arXiv:2611.33028v1 [cs.DC] 24 Nov 2026

用于计算 V-BLAST置换的高性能算法*

TOMER PELLEG†、HAIM AVRON‡、和SIVAN TOLEDO§

摘要。 我们提出了两种用于计算一种称为 V-BLAST置换或其近似值的新颖高效算法。V-BLAST列置换用于 QR分解，而 QR分解是整数/离散最小二乘求解器的一部分。整个求解器是先进通信系统中接收机的一部分，这些系统会同时传输多个符号。文献中已经提出了两类用于计算精确 V-BLAST置换的高效算法。第一类算法基于一系列 Givens旋转，但遗憾的是具有很强的串行性。另一类算法基于对伪逆执行一系列降阶更新操作，虽然关键路径较短，但在数值上不太可能稳定。我们提出的新算法受到 V-BLAST与秩揭示 QR 分解之间相似性的启发。我们的第一种算法使用 Householder反射，具有数值稳定性和较短的关键路径。然而，它无法高效利用缓存存储器。为此，我们采用一种随机化算法加以改进，该算法计算 V-BLAST置换的近似值，具有较短的关键路径，并能有效利用缓存存储器。我们证明了该随机化算法的近似性质，并通过实验表明，两种算法都比现有算法更快，无论是在单独测试时，还是在整数最小二乘求解器中进行测试时，均是如此。

关键词。 V-BLAST、整数最小二乘、QR分解、随机化算法、并行计算

MSC分类号。 65F20、65F25、65Y05、65Y20

1. 引言。 包括无线通信、GPS定位和密码分析在内的若干重要应用，都需要求解如下形式的整数最小二乘问题

$$\hat{x} = \arg\min_{x} \|W\,(Ax - b)\|_2$$

其中， A 是一个实或复 m×n 矩阵，满足 $m \geq n$； b 是一个实或复向量；W 是表示 m 约束权重的实矩阵； x或者是一个整数向量，或者是一个其元素属于实数或复数离散集合 [8, 16, 25]的向量。在本文中，我们假设 A 和 b 已由 W预乘，因此从现在起可以假设$\hat{x} = \arg\min_x \|Ax - b\|_2$，并且还假设 A 具有满列秩。

在通信领域，这些问题有时采用近似方法求解，有时则精确求解。在其他领域，通常需要精确的最小化解，甚至
k 最佳解。近似解通常会返回一个所谓的 Babai点 [2]，它可以通过计算 QR分解 $A = QR$得到，其中 Q 是具有正交归一列的 m×n矩阵， R 是上三角矩阵，随后再执行类回代算法。该算法求解 $R_{n,\,n}x_n = (Q^*b)_n$ ，然后将 x_n 舍入到最接近的可接受值 $\tilde{x}_n$（例如，舍入为整数；在通信中，这称为量化或切片）。接下来，通过从 $(Q^*b)_{1:n-1}$中减去 $R_{1:n-1,\,n}\tilde{x}_n$ ，将问题降维。随后重复此过程，直到近似解中的所有元素

*资助：本研究部分由布拉瓦特尼克计算机科学研究基金和以色列科学基金会（资助编号 1524/23）资助。

†布拉瓦特尼克计算机科学与人工智能学院，特拉维夫大学
‡特拉维夫大学数学科学学院§布拉瓦特尼克计算机科学与人工智能学院，特拉维夫大学；目前访问苏黎世联邦理工学院

最小化解 $\tilde{x}$ 均已确定[1]。

为了得到一个最小化解或最佳的 k 解，本文采用一种称为 Schnorr-Euchner（SE）搜索的分支定界算法 [20]。SE 搜索还需要对 A进行 QR分解。本文中的大部分 SE 搜索细节与本文主题无关。

Babai点的近似质量和 SE 搜索的效率都深受特定矩阵 A的影响；在此语境中，该矩阵的列被视为格的基。假设我们对 A的列进行变换，

$$\hat{x} = \arg\min_{x} \left\| APP^{-1}x - b \right\|_2 .$$

如果我们选择一种变换 P，使 x 所属的集合在该变换下保持不变，那么我们就可以在 x 的定义域中搜索，从而近似或精确地求解 $P^{-1}x$的问题，然后通过乘以 P来恢复解。

当 x 是整数向量时， P 可以是任意幺模变换。一种称为 LLL 的算法 [18]及其变体，常用于 GPS定位和密码分析，以寻找一个幺模矩阵 P，使其在 Babai点的表现和 SE 搜索效率方面都较为理想 [7]。然而，在通信中， x 的元素通常取自一个有限小集合。一般而言，这些集合在幺模变换下并不保持不变。在这种情况下， P 通常仅限于排列 [26]。

用于构造能够改善 Babai点和 SE 搜索的置换的主流方法称为 V-BLAST [8, 28]。最初用于构造 V-BLAST置换的算法需要执行 $\Theta(mn^2 + n^4)$ 次算术运算，但后来这一数量降至 $\Theta(mn^2 + n^3)$ 次。

下一节将介绍最初提出的用于构造 V-BLAST置换的算法，以及一些变体，其中有些变体在渐近运算次数方面高效得多。然而，在现代硬件上，运算次数通常与运行时间的缩短并不高度相关。能否同时利用多个处理单元，以及存储系统的负载，往往同样重要，甚至通常更为重要。

有趣的是，据我们所知，尚未有人分析 V-BLAST 算法中可用的并行性、这些算法给存储系统带来的负载（以缓存未命中和/或处理器间通信衡量），或致力于改善这些指标。

本文恰好针对这些问题展开研究。我们分析了现有 V-BLAST 算法的并行性和缓存未命中情况，并指出，在这两方面，其中大多数算法都远逊于其他大多数立方复杂度的稠密矩阵算法。现有的一种 V-BLAST 算法具有尚可的并行性，但代价是数值稳定性降低。

我们还提出了两种新算法：一种精确的 V-BLAST 算法，具有良好的并行性（与大多数立方复杂度的稠密矩阵算法相近）和出色的数值稳定性；另一种近似 V-BLAST 算法，兼具良好的并行性和对缓存中数据的有效复用。我们从理论和实验两方面分析了这两种算法，并证明它们确实如所述那样有效。

本文其余部分的组织如下。下一节将介绍有关 V-BLAST 的补充背景，以及用于计算 V-BLAST置换的现有算法。第 3 节

[1]我们使用 Matlab 记法表示索引范围： $i:\ j = \{i, i+1, \cdots, j\}$，其中 i和 j均为整数；若 $j < i$，则 $i:\ j = \emptyset$。

分析现有算法中的并行性和缓存未命中。第 4 节提出一种使用 Householder反射计算 V-BLAST置换的新算法。第 5 节提出一种基于矩阵草图的随机化算法，用于计算 V-BLAST 排序的近似结果。第6节确立了该随机化算法的若干重要理论性质，从而有效证明了它能够生成良好的近似结果。第7节给出了广泛的实验结果，验证了我们的理论分析；第8节总结了本研究的结论。

2. 背景。一般而言，V-BLAST（以及一种更简单的置换构造方法 P，还有 LLL 及其变体）的目标是对 A 的各列进行置换，使其 R 因子在右下角具有较大的对角元素，在左上角具有较小的对角元素。一种特别简单的贪心方法称为 SQRD，它通过 Gram-Schmidt过程，计算采用列主元选取的右看式 QR 分解。在第 j步中，将约化矩阵中范数最小的列与第 j 列交换，并从右侧各列中减去该列的投影 [30]。因此， r_{jj}的大小（等于约化矩阵中所选列的范数）在每一步中都会被（贪心地）最小化。列选择准则与秩揭示 QR 分解所用的准则恰好相反。该算法执行 $\Theta(mn^2)$ 次运算。然而，即使不考虑 Gram-Schmidt过程的数值不稳定性，研究发现，此方法选出的置换不如 V-BLAST 置换 [30,31]。

V-BLAST 更为复杂。它从右向左选择各列。在步骤 j，它计算每个剩余列相对于其他所有剩余列的正交补范数。它选择正交补最大的列，并将其置换到位置 $n-j+1$。更形式化地说， [6],将置换 P 中最后出现的 i 列记为 S_i，并将其他列记为 $U_i = \{1,\dots,n\} \setminus S_i$，V-BLAST 按照以下规则扩展 S_i

$$S_{i+1} = S_i \cup \arg\max_{j \in U_i} \left\| A_{1:m,j} - \operatorname{Span}\left(A_{1:m,U_i \setminus \{j\}}\right) \right\|_2 .$$

V-BLAST 背后的原理是统计学上的。在假设 $b = A\mathring{x} + e$ 的情况下，对于某个我们希望恢复的可行向量 $\mathring{x}$，以及具有良好统计特性的随机噪声向量 e，由 V-BLAST 排序恢复出的 Babai点会选择一种列顺序，使 x中的元素按方差递增的顺序进行估计。也就是说，V-BLAST 首先恢复连续（实数或复数）最大似然估计方差最小的符号（ x中的元素）。接下来恢复连续不确定性次小的符号（以第一个符号的离散赋值为条件），依此类推 [28]。

令人惊讶的是，尽管 V-BLAST 被定义为一种贪心算法，但它确实能提供某种全局保证 [28]:

$$P_{\text{V-BLAST}} = \arg\max_{P} \left\{ \min_i |R_{i,i}| \text{ such that } AP = QR \right\} .$$

也就是说，在所有排列中，V-BLAST 都能使 AP 的 R 因子中最小对角元素的模达到最大值。显然，最大化置换可能不止一种。

为 V-BLAST 提出的原始算法 [28] 执行 $\Theta(mn^3)$次运算。该算法计算 Moore-Penrose 伪逆 A^+，找到范数最小的行的索引

i ，并将其追加到排列的末尾。也就是说， A 的第 i 列变为 AP的第 n 列。然后，该算法将 A中的第 i 列置零，再次计算伪逆，并继续执行；在选择下一个范数最小的行时忽略伪逆中的第 i 行，依此类推。

这种方法的效率很低。文献中已经提出了几种渐近效率更高的方法。它们将在本节后文中介绍。

2.1. Hassibi 以及 Chang 和 Paige 提出的算法。Hassibi [14]以及后来的 Chang 和 Paige [6] 分别独立提出了效率高得多的算法。这两种算法密切相关。

两种算法都考虑（但不一定计算）QR分解$A = QR$。随后，它们根据 R^{-1}构造 P 。Hassibi 假设

$$A = \begin{bmatrix} H \\ \lambda I \end{bmatrix}$$

并提出使用所谓的平方根算法直接从 A 计算 R^{-1} 。本质上，他的计算 R^{-1} 的算法从（$1/\lambda$）I 开始，并通过一系列更新操作来更新逆矩阵 R 因子，每个操作对应 H的一行。Chang 和 Paige 不作任何额外假设。他们提出使用 Householder反射或 Givens旋转计算 R因子，然后显式求 R 的逆矩阵。

两种计算方法都需要 R^{-1} Θ（mn^2）次运算。除了一点之外，本节的细节与本文大多无关： R 因子可以通过并行且通信高效的算法 [9]稳定地计算。

从这里开始，两种算法以类似的方式继续，但 Hassibi 的文章论述非常简略，因此我们遵循 Chang 和 Paige 的表述。令 $L = R^{-T}$。 L 的对角元素是 R中对角元素的逆矩阵。该算法从最后一列（最右列）到第一列（最左列）构造置换，目标是在每一步中使 L 的对角元素的模最小。

我们将算法第 1 步至第 j 步中构造的置换记为 $P^{(j)}$ ，并将 $AP^{(j)}$的 R 因子的逆矩阵的转置记为 $L^{(j)}$ 。我们初始化$P^{(0)} = I$和 $L^{(0)} = R^{-T}$，其中 R是 A的因子。

