_{\ell,s}^{(\boldsymbol{r})}\|\cdot\|\boldsymbol{v}_{\ell',s}^{(\boldsymbol{r})}\|}\\ &= \sum_s \sqrt{\tfrac{n_s}{n+1}}\cdot\sqrt{\tfrac{n_s}{n+1}}\cdot C_{\ell,\ell'} = C_{\ell,\ell'}\end{aligned}$$

by the properties of SamplePoints. We use here the fact that $\boldsymbol{w}_\ell$ sampled using SamplePoints have fixed square-length distribution and will satisfy the configuration constraints involving newly sampled points exactly. Similarly, we obtain

$$\langle \boldsymbol{v}_\ell^{(\boldsymbol{r})}\,,\boldsymbol{v}_j\rangle = \sum_s \|\boldsymbol{v}'_j\|\cdot\sqrt{\tfrac{n_s}{n+1}}\cdot C_{\ell,j}.$$

Consequently, we have

$$\Big|\langle \boldsymbol{v}_\ell^{(\boldsymbol{r})}\,,\boldsymbol{v}'_j\rangle - C_{\ell,j}\Big| = \tfrac{|C_{\ell,j}|}{\sqrt{n+1}}\Big|\sum_s \sqrt{n_s}\Big(\|\boldsymbol{v}'_j\| - \sqrt{\tfrac{n_s}{n+1}}\Big)\Big| \le \tfrac{|C_{\ell,j}|}{\sqrt{n+1}}\cdot\tfrac{\varepsilon}{\sqrt{t}}\cdot\sum_s\sqrt{n_s}.$$

Since $\sqrt{\cdot}$ is concave, we have $\frac{\sum_s\sqrt{n_s}}{t} \le \sqrt{\frac{\sum n_s}{t}} = \frac{\sqrt{n+1}}{\sqrt{t}}$. Plugging this in gives $\big|\langle \boldsymbol{v}_\ell^{(\boldsymbol{r})}\,,\boldsymbol{v}'_j\rangle - C_{\ell,j}\big| \le |C_{\ell,j}|\cdot\varepsilon \le \varepsilon$.

Output Distribution. The statement about the distribution of $(\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell \in I_{\text{new}}}$ in individual instances follows easily from the rerandomization performed by the algorithm: consider any $\mathbf{B} \in \mathbb{O}_{n+1}$ that fixes all $\boldsymbol{v}_j$'s from the input. Since $\mathbf{A}$ follows the same distribution as $\mathbf{AB}^{-1}$, we may replace $\mathbf{A}$ by $\mathbf{AB}^{-1}$ everywhere in the algorithm without changing the distribution of its output. Note that the distribution of $(\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell \in I_{\text{new}}}$ only depends on the $\boldsymbol{v}_j$'s. Going through the algorithm, we see that the distribution of $(\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell \in I_{\text{new}}}$ on input $(\boldsymbol{v}_j)_{j \in I_{\text{old}}}$ is the same as the distribution of $(\mathbf{B}\boldsymbol{v}_\ell^{(\boldsymbol{r})})_{\ell \in I_{\text{new}}}$ on input $(\mathbf{B}\boldsymbol{v}_j)_{j \in I_{\text{old}}}$. Since $(\mathbf{B}\boldsymbol{v}_j)_{j \in I_{\text{old}}} = (\boldsymbol{v}_j)_{j \in I_{\text{old}}}$, this shows the claim (cf. the discussion about conditional probabilities in Section A.2 above).