在步骤 j中，算法找出 A 中应位于 AP 的位置 $n-j+1$的那一列。它会找到

$$i = \arg\min_{i \le n-j+1} \left\| L^{(j-1)}_{1:n-j+1,i} \right\|_2 .$$

现在，算法通过保留最后 $j-1$ 列，并将第 $n-j+1$ 列设为.....，从 $P_{(j-1)}$ 构造出 $P_{(j)}$

$$P^{(j)}_{:,n-j+1} = P^{(j-1)}_{:,i} ,$$

将列 $i+1$ 向左滑动： $n-j+1$ 一列，

$$P^{(j)}_{:,i:n-j} = P^{(j-1)}_{:,i+1:n-j+1} .$$

前 $i-1$ 列保持不变。

L 的各列经历相同的变换，

$$\begin{aligned} \bar{L}^{(j)}_{:,1:i-1} &= L^{(j-1)}_{:,1:i-1} \\ \bar{L}^{(j)}_{:,i:n-j} &= L^{(j-1)}_{:,i+1:n-j+1} \\ \bar{L}^{(j)}_{:,n-j+1} &= L^{(j-1)}_{:,i} \\ \bar{L}^{(j)}_{:,n-j+2:n} &= L^{(j-1)}_{:,n-j+2:n} . \end{aligned}$$

最后，我们对 $\bar{L}^{(j)}$ 的成对行进行一系列旋转，使该矩阵恢复为下三角形式 $L^{(j)}$。从数学上讲，这会修改 AP的 QR分解中的 Q 因子，但这没有关系，因为对于 V-BLAST 而言， Q 因子并不重要。 $\bar{L}^{(j)}$ 的非零结构为

$$\bar{L}^{(j)} = \begin{bmatrix} \bullet & & & & & & & & \\ \vdots & \ddots & & & & & & & \\ & & \bullet & & & & & & \\ & & \bullet & & & \bullet & & & \\ & & \bullet & \bullet & & \bullet & & & \\ \vdots & & \vdots & \vdots & \ddots & \vdots & & & \\ & & & & & \bullet & \bullet & & \\ & & & & & \bullet & \bullet & \bullet & \\ \vdots & & & & & \vdots & \vdots & \ddots & \\ \bullet & \cdots & \bullet & \bullet & \cdots & \bullet & \bullet & \bullet & \cdots & \bullet \end{bmatrix}.$$

现在，算法对相邻行对执行一系列旋转，从 $i, i+1$ 开始，一直向下进行到 $n-j$, $n-j+1$。这会消除列 $n-j+1$ 中的尖峰，并在列 i 至 $n-j$ 中重新填入主对角线。这样就得到了 $L^{(j)}$ ，我们可以开始下一步。

Hassibi 也使用酉变换，在某一列移动到最终位置后，将逆矩阵 R 因子恢复为三角形式，但没有说明如何构造这种变换。

现在让我们分析该算法的关键性能特征，以构造 $P = P^{(n)}$。在步骤 j 中，我们执行 $O(n^2)$ 次运算来找出 i （通过维护一个列范数平方数组，并在每一步结束时更新该数组，可将运算量降至 $O(n)$）。移动 L 的列也需要 $O(n^2)$ 次运算，但使用指向各列的指针数组，同样可将运算量降至 $O(n)$。步骤 j 的主要开销是一系列 Givens旋转。共有 $O(n)$ 次旋转，每次旋转作用于两行，每行有 $O(n)$ 个元素，因此 Givens旋转的总开销为 $O(n^2)$。因此，该阶段的总开销为 $O(n^3)$。文献中没有对其他性能预测指标进行分析；我们将在下一节对此进行分析。

2.2. 基于伪逆降阶的算法。Benesty、Huang 和 Chen [4] 提出了另一种方法，以降低原始 V-BLAST 算法的复杂度。随后，Luo 等人 [19], 、Szczeciński 和 Massicotte [23] 以及 Shang 和 Xia [22]对该算法进行了改进。所有这些研究都采用了相同的核心思路，我们将在此介绍。

考虑原始 V-BLAST 算法 [28]。该算法在每次迭代中将 A的一列置零，并对各列进行置换，使置零的列位于末尾。由于我们只关心置换，因此可以将置零的列留在原位。我们将经过第 i次迭代、其中已有 i列被置零后的矩阵记为 $A^{(i)}$。原始算法会计算每个 $A^{(i)}$的 Moore-Penrose 伪逆，以找出$\left(A^{(i)}\right)^{+}$中范数最小的行，同时忽略 $A^{(i)}$ 中与已置零列对应的行。该行的索引就是第 $i+1$次迭代中被置零的列的索引。

Benesty 等人 [4] 的算法基于两个想法。首先，观察到 $\left(A^{(i)}\right)^{+}$ 中各行的范数是

$A^{(i)}$的格拉姆矩阵伪逆的对角元素的平方根，

$$\left(\left(A^{(i)}\right)^* A^{(i)}\right)^+_{j,j} = \left\|\left(A^{(i)}\right)^+_{j,:}\right\|_2^2 .$$

记

$$M^{(i)} = \left(\left(A^{(i)}\right)^* A^{(i)}\right)^+ .$$

De注意到在第 $i+1$ 步中要消去的列为 j，Benesty 等人证明了

$$M^{(i+1)} = M^{(i)} - \frac{M^{(i)}_{:,j} \cdot \left(M^{(i)}_{:,j}\right)^*}{M^{(i)}_{j,j}} .$$

这使得可以通过从 $M^{(0)} = (A^*A)^{-1}$ 开始进行一系列降阶更新操作，计算 $M^{(i)}$s 的序列。

文献中提出的这种方法的不同变体，在 $M^{(0)}$ 的计算方式以及与 V-BLAST置换构造无关的方面存在差异。这些问题基本上都与本文无关。

2.3. 与带列主元的 QR 分解的表面联系。我们注意到，V-BLAST置换 的构造与带列主元的 QR分解存在表面上的相似性。带列主元的 QR 分解同时对 A 的列置换进行 QR分解 $AP = QR$，并构造该置换。该分解采用右看方式。在选定前 $i-1$ 列后，算法以使$|R_{i,i}|$最大化的方式选择下一列进行消去。

带列主元的 QR 分解是计算秩揭示 QR 分解的传统策略；如今还有其他策略，往往能带来更高的性能。

我们在设计新的 V-BLAST 算法时利用了这一观察。

3. 现有算法的复杂度分析。原始 V-BLAST 算法 [28] 的渐近计算开销很大。假设不使用 Strassen 类方法，则每次都从头开始计算一系列 n矩阵的伪逆，需要执行 $\Theta(mn^3)$次算术运算。

Hassibi [14] 和 Chang 和 Paige [6] 提出的算法只需要$\Theta(mn^2)$次运算，显著改善了渐近性能。无线通信文献中的大多数其他算法，则着重于将所需的算术运算次数减少一个常数因子。

然而，算法在现代计算机上的实际性能，不仅取决于该算法执行的算术运算次数，还取决于它利用多处理器或算术单元的能力，以及算法中的数据移动模式。与算术运算总次数相比，这些算法特性对运行时间的影响可能大得多。

3.1. 并行性与数据重用的简单模型。我们通过算法的关键路径来分析并行性。也就是说，我们利用算法中最长的依赖链（称为关键路径）来刻画算法可用的并行性。关键路径的长度称为跨度（或简称关键路径），它对运行时间构成上界，且与可用处理器的数量无关

也与数据移动的速度无关。关键路径越短，算法就越有可能利用多个处理单元。特别是在许多矩阵算法中，跨度记作 T_∞，是预测性能的良好指标。更具体地说，在对算法采取相当保守的假设、但对存储系统采用较宽松假设的情况下，p 个处理器可以在略多于 $T_\infty + T_1/p$ 步的时间内执行一个需要进行 T_1 次操作的算法 [5]。

我们使用一种顺序双层存储器层次结构模型来分析数据移动的代价，该模型正式称为 Hong-Kung棋子游戏 [17, 24]。具体而言，我们假设计算机配有容量为 M 字的快速存储器（缓存），以及容量无限的慢速存储器。处理器只能对快速存储器中的数据进行操作，并且只能将结果存入快速存储器。我们还假设，算法数据结构的任意子集都可以同时驻留在缓存中，并且缓存驱逐采用最优方式执行。我们使用算法必须在慢速存储器与快速存储器之间执行的数据传输次数来衡量其性能。

对于大多数常见的矩阵分解，例如采用部分选主元的 LU分解、Cholesky分解和 QR 分解，都存在实用且向后稳定的算法，其关键路径为 $\Theta(n)$；也存在缓存高效的算法，可在快速存储器与慢速存储器之间移动 $\Theta(T_1/\sqrt{M})$字 [3]。这些数据移动上界在渐近意义下是最优的。虽然存在跨度为多对数的分解算法，但它们在数值上不稳定。

本文不讨论更复杂的数据移动（或并行性）理论模型，例如分布式存储模型、通信规避算法等。

3.2. Hassibi 算法以及 Chang 和 Paige 算法中的并行性与缓存未命中。这两种算法首先计算 R^{-1} 的 R因子的逆矩阵$A = QR$。如果使用 Householder反射计算 R，则关键路径长度为$\Theta(n)$，并且存在缓存高效的变体 [10]。通过回代可以计算 R 的逆矩阵，其关键路径为 $\Theta(n)$，且数据复用最优。此外，还有关键路径呈对数级、数据复用良好的三角矩阵求逆算法。这些算法都有高质量的实现。我们认为，从并行性和数据复用的角度来看，这些预处理步骤已经得到妥善解决。需要指出的是，Hassibi 以及后续提出的一些算法采用其他方法计算 R^{-1}，这些方法数值稳定性较差，而且数据重用比似乎也差得多。

Chang 和 Paige 算法的关键路径长度为 $\Theta(n^2)$，渐近地看，这比大多数矩阵分解中的关键路径长得多。Hassibi 的算法中缺少相关细节，但它似乎暗示了同一个算法。

更具体地说，该算法执行 $\Theta(n^2)$ 次 Givens旋转。每次旋转都可以在常数级关键路径内完成，从而得到该上界。据我们所见，这些 Givens旋转必须顺序地执行，而任何此类实现的关键路径均为 $\Theta(n^2)$。

这种 Givens旋转的顺序意味着，在算法的大多数 $n-1$步中，它可能会变换 L 中常数比例的元素，因此，当内存大小为 $M \leq n^2/8$ 字时，缓存未命中次数为 $\Omega(n^3)$。

3.3. 对伪逆进行降阶更新的算法中的并行性与缓存未命中。我们再次忽略计算（A^*A）$^{-1}$的预处理阶段，因为可以采用多种方式，利用具有线性关键路径和良好数据重用比的算法来完成该计算。这些方式包括计算 R^{-1}，然后将（A^*A）$^{-1} =$（R^*Q^*QR）$^{-1} =$（R^*R）$^{-1} = R^{-1}R^{-*}$相乘，或者甚至采用奇异值分解。我们再次指出

无线通信文献中的大多数变体提出了计算初始伪逆（A^*A）$^{-1}$的其他方法，而这些方法似乎存在次优的数据重用比问题。我们再次认为，这一预处理步骤的性能问题留待解决。

Benesty 等人的算法 [4] 对一个方形对称（厄米）矩阵执行一系列 n 降阶更新操作。这些降阶更新操作很简单；它们相当于具有常数关键路径的简单秩一更新。每次秩一更新后，算法都会寻找符合条件的最小对角元素；此操作的关键路径为对数级。因此，关键路径的长度为Θ($n$log n)。 $M \leq n^2/8$ 的数据复用率为常数，因此缓存未命中的数量为 Ω(n^3)；这是这些算法的一个缺点。

4. 使用正交变换构造置换，并实现近线性跨度。现在，我们介绍 Hassibi/Chang 和 Paige 算法的一种变体，其跨度近似线性。与他们的算法一样，该算法仅对 R^{-T}使用酉变换，因此与伪逆降阶更新方法相比，它较不容易受到数值问题的影响。

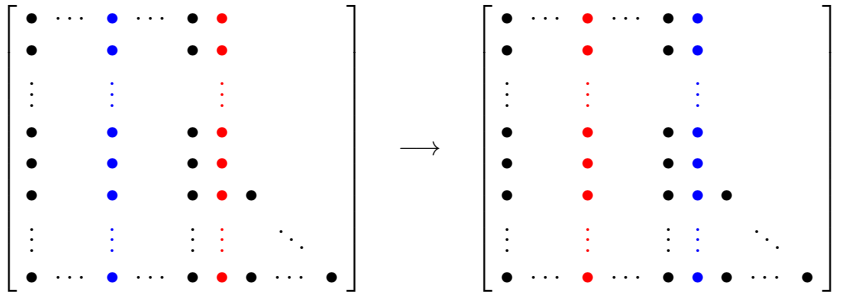
思路很简单。我们从 $L = R^{-T}$ 开始，计算其各列的二范数，并将范数最小的列与最后一列交换。Chang 和 Paige 使用 Givens旋转消去列 n 中的尖峰，而我们改用 Householder反射。这样会填充第 1 至 $n-1$ 行和列中的上三角部分。我们不试图将矩阵恢复为下三角形式。我们更新列范数，以反映这次交换，并排除最后一行中的元素。至此，步骤 1 完成。我们在 1 至 $n-1$ 列中寻找最短的列 j' ，将其与第 $n-1$ 列交换，并消去列 $n-1$ 中的尖峰。注意，Householder反射不会影响第 n 行，因此最后一列的非零结构得以保留。

然后我们重复这一过程，同时保持这样一个不变量：步骤 i 中的 Householder反射不会影响第 $n-i+2$ 至 n 行。因此，步骤 i 之后，最后 i 列构成一个下三角矩阵，

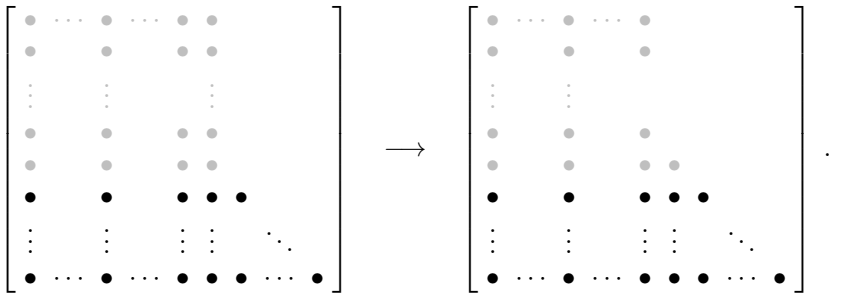
$$\bar{L}^{(i)} = \begin{bmatrix} \bullet & \cdots & \bullet & \cdots & \bullet & \bullet & & \\ \bullet & & \bullet & & \bullet & \bullet & & \\ \vdots & & \vdots & & & \vdots & & \\ \bullet & & \bullet & & \bullet & \bullet & & \\ \bullet & & \bullet & & \bullet & \bullet & & \\ \bullet & & \bullet & & \bullet & \bullet & \bullet & \\ \vdots & & \vdots & & \vdots & \vdots & & \ddots \\ \bullet & \cdots & \bullet & \cdots & \bullet & \bullet & \bullet & \cdots & \bullet \end{bmatrix}.$$

在此 此时，算法将范数最小的列（不包括

t底部的 i 个元素）与列 $n-i$ 交换，如下方蓝色所示；列 $n-i$ 如下方红色所示。

$\bar{L}^{(i)} =$ 

对灰色所示的行进行 Householder反射，可使下三角结构再扩展一列，



我们注意到，该算法与带列主元的 QR 分解完全互为镜像：

- 我们处理的是下三角矩阵，而不是上三角矩阵，
- 我们从末端（右侧）构造置换，而不是从起始端（左侧）构造置换，
- 并且我们选择范数最小的列（在尚未成为三角结构一部分的行中），而不是选择范数最大的列。

4.1. 并行性和缓存未命中的分析。我们忽略 $L = R^{-T}$的计算，因为正如第3.2节所述，该计算可以在短跨度内以缓存高效的方式完成。

我们首先分析上述基于 Householder 变换的算法的并行性。该算法包含 $n-1$ 个顺序步骤。每个步骤选择一列。在每个步骤中：

- 算法计算所有剩余列的范数，
- 计算范数最小列的 Householder反射，
- 并将该反射应用于其余的剩余列。

这些阶段的跨度为 $\Theta(\log n)$。因此，该算法的总跨度为$\Theta(n\log\ n)$。这比 Chang 和 Paige 的算法有了显著改进，并且与降级更新方法具有相同的渐近界。

遗憾的是，该算法的缓存效率不高。每一步都需要访问剩余的子矩阵，以更新其中剩余的列。对每个元素执行的操作次数是一个较小的常数，因此，对于容量为 $M \leq n^2/8$ 字的内存，缓存未命中次数为 $\Omega(n^3)$。

5. 使用随机投影近似 V-BLAST。 第 4 节提出的算法就操作次数而言具有渐近高效性

并且具有相当短的跨度。然而，其数据访问模式较差，因为在处理每一列时，它都会扫描整个约化子矩阵。带列主元的 QR 分解具有类似的行为。设计具有更佳数据访问模式的秩揭示 QR 分解的尝试，至少催生了两类算法。本节将展示，其中一类基于随机投影的思想 [13],适用于 V-BLAST。（另一类基于锦标赛 [10]；我们尚未找到将这一思想应用于 V-BLAST 的方法。）

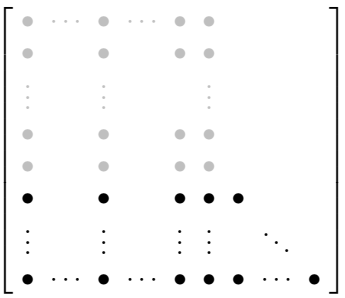
我们的算法与 Chang 和 Paige 的算法以及第 4 节的算法一样，从$L = R^{-T}$开始，其中 $A = QR$。该算法采用分块（或划分）方式，也就是说，扫描整个约化子矩阵后，我们可以选出 AP中下一个包含b 个连续列的块，而不只是选出一列。该算法具有随机性，因此所选块中的列不一定是 V-BLAST 接下来会选出的 b列。

我们首先为 n×n 矩阵 L构造一个 (b+p)×n的草图。构造草图的方法是将 L 左乘一个 (b+p)×n 的随机矩阵 G；该矩阵具有随机投影性质（也就是说，它以很高的概率保持范数以及子空间内列之间的某些关系，这是一种无偏子空间嵌入 [29]）。我们的理论（第6节）分析了含有独立同分布标准正态元素的 G 。这些结果可以推广到其他类型的无偏子空间嵌入（OSE），但界限会有所不同。

现在，我们对 GL 执行基于 Householder 变换的算法的 b 个步骤，该算法见第 4 节。更具体地说，我们选择的第一列是 GL 中欧几里得范数最小的列。然后使用 Householder 反射消去该列的前 $b+p-1$ 个元素，并继续处理约化矩阵的前 $b+p-1$ 行。我们将把所选的 b 列依次移到末尾的置换记为 $P^{(1:b)}$。

现在，我们相应地重新排列 L，用 $LP^{(1:b)}$ 替换 L。接下来，我们使用 b 个 Householder 反射，对 $LP^{(1:b)}$ 中最右侧的 b 列（即使用草图选出的列）进行下三角化。随后，利用 WY 表示 [21]，将这 b 个反射以 Θ（b）列为一块应用于 $LP^{(1:b)}$ 的其余部分。

这一过程得到如下形式的约化矩阵



其中，右侧的三角部分包含 b 列。该算法使用独立采样的 G，针对不属于矩阵中已下三角化部分的行和列（即上方以灰色显示的元素）构造一个新的草图，利用它选出接下来的 b 列，依此类推。

该算法所用草图矩阵的分布会将给定的输入矩阵 L （或原始的 A）映射为一个置换分布。下面我们通过理论分析和实验表明，得到的置换几乎与 V-BLAST 同样有效的概率很高。

6. 随机投影算法的理论分析。现在我们来分析第 5 节中介绍的算法。我们证明，该算法返回的置换以很高的概率具有两项性质，这两项性质与 V-BLAST置换的两项重要最优性性质非常相似。

第一项性质是 V-BLAST 贪心列选择规则的概率版本：在每次迭代中，V-BLAST 都会选择与尚未选取的其他列的跨度距离最大的那一列 [6]。

引理 1. 设 $0 < \delta < 1/2$ 和 $0 < \varepsilon < 1$。假设我们的算法使用的投影维数为

$$b + p \geq C\left(\frac{b \log n + \log \frac{1}{\delta}}{\varepsilon^2}\right)$$

其中 C为某个绝对常数。设 P 为第 5 节所述算法返回的置换（注意，这是一个随机排列）。设 U_t 为对应于 AP 前 $n-t$ 列的索引集合（也就是说，在算法第 t 次迭代开始时尚未选取的索引，迭代计数从0开始）。设 c_{t+1} 为我们的算法所选第 $t+1$ 列的索引（在第t次迭代中选取），即 $A_{1:m,\ c_{t+1}} = (AP)_{1:m,\ n-t}$。

那么，以至少 $1-\delta$ 的概率，对每个 $0 \leq t \leq n-1$，都有

(1)
$$\left\|A_{1:m,c_{t+1}} - \ Span\left(A_{1:m,U_t\setminus\{c_{t+1}\}}\right)\right\|_2 \geq \frac{1-\varepsilon}{1+\varepsilon} \max_{j\in U_t} \left\|A_{1:m,j} - \ Span\left(A_{1:m,U_t\setminus\{j\}}\right)\right\|_2$$

这意味着，尽管我们的算法在每次迭代中可能不像 V-BLAST 那样选择最优的可用列，但它会选择一列近乎最优的列：在受算法先前选择影响的条件下，它所选的每一列与最优可用距离都很接近。请注意，在上述性质中将 $\frac{1-\varepsilon}{1+\varepsilon}$ 替换为 1，意味着算法会选择距离尚未选取的各列的跨度最远的那一列，这正是 V-BLAST 的做法 [6]。

为了证明引理1，我们需要以下辅助引理。该引理指出，投影大致保留了 L 的各列（回顾一下 $L = R^{-T}$，其中 $A = QR$）。在给出引理之前，我们需要引入一些记号。令 l_i 为 L 的第 i 列 ($l_i = L_{1:n,\ i}$)。令 $S_t = [n] \setminus U_t$ 表示算法选取的前 t 列的索引集合，其中 $t = 0$、.....、n。令 $l_i^{(t)}$ 为将 l_i 正交投影到 Span ($l_j | j \in S_t$) 的正交补所得的投影。请注意，在第 t 次迭代时，V-BLAST 选择使 $\|l_i^{(t)}\|_2$ 最小的列，搜索范围为 $i \in U_t$ [6]。除非 t 是 b 的倍数，否则我们的算法不会显式计算 $l_i^{(t)}$ 或其范数，具体如下文所述。为了近似计算 $\|l_i^{(t)}\|_2$，我们的算法依赖于对约化矩阵进行草图处理。选取 t 列后，令 $w = b \cdot \lfloor \frac{t}{b} \rfloor$ 为该块第一步的索引，它也等于此前所有已完成块中选取的列数（每个块选取了 b 列）。