List Sizes. Now consider the statement about the expected list sizes. Consider a block $1 \leq s \leq t$ and the trimming done in the r_s-loop. For any $\boldsymbol{x}_i \in L_i$, and hence for any $\boldsymbol{x}_i \in L_i'^{(r_0,\ldots,r_{s-1})}$, the local configuration $\text{Conf}_s(\boldsymbol{x}_i, (\boldsymbol{v}_j')_{j \in I_{\text{old}}})$ is 2ε-close to the target configuration due to lines 7–11. Furthermore, conditioned on their local configuration, the distribution of the (direction of) $(\boldsymbol{v}_{j,s}')_{j \in I_{\text{old}}}$ is uniform, since all previous operations only depended on this block of coordinates via possibly their configuration or length. This includes any previous trimmings. Since $\boldsymbol{w}_\ell$'s choice also only depends on the configuration of $(\boldsymbol{v}_{j,s}')_{j \in I_{\text{old}}}$, this implies that we may use Lemma 2 to compute the factor by which the lists shrink in each trimming. We directly see that the factor is bounded by $\kappa_i^{n_s}$, from which the result follows immediately. Note that the fact that we only know $\text{Conf}_s(\boldsymbol{x}_i, (\boldsymbol{v}_j')_{j \in I_{\text{old}}})$ approximately is why we maximize over $C' \approx_{2\varepsilon} C$ in the definition of κ_i in the theorem's statement. Also, we ignore any shrinking of lists coming from line 11 and line 29, as that can only make the lists shorter.

Concentration. Consider the concentration results on the expected value next. Consider any list $L_i'^{(r_0,\ldots,r_s)}$. Whether an input element $\boldsymbol{x} \in L_i$ is contained in the list only depends on $\mathbf{A}$ and $(\boldsymbol{v}_\ell^{(r_0,\ldots,r_s)})_{\ell \in I_{\text{new}}}$. In particular, conditioned on $(\boldsymbol{v}_j)_{j \in I_{\text{old}}}$ and the choices of $\mathbf{A}$ and $(\boldsymbol{v}_\ell^{(r_0,\ldots,r_s)})_{\ell \in I_{\text{new}}}$, we know that whether any $\boldsymbol{x}_i \in L_i$ are contained in $L_i'^{(r_0,\ldots,r_s)}$ is independent (for different choices of $\boldsymbol{x}_i \in L_i$). Using the randomness in the $\boldsymbol{x}_i \in L_i$ input to $\mathsf{ConfExt}$, this allows us to use Chernoff bounds to deduce that the probability that any list size is twice are large as expected becomes superexponentially small if the expected list size is exponential. Using a union bound over the ("only" exponentially many) choices of lists $L_i'^{(r_0,\ldots,r_s)}$ then shows the claim.

Solution Preservation. Let us now consider the solution preservation property. We will show that for any given solution $(\boldsymbol{x}_i)_{i \in I_{\text{lists}}}$ from $L_1 \times \ldots L_k$ and any given run of the outermost loop (over r_0), with at least constant probability p, there exists some $r_1, \ldots, r_t$ such that $(\boldsymbol{x}_i)_{i \in I_{\text{lists}}}$ is retained in the instance indexed by $r_0, \ldots, r_t$. For given $(\boldsymbol{x}_i)_{i \in I_{\text{lists}}}$ and $(\boldsymbol{v}_j)_{j \in I_{\text{old}}}$, the above property is independent for different runs of the outermost loop, since it only depends on the configuration $\text{Conf}((\boldsymbol{x}_i)_{i \in I_{\text{lists}}}, (\boldsymbol{v}_j)_{j \in I_{\text{old}}})$. Consequently, due to the n^2-fold repetition of the outermost loop, the probability that a given solution $(\boldsymbol{x}_i)_{i\in I_{\text{lists}}}$ is not retained in any instance has probability at most $(1-p)^{n^2}$, which is superexponentially small. By a union bound over the at most exponentially many solutions, the claim follows. So it suffices to show that for any given $(\boldsymbol{x}_i)_{i\in I_{\text{lists}}}$ and r_0, we retain the solution with at least constant probability.

Choose some $0<\varepsilon'$ sufficiently small. The exact conditions on ε' (depending on the parameters $\varepsilon, C, t, c_1, \ldots, c_t$) can be determined from the proof. Denote by C' the configuration obtained from C by replacing the $[I_{\text{lists}}, I_{\text{old}}]$-part with the true configuration $\operatorname{Conf}\big((\boldsymbol{x}_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}_j)_{j\in I_{\text{old}}}\big)$. The fact that the $\boldsymbol{x}_i$ are a solution precisely means that $C'[I_{\text{lists}}, I_{\text{old}}] \approx_\varepsilon C[I_{\text{lists}}, I_{\text{old}}]$.