用 $V_t = S_t \setminus S_w$ 表示在第 t 次迭代开始时，当前块中选取的 $t \mod b$ 列的索引集合。令 H_w 为表示迄今为止作用于 L 上的累积 Householder反射的 $n \times n$ 酉矩阵（注意，Householder 反射仅在块边界处作用于 L；因此我们只在 w 是 b 的倍数时定义 H_w）。用 $n-w$ 维向量$r_i^{(w)} = (H_w l_i)_{1:(n-w)}$表示经过 w 次列选择后约化矩阵中的第 i 列。当 H_w 作用于 l_i时，$H_w l_i$ 的底部 w 个坐标的范数与

l_i 在 Span $(l_j|j \in S_w)$ 上的投影范数相同，因为 $H_w(LP)$ 最右侧的 w 列构成三角形。由于 H_w 是酉矩阵，这意味着 $H_w l_i$的顶部 $n-w$ 个元素的范数与 l_i 在 Span $(l_j|j \in S_w)$ 的正交补上的正交投影范数相同，即 $\|r_i^{(w)}\|_2 = \|l_i^{(w)}\|_2$ （形式化证明见附录 (B)[2]）。

由于会显式计算 $r_i^{(w)}$ （在每一步应用 b 个分块 Householder 反射之后），因此我们确实可以显式计算 $\|l_i^{(t)}\|_2$，当 t 是 b 的倍数时，正如上文所述。对于从迭代 w 开始的块，该算法抽取一个大小为 $(b+p) \times (n-w)$ 的独立草图矩阵 G_w。记 $\tilde{l}_i^{(w)} = G_w r_i^{(w)}$，并将 $\tilde{l}_i^{(t)}$ 定义为 $\tilde{l}_i^{(w)}$ 在 Span $\left(\tilde{l}_j^{(w)}|j \in V_t\right)$ 的正交补上的正交投影。根据与上文相同的推理，我们的算法通过选择具有最小 $\|\tilde{l}_i^{(t)}\|_2$ 的列来选择第 $(t+1) - th$ 列，并在每一步显式计算该值。

现在我们可以陈述辅助引理了。该引理表明，我们的算法可以使用 $\|\tilde{l}_i^{(t)}\|_2$ （算法确实计算了该量）来估计$\|l_i^{(t)}\|_2$ （算法并不计算该量）。

引理 2. 设 $0 < \delta < 1/2$ 和 $0 < \varepsilon < 1$。假设我们的算法使用的投影维数为

$$b + p \geq C\left(\frac{b \log n + \log \frac{1}{\delta}}{\varepsilon^2}\right)$$

其中 C 为某个绝对常数。

那么，以至少 $1-\delta$ 的概率，对于每个可行的 i 和 t,

$$(1-\varepsilon)\left\|l_i^{(t)}\right\|_2 \leq \frac{1}{\sqrt{b+p}}\left\|\tilde{l}_i^{(t)}\right\|_2 \leq (1+\varepsilon)\left\|l_i^{(t)}\right\|_2$$

证明。注意，在减去所选取的前t 列的某个线性组合后，$l_i^{(t)}$ 是 l_i。此外，我们可以将 $l_i^{(t)}$ 表示为 $l_j^{(w)}$ 的线性组合，其中某些系数为 j：对于 $j \in V_t$，在减去 $l_j^{(w)}$ 的一个线性组合后，它是 $l_i^{(w)}$。根据定义，我们还知道，$\tilde{l}_i^{(t)}$ 是 $\tilde{l}_j^{(w)}$ 的线性组合，其中某些系数为 $j \in V_t \cup \{i\}$。众所周知，如果 G是一个含有独立同分布标准正态元素的 $\underline{k} \times n$ 随机矩阵，那么当 k 足够大时，$G/\sqrt{k}$ 是一个无偏子空间嵌入。也就是说，对于每个固定的 d 维子空间，以至少 $1-\Delta$ 的概率，对于该子空间中的每个向量 x，

$$(1-\varepsilon)\|x\|_2 \leq \frac{1}{\sqrt{k}}\|Gx\|_2 \leq (1+\varepsilon)\|x\|_2 .$$

获得这一保证所需的规模为 k $k \geq C_0$ （$d + \log$（$1/\Delta$）） $\cdot \varepsilon^{-2}$，其中 C_0 [29, 是某个绝对常数 6]定理。

对于每个块 w，在给定前面各块产生的全部随机性的条件下，向量 $r_i^{(w)}$ 随即固定，而新的草图 G_w 与这一历史信息相互独立。对于固定块，在每一种这样的历史条件下，以下失败概率界均成立，因此无条件下同样成立。我们令 Δ足够小，使得上述范数保持性对于所有由以下向量张成的子空间同时成立

[2]由于这是一个技术性很强的证明，且与主证明的行文脉络无关，因此我们将其推迟到附录中。

由 b 中的向量从 $\left\{r_1^{(w)},\ \ldots,\ r_n^{(w)}\right\}$ 对任意 w 张成。对于固定的 w，此类子空间至多有 $\binom{n}{b}$ 个，因此我们可以将子空间的数量界定为

$$\binom{n}{b} \le \left(\frac{en}{b}\right)^b .$$

因此，如果我们选择

$$\Delta = \frac{\delta}{n \cdot \left(\frac{en}{b}\right)^b}$$

那么，对于每个固定的 w，以 $\left\{r_1^{(w)}, \ldots, r_n^{(w)}\right\}$ 中的 b 个向量张成的所有子空间都满足范数保持的概率至少为 $1-\delta/n$。注意到共有 $\lceil \frac{n}{b} \rceil \le n$ 个块，我们现在使用并合界，得出该保证对所有块和子空间均成立的概率至少为 $1-\delta$。因此，投影维数需要为

$$\begin{aligned} b + p \ge C_0 \left(\frac{b + \log\left(\frac{1}{\Delta}\right)}{\varepsilon^2}\right) &= C_0 \left(\frac{b + \log\left(\frac{n\cdot\left(\frac{en}{b}\right)^b}{\delta}\right)}{\varepsilon^2}\right) \\ &= O\left(\frac{b\log(n) + \log\left(\frac{1}{\delta}\right)}{\varepsilon^2}\right) \end{aligned}$$

因此，我们已经证明，对于足够大的 $b+p$ 值，$G_w/\sqrt{b+p}$ 会保持由 $\left\{r_1^{(w)}, \overline{\ldots, r_n^{(w)}}\right\}$ 中的 b 个向量张成的所有子空间的范数，对每个 w 都成立。

记 $q_i^{(t)}$ 为将 $r_i^{(w)}$ 正交投影到 $\mathrm{Span}\left(r_j^{(w)} | j \in V_t\right)$ 上的结果。由于 $r_i^{(w)} = (H_w l_i)_{1:n-w}$ 且 H_w 正交，我们有 $\left\|l_i^{(t)}\right\|_2 = \left\|r_i^{(w)} - q_i^{(t)}\right\|_2$（更正式的证明见附录 B）。记 $G_w \cdot \widetilde{q}_i^{(t)}$ 为将 $\widetilde{l}_i^{(w)}$ 正交投影到 $\mathrm{Span}\left(\widetilde{l}_j^{(w)} | j \in V_t\right)$ 上的结果（因为对于 $j \in V_t$，向量 $\widetilde{l}_j^{(w)}$ 都在 G_w 的值域中，所以投影也在其中）。因此，根据定义，$q_i^{(t)}$ 位于 $\mathrm{Span}\left(r_j^{(w)} | j \in V_t\right)$ 内，我们可以选择 $\widetilde{q}_i^{(t)}$ 使其也位于同一子空间中。

由于 $q_i^{(t)}$ 是跨度 $\left(r_j^{(w)} | j \in V_t\right)$ 中距离 $r_i^{(w)}$ 最近的向量，因此我们有

$$\left\|l_i^{(t)}\right\|_2 = \left\|r_i^{(w)} - q_i^{(t)}\right\|_2 \le \left\|r_i^{(w)} - \widetilde{q}_i^{(t)}\right\|_2 .$$

由于 $G_w \cdot \widetilde{q}_i^{(t)}$ 是由 $\left(\widetilde{l}_j^{(w)} | j \in V_t\right)$ 和 $\widetilde{l}_j^{(w)} = G_w \cdot r_j^{(w)}$ 张成的空间中距离 $\widetilde{l}_i^{(w)}$ 最近的向量，因此我们有

$$\left\|\widetilde{l}_i^{(t)}\right\|_2 = \left\|\widetilde{l}_i^{(w)} - G_w \cdot \widetilde{q}_i^{(t)}\right\|_2 \le \left\|\widetilde{l}_i^{(w)} - G_w \cdot q_i^{(t)}\right\|_2 = \left\|G_w \cdot r_i^{(w)} - G_w \cdot q_i^{(t)}\right\|_2 .$$

最后，为完成证明，我们结合最后两个不等式，以及由所有块的列所张成的所有子空间的范数保持性质：如果每个 $G_w/\sqrt{b+p}$ 都将形如 $\{\overline{r_i^{(w)}}\} \cup \{r_j^{(w)}$ 的跨度中的向量的范数保持在 $1 \pm \varepsilon$ 的因子范围内（如上文所示，这一情况发生的概率至少为 $1-\delta$，因为这些子空间至多由来自 $\left\{r_1^{(w)}, \ldots, r_n^{(w)}\right\}$ 的 b 个向量张成），则有

$$
\begin{aligned}
(1-\varepsilon)\left\|l_i^{(t)}\right\|_2 &\le (1-\varepsilon)\left\|r_i^{(w)}-\widetilde{q}_i^{(t)}\right\|_2 \\
&\le \frac{1}{\sqrt{b+p}}\left\|G_w\cdot\left(r_i^{(w)}-\widetilde{q}_i^{(t)}\right)\right\|_2 \\
&= \frac{1}{\sqrt{b+p}}\left\|\widetilde{l}_i^{(w)}-G_w\cdot\widetilde{q}_i^{(t)}\right\|_2 \\
&= \frac{1}{\sqrt{b+p}}\left\|\widetilde{l}_i^{(t)}\right\|_2
\end{aligned}
$$

并且

$$
\begin{aligned}
\frac{1}{\sqrt{b+p}}\left\|\widetilde{l}_i^{(t)}\right\|_2 &\le \frac{1}{\sqrt{b+p}}\left\|G_w\cdot r_i^{(w)}-G_w\cdot q_i^{(t)}\right\|_2 \\
&\le (1+\varepsilon)\left\|r_i^{(w)}-q_i^{(t)}\right\|_2 \\
&= (1+\varepsilon)\left\|l_i^{(t)}\right\|_2
\end{aligned}
$$

□

证明了引理 2 后，我们现在可以证明引理1。

证明。 我们知道， $\|l_i^{(t)}\|_2$ 是 $A_{1:m,\ i}$ 到尚未选出的其他列的跨度 [6]的距离的逆矩阵。引理 2 保证，我们所选取的、使 $\|\widetilde{l}_i^{(t)}\|_2$，最小的列，其真实残差范数 $\|l_i^{(t)}\|_2$ 与最小残差范数之比 $\frac{1+\varepsilon}{1-\varepsilon}$ 在一个因子范围内。由于 $\|l_i^{(t)}\|_2$ 是 $A_{1:m,\ i}$ 到跨度$\left(A_{1:m,\ U_t\setminus\{i\}}\right)$的距离的逆矩阵，这就证明了引理1。

□

第二个性质是 V-BLAST 的另一个性质的概率版本，即它在 R因子 AP 的所有排列中，使最小对角元素的大小最大:

引理 3. 对于每个置换 P，记作

$$
R(P)=\min_i |R_{i,i}| \ \textit{such that}\ AP=QR
$$

令 $P_{R\text{-}V\text{-}BLAST}$ 表示我们的算法返回的置换矩阵。令 $0<\delta<1/2$ 和 $0<\varepsilon<1$。如果我们选择投影维数

$$
b+p\ge C\left(\frac{b\log n+\log\frac{1}{\delta}}{\varepsilon^2}\right)
$$

对于某个绝对常数 C，则其概率至少为 $1-\delta$，

$$
R\left(P_{R\text{-}V\text{-}BLAST}\right)\ge\frac{1-\varepsilon}{1+\varepsilon}\cdot\max_P R(P)
$$

证明。 首先，我们引入一些记号。 R（P）的记号中隐去了矩阵A 。为了证明，显式写出 A 矩阵很有用，即记作

$$
R(A,P)=\min_i |R_{i,i}| \text{ such that } AP=QR
$$

我们还记

$$
R_i(A,P)=|R_{i,i}| \text{ such that } AP=QR.
$$

给定一系列主元选择 c_1、.....、c_n，它定义了一个唯一的置换 P_c，该置换将列 c_1 放在最后， c_2 放在倒数第二位，依此类推。反过来，每个置换P都定义了唯一的一系列主元选择 c_1、.....、c_n 以及相应的集合U_0、.....、U_n。为使证明简洁，我们将省略 P_c 的下标，并默认 P 对应主元 c_1、.....、c_n 和集合 U_0、.....、U_n，而 P' 对应主元 c'_1、.....、c'_n 和集合 U'_0、.....、U'_n。注意 [6]:

$$R_{n-t}(A,P) = \left\| A_{1:m,c_{t+1}} - \operatorname{Span}\left(A_{1:m,U_t\setminus\{c_{t+1}\}}\right)\right\|_2 . \tag{2}$$

另一项重要性质是 $R(A,\ P)$ 的单调性。对于每个子集 U，定义$P_{|U}$ 为通过删除P中其 1 位于由 U 指定的行之外的列，再删除相应行后得到的 $|U|\times|U|$ 置换矩阵。剩余的行和列保持原有顺序。因此，我们有 [28]

$$R(A_{1:m,U}, P_{|U}) \geq R(A,P) .$$

我们证明，任何满足式 (1) 的置换 P' 都满足

$$R\left(P'\right) \geq \frac{1-\varepsilon}{1+\varepsilon}\cdot \max_{P} R\left(P\right)$$

由于引理1保证式 (1) 以至少 $1-\delta$ 的概率对$P_{\text{R-V-BLAST}}$ 成立，我们可知，若按照引理 3 的前提选择投影维数，我们的论断便得证。

我们对 n 使用归纳法证明该论断。当 $n=1$ 时，该论断显然成立（只有一列可供选择）。假设对于列数至多为 n 的所有矩阵，该论断均成立。设 A 是一个有 $n+1$ 列、 m 行的矩阵。设P' 为满足式 (1) 的置换。令 P 为任意置换，并考虑式 (1) 中的 $t=0$ 。利用式 (2)，并且由于 P 将某一列置于最后，式 (1) 意味着

$$R_{n+1}(A,P') \geq \frac{1-\varepsilon}{1+\varepsilon}\cdot R_{n+1}(A,P) .$$

由于公式 (1) 仅取决于集合 U_t ，而与先前主元的顺序无关，因此 $P'_{|U'_1}$ 对 $A_{1:m,\ U'_1}$ 也满足公式 (1)。因此，我们有

$$R(A_{1:m,U'_1}, P'_{|U'_1}) \geq \frac{1-\varepsilon}{1+\varepsilon}\cdot R(A_{1:m,U'_1}, P_{|U'_1}) \geq \frac{1-\varepsilon}{1+\varepsilon}\cdot R(A,P)$$

第一个不等式是归纳假设，第二个不等式体现了单调性。最后，

$$\begin{aligned} R(A,P') &= \min(R_{n+1}(A,P'), R(A_{1:m,U'_1}, P'_{|U'_1})) \\ &\geq \min\left(\frac{1-\varepsilon}{1+\varepsilon}\cdot R_{n+1}(A,P), \frac{1-\varepsilon}{1+\varepsilon}\cdot R(A,P)\right) \\ &= \frac{1-\varepsilon}{1+\varepsilon}\cdot R(A,P) \end{aligned}$$

□

6.1. 并行性与缓存未命中分析。现在，我们从关键路径和缓存未命中的角度分析第 5 节中的算法。我们同样忽略 $L = R^{-T}$ 的计算，因为相较于实际算法，它的跨度和缓存未命中次数都可以忽略不计。

该算法由 $\lceil \frac{n}{b} \rceil$ 个阶段组成，每个阶段都会确定顺序中最多b 列的位置。阶段 i处理剩余矩阵，其中包含

$n_i = n-(i-1)b$ 行和列，并将其缩减为尺寸为 $(b+p)\times n_i$的草图（最后一个阶段可能更小）。我们的性能分析聚焦于前$\lceil n/b \rceil/2$ 个阶段，在这些阶段中 $n_i = \Theta(n)$。每个阶段都由三个子阶段组成：

1. 算法生成一个随机（$b+p$）$\times n_i$ 算子，并将其应用于剩余的 $n_i \times n_i$ 矩阵，以生成草图。2. 算法在该草图上运行基于 Householder 变换的算法（来自第 4 节）的 b 步，同时选择并排列 b 列。3. 算法将 b 列（位于剩余子矩阵中，而非草图中）聚合为一个块反射子，就像分块（分区）QR 分解中所做的那样，然后将该块反射子应用于剩余子矩阵。

我们再次从并行性分析开始。无论使用稠密草图算子还是基于快速变换的算子，第一个子阶段的跨度都是Θ（$\log n$）。第二个子子阶段的跨度为 Θ（$b\log\ n$）；这是由我们对基于 Householder 变换的算法的分析得出的。第三个子阶段的跨度同样为 O（$b\log\ n$）。鉴于阶段数为 Θ（n/b），总跨度为 Θ（$n\log\ n$），渐近上与基于 Householder 变换的算法和降阶更新算法相同。

然而，该算法的缓存效率可能远高于基于 Householder 的算法和降阶更新算法。与之前一样，我们假设$M \leq n^2/8$，并且为简便起见，还假设 $p = O(b)$（令 $p = \Theta(b)\log\ n$ 会扰动后续估计，但这种扰动不足以证明增加的复杂性是合理的）。第一阶段和第三阶段都会访问整个 $n_i \times n_i$ 剩余子矩阵，因此在所有阶段中，它们总共会产生 $\Omega((n/b)n^2) = \Omega(n^3/b)$ 次缓存未命中。第三个子步骤应用块 Householder 反射，每个阶段需要 $\Theta(n_i^2 b/\sqrt{M})$ 次缓存未命中，整个算法则需要 $\Theta(n^3/\sqrt{M})$ 次。如果草图化算子是稠密的，将其与剩余子矩阵的面板相乘，每个阶段还会产生 $\Theta(n_i^2 b/\sqrt{M})$ 次缓存未命中。如果使用快速变换实现，缓存未命中的次数通常会更少，因此我们忽略这一项。无论哪种情况，第一阶段的成本都占主导地位。

现在，我们分析对每个草图应用基于 Householder 变换的算法所产生的缓存未命中次数。如果 $(b+p)n \leq cM$ 对于某个适当的$c < 1$成立，使得草图能够放入缓存的一部分空间中（这是基于面板的算法通常采用的参数选择），那么读取草图所需的缓存未命中次数$O((b+p)n)$占主导地位，因此整个算法在这些子阶段中的缓存未命中次数总数为 $O(n^2)$，显然是一个低阶项。如果草图无法放入缓存，那么我们可能需要在选择每一列之前读取（更新后的）草图，从而使缓存未命中次数上界增加 b 倍，达到 $O(bn^2)$。

T总之，如果草图能够放入缓存中，则缓存未命中次数总数为

$$\Theta\left(\frac{n^3}{b} + \frac{n^3}{\sqrt{M}} + n^2\right)$$

and

$$\Theta\left(\frac{n^3}{b} + \frac{n^3}{\sqrt{M}} + bn^2\right)$$

否则。

7. 实验结果。 实验在一台配备4核/8线程 2.60 GHz 英特尔酷睿 i5-1145G7 处理器和32 GB 内存的笔记本电脑上进行。我们使用 C语言实现了所有算法，并在适当情况下调用 BLAS [11, 12] 和 LAPACK [1]。我们将代码与英特尔数学核心函数库（MKL）链接，该函数库包含许多经过高度优化的 BLAS和LAPACK例程。

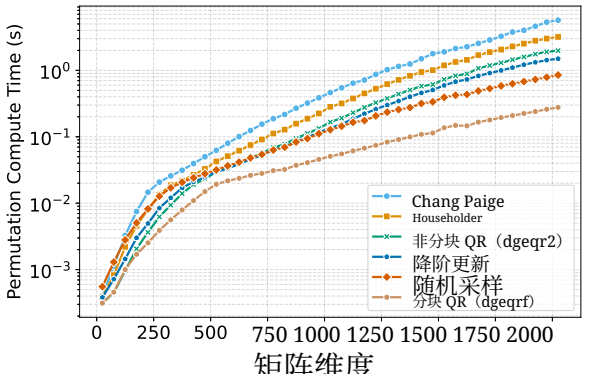


图 1。 精确和近似 *V-BLAST* 算法的平均运行时间，以及为校准而纳入的两种 *QR* 分解的平均运行时间。每个标记表示在不同高斯随机矩阵上运行 *100* 次所得的平均值。

我们使用的 MKL 版本属于英特尔 OneAPI 版本 2026.1。附录 A 介绍了有关我们实现的更多细节。以下实验使用调用 C 例程的 Python 代码完成。我们在单个核心上使用单线程运行代码，以便更容易解释运行时间和缓存未命中次数。

我们开展了两组实验。第一组实验将在下一节中介绍，旨在描述我们新算法相对于现有 V-BLAST 算法的性能（以及相对于相关基准方法的性能，例如简单的 QR 分解）。第二组实验分析了在完整的整数最小二乘求解器中，精确和近似 V-BLAST 排序的有效性。

7.1. 我们算法的效率。 我们比较了若干精确和近似 V-BLAST 算法的性能：

- 我们在第 4 节提出的新型基于 Householder 变换的算法，可计算精确的 V-BLAST 排序，
- 我们在第 5 节提出的新型随机投影算法，可生成 V-BLAST 排序的近似结果，
- Chang 和 Paige 提出的精确 V-BLAST 算法 [6]，
- 以及基于 Benesty、Huang 和 Chen 的研究提出的精确降阶更新算法 [4]。

我们自行实现了所有算法，因为没有找到这些算法现有的高性能实现。对于随机采样算法，我们选择了块大小 $b = 16$ 和过采样参数 $p = 6$。

这些算法的运行时间如图 1 所示。图中还列出了 LAPACK 的 dgeqr2（基于 BLAS 2 的 QR 分解）和 dgeqrf（基于 BLAS 3 的 QR 分解）例程的运行时间，供校准参考。我们的 Householder V-BLAST 算法在结构上与 dgeqr2 相似，因为它会计算 Householder反射并将其应用于矩阵。我们的重复随机采样算法在结构上与 dgeqrf 相似，因为二者都会对矩阵依次应用一系列块反射。