Let $\boldsymbol{x}'_i := \mathbf{A}\boldsymbol{x}_i$ be our fixed solution after rotating by the random $\mathbf{A}$. We denote by C'_s the result of replacing the $[I_{\text{lists}}, I_{\text{old}}]$-part of C by the local configuration $\operatorname{Conf}_s\big((\boldsymbol{x}'_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}'_j)_{j\in I_{\text{old}}}\big)$ after rotation.

By Thm. 6, for any fixed ε', we simultaneously have $\big|\|\boldsymbol{v}'_{j,s}\| - \sqrt{\frac{n_s}{n+1}}\,\big| \le \varepsilon'$ for all s, j and $C'_s[I_{\text{lists}}, I_{\text{old}}] \approx_{\varepsilon'} C'[I_{\text{lists}}, I_{\text{old}}]$ for all s with overwhelming probability over the choice of $\mathbf{A} \in \mathbb{O}_n$. In particular, by choosing $\varepsilon' < \frac{\varepsilon}{\sqrt{t}}$, our solution survives line 7 with overwhelming probability. For lines 9 and 11, we get $\operatorname{Conf}_s(\boldsymbol{x}'_i, (\boldsymbol{v}'_j)_{j\in I_{\text{old}}}) \approx_{\varepsilon+\varepsilon'} C[i, I_{\text{old}}]$, which means that the solution survives these checks as well, since $\varepsilon+\varepsilon' < 2\varepsilon$.

We next show that for any block s, if the solution is still present in the $L_i'^{(r_0,\ldots,r_{s-1})}$, then with constant probability, there exists some r_s such that the solution is present in the $L_i'^{(r_0,\ldots,r_s)}$.

For this, consider a given iteration of the r_s-loop and the choice of $\boldsymbol{w}_\ell$. SamplePoints gives us uniform $(\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}$ with $\operatorname{Conf}_s\big((\boldsymbol{v}'_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}\big) = C'_s[I_{\text{old}}, I_{\text{new}}]$. Note that we actually have equality, because the configuration of the new points is enforced exactly. In particular, we can use Lemma 2 and obtain for any $\varepsilon' > 0$ that

$$\begin{aligned}&\Pr\Big[\operatorname{Conf}_s\big((\boldsymbol{x}'_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}'_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}\big) \approx_{\varepsilon'} C'_s\Big]\\ =&\widetilde{\Omega}\Big(\Big(\frac{\det C'_s[I_{\text{lists}}, I_{\text{old}}, I_{\text{new}}]\det C'_s[I_{\text{old}}]}{\det C'_s[I_{\text{lists}}, I_{\text{new}}]\det C'_s[I_{\text{old}}, I_{\text{new}}]}\Big)^{\frac{n_s}{2}}\Big).\end{aligned}$$

In particular, with $\rho > \rho'$, we ensure that the number of iterations R_s is larger than the inverse of that probability. This shows that with high probability, there exists at least one iteration of the r_s loop where we have $\operatorname{Conf}_s\big((\boldsymbol{x}'_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}'_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_\ell)_{\ell\in I_{\text{new}}}\big) \approx_{\varepsilon'} C'_s$. The latter readily implies that the solution survives the r_s-loop.

The only thing left to verify is that the solution survives the final check in line 29. In this final check, we verify that $|\langle \boldsymbol{x}_i\,, \boldsymbol{v}_\ell^{(r_0,\ldots,r_s)}\rangle - C_{i,\ell}| \le \varepsilon$, where $\boldsymbol{v}_\ell^{(r_0,\ldots,r_s)}$ is composed out of the individual $\boldsymbol{w}_\ell$ sampled in each loop (up to undoing the rotation $\mathbf{A}$). Denote by $\boldsymbol{w}_{\ell,s}$ the value of $\boldsymbol{w}_\ell$ that was sampled in the r_s-loop and that satisfies the guarantee

$$\operatorname{Conf}_s\big((\boldsymbol{x}'_i)_{i\in I_{\text{lists}}}, (\boldsymbol{v}'_j)_{j\in I_{\text{old}}}, (\boldsymbol{w}_{\ell,s})_{\ell\in I_{\text{new}}}\big) \approx_{\varepsilon'} C'_s \tag{24}$$