图 1 中展示的运行时间是对不同高斯独立同分布元素随机矩阵进行 100 次运行所得的平均值。总体而言，运行时间的方差相当小；我们在

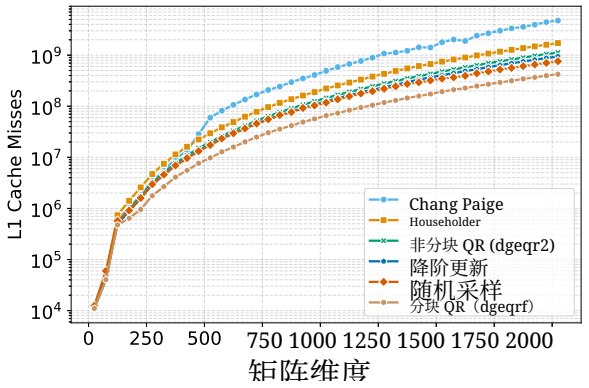


图 E 2. 精确和近似 *V-BLAST* 算法的 *L1* 缓存未命中平均次数。

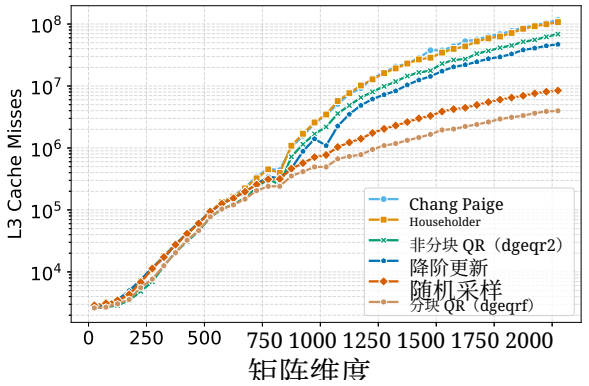


图 E 3. 精确和近似 *V-BLAST* 算法的 *L3* 缓存未命中平均次数。

附录 C。

图表显示，我们的 Householder 算法比 LA- PACK 的 dgeqr2 慢约 50%（后者在结构上与之相似，但不需要搜索范数极大的列）。Chang 和 Paige 的算法慢得多，比 dgeqr2 多耗时约 125%（运行时间超过后者的两倍）。

降阶更新算法比这三种算法都快：比 dgeqr2 快 25%。其运行时间约为我们的 Householder 算法的一半，但数值稳定性不如后者。

我们的新随机采样算法在维度大于约 750 的矩阵上比所有精确 V-BLAST 算法都快，在维度大于约 500 时比 Householder 算法快；但对于较小的矩阵，它并不是最快的。

图 2 和图 3 展示了各算法产生的平均缓存未命中次数。缓存未命中次数是通过在运行实验的进程上动态附加 Linux perf 工具，从 CPU 的硬件性能计数器中读取的。动态附加可确保这些计数仅反映算法执行期间发生的缓存未命中，而不包括设置或收尾阶段产生的未命中。图中展示了第一级和末级缓存（L1 和 L3）的未命中次数，这些数据相对容易通过 perf性能分析工具计数（L2 未命中则

更难计数）。

结果表明，正如所声称的那样，我们的随机采样算法产生的缓存未命中远少于精确 V-BLAST 算法。这一点在 L3计数 中尤为明显：随机化算法的缓存未命中曲线与三级基本线性代数子程序 dgeqrf 的曲线相近，而所有精确算法从约 800 的维度开始，其行为在定性上都明显不同，并开始与二级基本线性代数子程序 dgeqr2 的行为趋于一致。

L1图主要突出了 Chang 和 Paige 算法较差的缓存性能，该性能在约 500 的维度处显著下降。

7.2. 近似 V-BLAST置换的质量评估。接下来，我们介绍相关实验，以量化近似 V-BLAST置换 对整体整数最小二乘求解器的影响。我们评估两种近似方法：我们的随机采样算法，以及一种更简单的列主元QR算法 SQRD [30]。

我们在一个求解器中评估这三种排列，该求解器能为以下问题求得精确的最小化解（浮点舍入误差的影响除外）：

$$\hat{x} = \arg\min_{x \in \mathbb{Z}^n} \|Ax - b\|_2$$

其中， A 是一个元素服从标准高斯独立同分布的随机矩阵，且 $b = A \cdot y + e$，其中 y 是通过对一个标准高斯独立同分布随机向量进行舍入而生成的整数向量，而 e 是一个元素独立同分布、标准差给定的高斯噪声向量。系数矩阵 A 的分布选择意味着它以很高的概率具有满列秩。 e 的标准差选择会显著影响最小二乘问题的难度，正如我们下面将看到的。

求解器通过计算QR分解 $A = QR$、计算 R的精确或近似V-BLAST列置换 P 、计算QR分解 $AP = Q_1 R_1$，然后调用Schnorr-Euchner（SE）搜索求解以下问题，从而求得$\hat{x}$ ：

$$\begin{aligned}
\hat{x} &= \arg\min_{x \in \mathbb{Z}^n} \|Ax - b\|_2 \\
&= \arg\min_{x \in \mathbb{Z}^n} \left\|AP\left(P^T x\right) - b\right\|_2 \\
&= \arg\min_{x \in \mathbb{Z}^n} \left\|Q_1 R_1\left(P^T x\right) - b\right\|_2 \\
&= \arg\min_{x \in \mathbb{Z}^n} \left\|R_1\left(P^T x\right) - Q_1^T b\right\|_2 .
\end{aligned}$$

第一次QR分解的运行时间基本不受任何随机选择的影响，确定 P 的运行时间也基本不受这些选择的影响。但SE搜索的运行时间则强烈取决于 e 的标准差以及 P的质量。

图4探讨了元素 e 的标准差 σ 对总求解时间的影响。在约为 $\sigma = 0.15$之前，采用精确V-BLAST置换和我们的随机化算法时，运行时间都与 σ无关。在这一范围内，SE搜索非常快，QR 分解和置换的构造占据了主要运行时间。在这一范围内，更快但属于近似方法的随机化算法能够缩短求解时间。当 σ 超过0.15并继续增大时，搜索阶段的开销越来越大，运行时间迅速增加，而使用精确V-BLAST置换的求解器则变得更快。

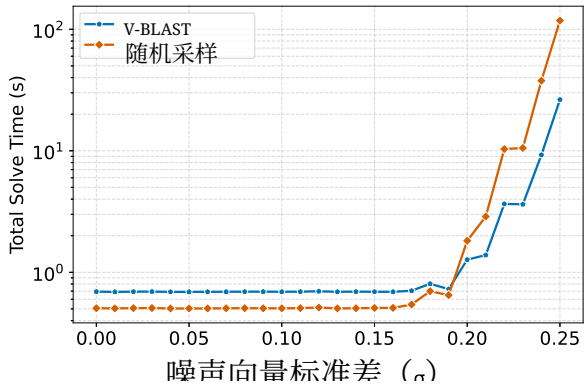


图4。 1500×1500 矩阵的平均总求解时间（包括置换计算和*SE*算法）与 e元素的标准差以及右端项 b 中的加性误差之间的关系。每个标记表示100 个矩阵的平均值。

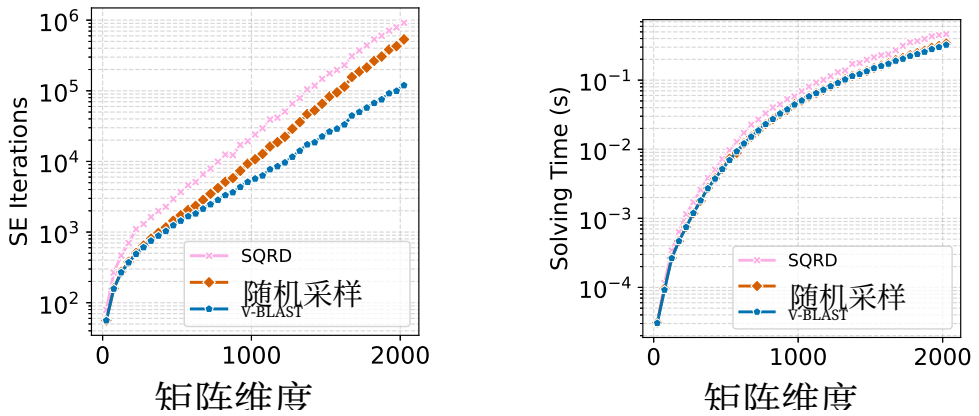


图5。 不包括置换计算时，求解器的*SE*迭代次数和总运行时间与问题维度之间的关系。每个标记表示 1000 次实验的几何平均值。右端项误差元素的标准差为 $\sigma = 0.15$。

因此，从这里开始，我们对元素 e采用固定的标准差 $\sigma = 0.15$ 。对于维数为 1500的问题，这是搜索仍然开销不大的最大噪声的近似值。

图5展示了对于三种排列（V-BLAST、SQRD 和我们的随机化算法），SE 迭代次数及求解器运行时间（不包括置换计算）如何随矩阵维度变化；这里所用的噪声向量中，各元素的标准差为 $\sigma = 0.15$。我们将 SE 迭代次数限制为 $25n^2$，以使运行时间保持在合理范围内。尽管从理论上讲，SE 可能需要指数级迭代次数，但噪声的标准差相对较小，因此在大多数运行中，SE 都能在此上限内收敛到一个最小化解。采用 V-BLAST 和我们的随机化排序时，至少 98% 的运行都能收敛；采用 SQRD 时，收敛率至少为 90%。本节稍后将更详细地探讨 SE 迭代次数，附录 C 中还会进一步详述。

数据表明，我们的随机化算法（块大小为 $b = 16$，过采样参数为 $p = 6$）产生的排列效果不如精确 V-BLAST

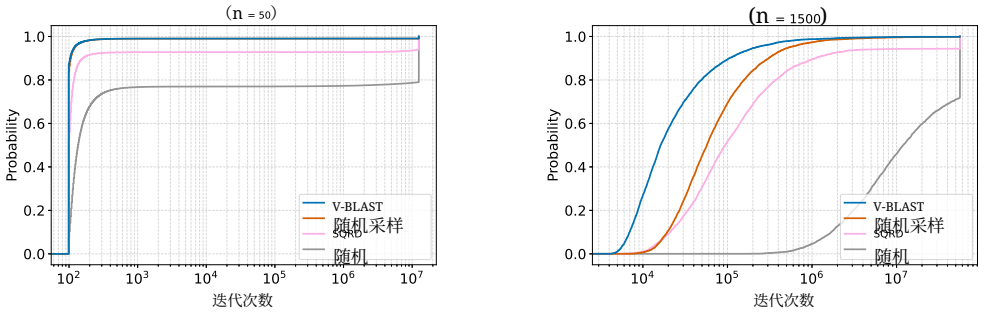


图 6。 对于维度为50 （左）和1500 （右）的矩阵，采用不同置换方法时 *SE* 所需迭代次数的经验累积分布函数。迭代次数分别限制为 $1.25 \cdot 10^7$ 和 $5.625 \cdot 10^7$。

但优于 SQRD。矩阵维度为 2000 时，SQRD 的迭代次数约为 V-BLAST 的 10 倍；这一倍数会随着矩阵维度的增大而增加。

运行时间的差异小得多。采用精确 V-BLAST 的求解器与采用随机化算法生成的置换的求解器，运行时间相近。采用 SQRD 置换的求解器则较慢。

在图5中，仅用（几何）均值表示迭代次数的分布，会掩盖一些重要细节。图6给出了两个问题维度（50 和 1500）下的经验累积分布。数据分别基于每个维度的 100,000 次和 5,000 次实验，噪声各元素的标准差为 $\sigma = 0.15$。

数据表明，更好的排序方法在两个方面各有优势。首先，在所有排序方法下，一定比例的问题实例中，SE 搜索的收敛速度都非常慢（图线看起来近乎平坦的区域）。对于 $n = 50$，在 1,000 次迭代内未收敛的大多数实例，在经过 10^7 次迭代后仍未收敛。如果不设定迭代次数上限，所有运行最终都会收敛（SE 会终止），但即使采用给定的排序，仍有一些问题实例难以求解。这些顽固实例所占比例在 V-BLAST 和随机采样中最小，在 SQRD 中大得多，而在随机列排序中最大。

其次，对于并不棘手的问题实例，V-BLAST 通常比随机化排序所需的迭代次数更少，而随机化排序所需的迭代次数又少于 SQRD。当图表中的迭代次数区域快速上升时，这一点尤为明显。

8. 结论。 Wilkinson 的《代数过程中的舍入误差》 *[27]* 出版 60 多年后，以及 Higham 的《数值算法的精度与稳定性》 [15],第 2 版出版近 25 年后，稠密矩阵计算算法的技术发展现状已经十分成熟。对于大多数问题，我们已经拥有具有三次复杂度的后向稳定算法，其关键路径（跨度）为 $O(n\log\ n)$，其中 n 为维数；这些算法能够很好地复用缓存和本地存储器中的数据，而且在许多情况下还采用了缓存无关算法和通信规避算法，使这些数据移动指标达到极高的渐近效率。许多最稳定的算法都基于正交变换。对于大多数问题，也存在次立方算法，

但它们通常不够稳定（而且在实践中很难实现快速运行）。

属于此类问题的 V-BLAST置换计算算法一直进展滞后。尽管在 2000 [14] 中发现了一种三次算法，并在 2007 [6], 中再次由他人独立发现，但这些算法的并行性、数据复用和数值稳定性均未得到分析。

本文改变了这一局面。我们分析了现有算法的并行性和数据复用，指出它们存在不足，并提出了两种全新算法，在并行性、稳定性和数据复用方面实现了更佳的平衡；我们还通过理论和实验验证了它们的有效性。

我们提出的一种新算法是精确算法，结果表明，它优于现有的精确 V-BLAST 算法。另一种是近似算法，但其近似效果非常好，性能甚至更优。因此，我们建议在可能的情况下，以其中一种算法取代现有算法。

我们的两种新算法都采用了借鉴自秩揭示QR分解算法的策略。这两个问题密切相关（但并不完全相同），因此我们得以将这些策略改用于 V-BLAST。由此得到的算法在结构上与相应的秩揭示QR分解算法相似。因此，我们相信，我们的新算法在并行硬件（多核CPU、GPU 等）上的性能将与相应的秩揭示 QR 分解算法相当。

最后，还有一类用于秩揭示 QR 分解的高性能算法，即基于锦标赛的算法 [10], ，但我们尚未找到可用于 V-BLAST 的对应算法。能否使用基于锦标赛的算法，高效地计算近似的 V-BLAST置换?

附录 A. 代码的实现细节与文档。

我们最初使用 Python 实现了所有算法，没有使用任何人工智能。这些实现注重清晰易懂，我们用它们来验证算法的正确性。随后，我们使用 Gemini 提升 Python 代码的性能，并将算法转换为 C语言。我们通过将所有高性能实现的输出与纯手写版本的输出进行比较，测试了这些实现。

我们的代码可在此处找到：https://github.com/TomerPelleg/V-BLAST。代码的主要组成部分如下：

- 列_顺序_算法目录：此目录包含各种 V-BLAST 算法、我们的重复采样算法、SQRD，以及 dgeqrf 和 dgeqr2 封装程序的 C语言和 Python 实现。它还包含项目的 CMake 文件，以及一个用于封装置换结果的 Python 置换类。确切的_vblast/main.py 中包含一些测试，用于检查不同函数是否会对其生成的矩阵产生相同的置换。这些测试对于确认我们正确理解了所有 V-BLAST 算法尤其有用，因为我们找不到这些算法的实现。

- schnorr_euchner 目录：此目录包含 Schnorr-Euchner 算法的 C语言和 Python 实现。
- experiments 目录：此目录包含用于运行实验的 Python 代码。其中包含用于生成矩阵和向量以及运行

端到端实验的代码。此外，它还包含一个用于实验的配置类，用于告知实验运行器应使用哪些参数，以及用于将 perf性能分析工具附加到每个进程的代码。visuals 子目录中包含用于生成本文所示图表的代码。configs 子目录包含用于生成这些图表的配置。

- tests 目录：项目的开发测试，基于 pytest 测试框架。

我们还列出了实现中使用的主要 BLAS 和 LAPACK 例程。为了计算 $L = R^{-T}$（这是多种算法共同的预处理步骤），我们使用 dgeqrf 计算 $A = QR$中的 R 因子，并使用 dtrtri 求三角矩阵的逆以计算 R^{-1}。我们使用 dswap 交换列。

- Chang 和 Paige V-BLAST算法：我们使用上述方法计算 $L = R^{-T}$。我们使用 drot 应用 Givens旋转。
- 降阶V-BLAST算法：我们使用 dsyrk 计算 A^*A。我们使用 dpotrf 和 dpotri 对 A^*A 进行 Cholesky分解并求其逆，从而得到 $M = (A^*A)^{-1}$。我们使用 dger 对 M进行 秩一降阶。

- Householder V-BLAST 算法（本文提出）：我们使用上述方法计算 $L = R^{-T}$。我们使用 dgemv 和 dger 计算 Householder 反射。

- 重复随机采样算法（本文提出）：我们使用上述方法计算 $L = R^{-T}$。我们使用 dgemm 将 L与随机矩阵 G相乘。与 Householder 算法一样，我们使用 dgemv 和 dger 在投影空间中应用 Householder 反射。我们使用 dnrm2 计算投影空间中各列的范数。为了构造块反射子，我们使用 dgemv、dger、dscal、dcopy 和 dnrm2，并使用 dgemm 应用这些反射子。实现方式与 dgeqrf 类似。

- SQRD算法：找到范数最小的列后，我们使用 dgemv 和 dger 计算所需的 Householder 反射。
- Schnorr-Euchner：我们使用 dgeqrf 和 dormqr 将具有一般矩阵的原始最小二乘问题 A 转换为具有三角矩阵的最小二乘问题，这是 Schnorr-Euchner 算法所要求的。

人工智能使用说明： 如上所述，高性能实现由 Gemini 生成，并以清晰的手写 Python 代码作为算法的规范。生成图表的代码也在一定程度上使用了人工智能编写。

附录 B. 其他辅助结果。

在本附录中，我们证明了证明第（6）节结果时所使用的一些额外技术引理。

回想一下， $S_t = [n] \setminus U_t$ 表示算法选出的前 t 列的索引集合。回想一下， $l_i^{(t)}$ 是将 l_i 投影到 Span $(l_j | j \in S_t)$ 的正交补上所得的正交投影。设 H 为一个酉矩阵，它汇集了所有n反射变换。这意味着 $H(LP)$是某个下三角矩阵 T。设 c_t为选定的 t列（即 $\{c_t\} = S_t \setminus S_{t-1}$）。 记 $h_1, \ldots, h_n$为 H的各行转置后得到的列向量（回想一下，H 作用于 L，其维数为 n×n）。由于H是酉矩阵，我们有 $LP = H^T T$，或者

$$(LP)_{1:n,i} = \sum_{j=1}^{n} T_{j,i} \cdot h_j$$

引理 4。

$$h_{n-t+1} \parallel l_{c_t}^{(t-1)}$$

证明。注意，c_t、$l_{c_t} = (LP)_{1:n,\ n-t+1}$我们对 t 使用归纳法进行证明。对于基础情形，首先注意到，$H(LP)$ 的最右列中的顶部 $n-1$ 个元素为零，因此位于 n, n 处的最底部元素，其绝对值等于 $H(LP)$ 中 n 列的范数。然而，H 是酉矩阵，因此该范数等于 LP 中列 n 的范数。此外，该元素等于 H 的第 n 行（即 h_n）与 LP 的第 n 列的点积，因此 $|h_n^T \cdot (LP)_{1:n,\ n}|$ 等于 $(LP)_{1:n,\ n}$ 的范数。现在，由于 h_n 的范数为 1（因为它是酉矩阵的一列），我们得出 $h_n \parallel l_{c_1}^{(0)}$。

由于 $LP = H^T T$，且 T 是下三角矩阵

$$(LP)_{1:n,n-t+1} = \sum_{j=n-t+1}^{n} T_{j,n-t+1} \cdot h_j$$

其中用到了该列顶部元素为 0 这一事实。根据归纳法，$\sum_{j=n-t+2}^{n} T_{j,\ n-t+1} \cdot h_j$ 属于 Span $(l_j | j \in S_{t-1})$。由于 h_{n-t+1} 与所有h_{n-t+2}，…，h_n正交，这意味着 $T_{n-t+1,\ n-t+1} \cdot h_{n-t+1}$是 $(LP)_{1:n,\ n-t+1}$ 在 Span $(l_j | j \in S_{t-1})$ 的正交补上的正交投影，根据定义即为 $l_{c_t}^{(t-1)}$。 □

推论5. $l_i^{(t)} \in$ 跨度 $(h_1, \ldots, h_{n-t})$

令 w 为 b的倍数。回想一下，该算法会聚合 Householder变换，并在这些步骤中应用它们；我们将选取 w 列后得到的聚合变换记为 H_w。容易看出，H 的底部 w行就是 H_w的底部 w 行。因此，我们可以写成

$$H_w = \begin{bmatrix} \tilde{H}_w \\ H_{n-w+1:n,1:n} \end{bmatrix}$$

根据定义

$$r_i^{(w)} = \tilde{H}_w l_i$$

引理。$\|r_i^{(w)}\|_2 = \|l_i^{(w)}\|_2$。

证明。记 $l_i = \sum_{j=1}^{n} \alpha_{ij} h_j$（即将 l_i 按基 h_1，…，h_n展开）。根据推论5以及 h_1，…，h_n 是标准正交基这一事实，我们得出$l_i^{(w)} = \sum_{j=1}^{n-w} \alpha_{ij} h_j$.我们得出

$$\|l_i^{(w)}\|_2^2 = \sum_{j=1}^{n-w} |\alpha_{ij}|^2 = \sum_{j=1}^{n} |\alpha_{ij}|^2 - \sum_{j=n-w+1}^{n} |\alpha_{ij}|^2 = \|l_i\|_2^2 - \sum_{j=n-w+1}^{n} |\alpha_{ij}|^2$$

另一方面,

$$H_w l_i = \begin{bmatrix} r_i^{(w)} \\ (H_{n-w+1:n,1:n})\, l_i \end{bmatrix} = \begin{bmatrix} r_i^{(w)} \\ \alpha_{i,n-w+1} \\ \vdots \\ \alpha_{i,n} \end{bmatrix}$$

由于 H_w 是酉矩阵，

$$\|l_i\|_2^2 = \|H_w l_i\|_2^2 = \|r_i^{(w)}\|_2^2 + \sum_{j=n-w+1}^{n} |\alpha_{ij}|^2$$

结论得证。□

引理。记 $q_i^{(t)}$ 为 $r_i^{(w)}$ 在 $\mathit{Span}\left(r_j^{(w)} | j \in V_t\right)$ 上的正交投影。由于 $r_i^{(w)} = (H_w l_i)_{1:n-w}$ 且 H_w 是正交的，因此有 $\left\|l_i^{(t)}\right\|_2 = \left\|r_i^{(w)} - q_i^{(t)}\right\|_2$。