from the argument above. Since the rotation $\mathbf{A}$ does not change the scalar product, we have

$$\left\langle \boldsymbol{x}_i \,, \boldsymbol{v}_\ell^{(r_0,\ldots,r_s)} \right\rangle = \sum_s \left\langle \boldsymbol{x}'_{i,s} \,, \boldsymbol{w}_{\ell,s} \right\rangle = \sum_s \sqrt{\tfrac{n_s}{n+1}} \cdot \|\boldsymbol{x}'_{i,s}\| \cdot \frac{\langle \boldsymbol{x}'_{i,s} \,, \boldsymbol{w}_{\ell,s} \rangle}{\|\boldsymbol{x}'_{i,s}\| \cdot \|\boldsymbol{w}_{\ell,s}\|}.$$

Now, our guarantee (24) ensures that $\frac{\langle \boldsymbol{x}'_{i,s}, \boldsymbol{w}_{\ell,s} \rangle}{\|\boldsymbol{x}'_{i,s}\| \cdot \|\boldsymbol{w}_{\ell,s}\|}$ deviates from the $(i,\ell)^{\text{th}}$ entry of C'_s by at most ε'. By construction of C'_s, this entry is nothing but $C_{i,\ell}$. Similarly, we had ensured that the length distribution satisfies $\left| \|\boldsymbol{x}_{i,s}\| - \sqrt{\frac{n_s}{n+1}} \right| \leq \varepsilon'$. So by choosing ε' sufficiently small, we can ensure that $\left\langle \boldsymbol{x}_i \,, \boldsymbol{v}_\ell^{(r_0,\ldots,r_s)} \right\rangle$ is ε-close to $\sum_s \sqrt{\frac{n_s}{n+1}} \sqrt{\frac{n_s}{n+1}} C_{i,\ell} = C_{i,\ell}$.

Running Time. Lastly, we need to argue about the running time. In the proof for the list sizes, we computed the factor by which each list shrinks per block and we know that the list sizes are concentrated at their expected values. So we may assume that the size of any $L_i'^{(r_0,\ldots,r_s)}$ is given by $|L_i| \, \widetilde{\mathcal{O}}(\kappa_i^{n_1+\ldots+n_s})$ (plugging in the upper bound only makes the algorithm slower). Since this is exponentially increasing in n, the running time of ConfExt is easily seen to be dominated by the cost of trimming. The total cost for all trimmings done in the r_s-loop is then given by

$$\widetilde{\mathcal{O}}\Big(\rho^{n_1+\ldots+n_s} \max_i |L_i| \, \kappa_i^{n_1+\ldots+n_{s-1}}\Big) \leq \widetilde{\mathcal{O}}\Big(\max_i \frac{1}{\kappa_i^{n_s}} \cdot \max_i |L_i| \, (\kappa_i \rho)^{n_1+\ldots+n_s}\Big). \tag{25}$$

We claim that we have $\kappa_i \rho \geq 1$. While this can be proven directly from the definition, the simplest way is just to observe that the (proof of the) Solution Preservation property implies that every individual $\boldsymbol{x}_i \in L_i$ will with high probability be contained in *some* output instance. The reason is that whether $\boldsymbol{x}_i \in L_i$ is contained in some output instance does not depend on the other lists $L_{i'}$ with $i' \neq i$. As $\boldsymbol{x}_i$ may be part of some solution, it must thus be contained in some output instance. This implies that the total size of all $L_i'^{(\boldsymbol{r})}$ together is at least $|L_i|$, which is equivalent (for $\varepsilon \to 0$) to $\kappa_i \rho \geq 1$. Having $\varepsilon > 0$ only increases both κ_i and ρ.

With $\kappa \rho \geq 1$, we get $\max_i \frac{1}{\kappa_i^{n_s}} \leq \rho^{n_s}$ and $(\kappa_i \rho)^{n_1+\ldots+n_s} \leq (\kappa_i \rho)^n$. Plugging this into (25) shows the claim.

This finally proves all the statements about ConfExt.