证明。设 W_t 为按形成 H时所用的相同顺序，对 L中的当前块进行处理时使用的前 $t-w$ 个 Householder 反射的乘积。因此， W_t 的大小为 $(n-w) \times (n-w)$。定义 $\tilde{H}_t$ 为 $W_t H_w$的前 $n-t$ 行。后续反射变换保持其余行不变，因此有

$$W_t \tilde{H}_w = \begin{bmatrix} \tilde{H}_t \\ H_{n-t+1:n-w,1:n} \end{bmatrix}$$

$$W_t r_i^{(w)} = W_t(\tilde{H}_w l_i) = \begin{bmatrix} \tilde{H}_t l_i \\ H_{n-t+1:n-w,1:n} l_i \end{bmatrix} = \begin{bmatrix} \tilde{H}_t l_i \\ \alpha_{i,n-t+1} \\ \vdots \\ \alpha_{i,n-w} \end{bmatrix}$$

其中 $\alpha_{i,\ n-t+1}$，……，$\alpha_{i,\ n-w}$ 来自展开式 $l_i = \sum_{j=1}^{n} \alpha_{ij} h_j$（见前面的证明）。因此，我们只需证明

$$q_i^{(t)} = W_t^T \begin{bmatrix} 0 \\ \alpha_{i,n-t+1} \\ \vdots \\ \alpha_{i,n-w} \end{bmatrix}$$

这是因为

$$W_t \ \mathrm{Span}\left(r_j^{(w)} | j \in V_t\right) = \{0\}^{n-t} \times \mathbb{R}^{t-w}.$$

因此， $q_i^{(t)}$ 的上述表达式是到 $\mathrm{Span}\left(r_j^{(w)} | j \in V_t\right)$ 的正交投影。根据正交性和推论5，

$$\left\|r_i^{(w)} - q_i^{(t)}\right\|_2 = \left\|\tilde{H}_t l_i\right\|_2 = \left\|l_i^{(t)}\right\|_2.$$ □

附录 C。其他实验结果。

本节包含一些未收录在论文正文中的图表。

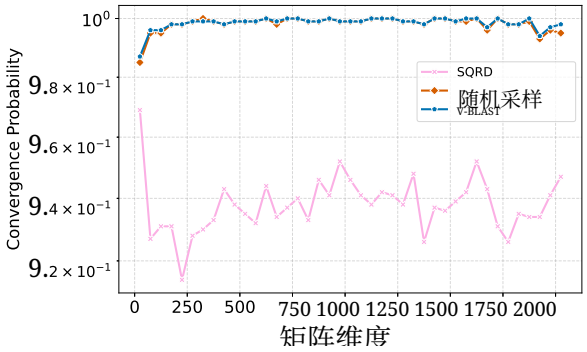


图 7. *SE* 在达到最大迭代次数之前收敛的实验百分比 $25 \cdot n^2$

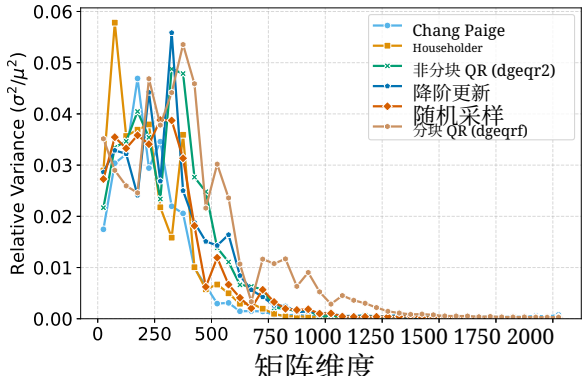


图 8。 图 *1* 中用于校准的两种 *QR* 分解，以及精确和近似 *V-BLAST* 算法运行时间的方差。每个标记表示在不同高斯随机矩阵上运行 *100* 次所得的平均值。结果以平均运行时间归一化，使其成为无量纲值。

致谢。 我们感谢 Xiao-Wen Chang 就 V-BLAST 算法开展的启发性讨论，并感谢他向我们发送了他与 Chris Paige 共同开发的该算法实现。在准备本文的过程中使用了人工智能，主要用于形式化证明、修正证明中的问题以及编写实验代码。第6节中各项主张的陈述和证明由作者准备，期间还与人工智能进行了多次交流，以改进并形式化相关文本。研究结果也使用 Claude 在 Lean 中进行了验证。除少量语法修改外，整篇手稿均由作者撰写。作者对全部内容承担责任。

参考文献

[1] E. Anderson、Z. Bai、C. Bischof、L. S. Blackford、J. Demmel、J. Dongarra、J. Du Croz、A. Greenbaum、S. Hammarling、A. McKenney 和 D. Sorensen，《LAPACK 用户指南》，1999，https://doi.org/10.1137/1.9780898719604。[2] L. Babai，《关于 Lovász 格约减与最近格点问题》，Combinatorica，6（1986），第 1–13 页，https://doi.org/10.1007/bf02579403。[3] G. Ballard、J. Demmel、O. Holtz 和 O. Schwartz，《数值线性代数中的通信最小化》，SIAM 矩阵分析与应用杂志，32（2011），

第 866–901 页，https://doi.org/10.1137/090769156。[4] J. Benesty、Y. Huang 和 J. Chen，《BLAST系统中的最优顺序信号检测快速递归算法》，IEEE信号处理汇刊，51（2003），第 1722–1730 页，https://doi.org/10.1109/tsp.2003.812897。[5] R. D. Blumofe、C. F. Joerg、B. C. Kuszmaul、C. E. Leiserson、K. H. Randall 和 Y. Zhou，《Cilk：一种高效的多线程运行时系统》，《并行与分布式计算杂志》，37（1996），第 55–69 页，https://doi.org/10.1006/jpdc.1996.0107。[6] X.-W. Chang 和 C. C. Paige，《给定向量集的欧几里得距离与最小二乘问题》，《应用数值数学》，57（2007），第 1240–1244 页，https://doi.org/ 10.1016/j.apnum.2007.01.008。[7] X.-W. Chang、J. Wen 和 X. Xie，《LLL约化对 Babai点成功概率及球译码复杂度的影响》，IEEE信息论汇刊，59（2013），第 4915–4926 页，https://doi.org/10.1109/tit.2013.2253596。[8] M. Damen、H. El Gamal 和 G. Caire，《关于最大似然检测与最近格点的搜索》，IEEE信息论汇刊，（2003），第 2389–2402 页，https://doi.org/10.1109/tit.2003.817444。[9] J. Demmel、L. Grigori、M. Hoemmen 和 J. Langou，《通信最优的并行与顺序 QR 分解和 LU 分解》，《SIAM科学计算杂志》，34（2012），第 A206–A239 页，https://doi.org/10.1137/080731992。[10] J. W. Demmel、L. Grigori、M. Gu 和 H. Xiang，《带列主元选取的通信规避秩揭示 QR 分解》，SIAM 矩阵分析与应用杂志，36（2015），第 55–89 页，https://doi.org/10.1137/13092157X。[11] J. J. Dongarra、J. Du Croz、S. Hammarling 和 I. S. Duff，《一组三级基本线性代数子程序》，《ACM数学软件汇刊》，16（1990），https: //doi.org/10.1145/77626.79170。[12] J. J. Dongarra、J. Du Croz、S. Hammarling 和 R. J. Hanson，《一组扩展的 FORTRAN 基本线性代数子程序》，《ACM数学软件汇刊》，14（1988），https://doi.org/10.1145/42288.42291。[13] J. A. Duersch 和 M. Gu，《带列主元选取的随机化 QR》，《SIAM科学计算杂志》，39（2017），第 C263–C291 页，https://doi.org/10.1137/15M1044680。[14] B. Hassibi，《一种高效的 BLAST 平方根算法》，载于《2000 IEEE声学、语音与信号处理国际会议论文集》（目录号：00CH37100），ICASSP-00 第 2 卷，2000，第 II737–II740 页，https://doi.org/10.1109/icassp.2000.859065。[15] N. J. Higham，《数值算法的精度与稳定性：第二版》，2002，https: //doi.org/10.1137/1.9780898718027。[16] B. Hochwald 和 S. ten Brink，《在多天线信道上实现接近信道容量的性能》，IEEE通信汇刊，51（2003），第 389–399 页，https://doi.org/10.1109/ tcomm.2003.809789。[17] J.-W. Hong 和 H. T. Kung，《输入/输出复杂度：红蓝石子游戏》，载于《第十三届 ACM计算理论研讨会论文集——STOC ’81》，81，1981，326 333，10 1145 800076 802486 STOC ’ 第 – 页 https://doi.org/ . / . . [18] A. K. Lenstra、H. W. Lenstra 和 L. Lovász，《具有有理系数的多项式分解》，《数学年刊》，261（1982），第 515–534 页，https://doi.org/10.1007/ bf01457454。[19] Z. Luo、S. Liu、M. Zhao 和 Y. Liu，《V-BLAST系统的新型快速递归 MMSE-SIC检测算法》，IEEE无线通信汇刊，6（2007），第 2022–2025 页，https://doi.org/10.1109/twc.2007.05806。[20] C. P. Schnorr 和 M. Euchner，《格基约化：改进的实用算法与子集和问题求解》，《数学规划》，66（1994），第 181–199 页，https: //doi.org/10.1007/bf01581144。[21] R. Schreiber 和 C. Van Loan，《Householder变换乘积的节省存储空间的 WY 表示》，《SIAM科学与统计计算杂志》，10（1989），第 53–57 页，https://doi.org/10.1137/0910005。[22] Y. Shang 和 X.-g. Xia，《采用最优排序SIC检测的 V-BLAST 快速递归算法》，IEEE无线通信汇刊，8（2009），第 2860–2865 页，https://doi.org/10.1109/twc.2009.081466。[23] L. Szczeciński 和 D. Massicotte，《MIMO MMSE接收机的低复杂度自适应：实现方面》，载于《GLOBECOM ’05：IEEE全球电信会议》，2005，https://doi.org/10.1109/glocom.2005.1578079。[24] S. Toledo，《数值线性代数中的外存算法综述》，1999，https://doi. org/10.1090/dimacs/050/09。[25] O. Uziel、E. Fogel、D. Halperin 和 S. Toledo，《非线性混合整数算法》

位置估计，2025，https://doi.org/10.48550/ARXIV.2505.12980。[26] J. Wen 和 X.-W. Chang，盒约束整数线性模型中 *Babai* 估计量的成功概率，IEEE信息论汇刊，63（2017），第 631–648 页，https://doi.org/10.1109/tit.2016.2627082。[27] J. H. Wilkinson，代数过程中的舍入误差，1963，https://doi.org/10.1137/1. 9781611977523。SIAM 于 2023 年再版。[28] P. Wolniansky、G. Foschini、G. Golden 和 R. Valenzuela，*V-BLAST*：一种用于在丰富散射无线信道上实现极高数据速率的架构，载于 1998 URSI 国际信号、系统与电子学研讨会，会议论文集（Cat. No.98EX167），ISSSE-98，1998 年，第 295–300 页，https://doi.org/10.1109/issse.1998.738086。[29] D. P. Woodruff，草图化作为数值线性代数的工具，理论计算机科学基础与趋势，10（2014），第 1–157 页，https://doi.org/10.1561/0400000060。[30] D. Wübben、R. Böhnke、J. Rinas、V. Kühn 和 K. Kammeyer，用于解码分层空时码的高效算法，Electronics Letters，37（2001），第 1348–1350 页，https: //doi.org/10.1049/el:20010899。[31] P. Xu、C. Shi 和 J. Liu，用于 *GPS* 模糊度解算的整数估计方法：面向应用的综述与改进，Survey Review，44（2012），第 59–71 页，https: // doi.org/10.1179/1752270611y.0000000004。