arXiv:2301.00074v1 [cs.CC] 30 Dec 2022

矩阵乘法：验证强唯一可解谜题*

Matthew Anderson、Zongliang Ji 和 Anthony Yang Xu

Union College 计算机科学系 斯克内克塔迪，纽约州，美国{andersm2、jiz、xua}@union.edu

摘要。 Cohn 和 Umans 提出了一种框架，通过将计算嵌入某些群代数中来开发快速矩阵乘法算法 [12]。在随后与 Kleinberg 和 Szegedy 的合作中，他们将这一框架与寻找称为强唯一可解谜题（强 USP）的组合对象联系起来 [11]。我们开始对这类对象进行系统的计算机辅助搜索。我们开发并实现了基于归约到可满足性问题（SAT）和整数规划（IP）的基于约束的算法，以验证谜题是否为强 USP，并搜索较大的强 USP。我们给出了宽度为 $k \leq 5$时强 USP 最大规模的紧确界，构造了宽度较小且规模超过以往工作的谜题，并改进了 $k \leq 12$的强 USP 规模上界。虽然我们的工作只涉及宽度为小常数的谜题，但我们找到的强 USP 推出了运行时间为 $O(n^{\omega})$ 且指数为 $\omega \leq 2.66$的矩阵乘法算法。尽管我们的算法并未超越最快的算法，但我们的工作提供了证据，也可能为寻找强 USP 族指明一条途径，从而推出比目前已知算法更高效的矩阵乘法算法。

关键词： 矩阵乘法 · 强唯一可解谜题 · 算术复杂性 · 整数规划 · 可满足性 · 可满足性基准 · 上界 · 归约 · 应用

1 引言

尽管人们为寻找矩阵乘法的最优算法付出了巨大努力，这一问题仍未解决。我们关注矩阵乘法问题的方阵形式，即给定域 $\mathcal{F}$ 上的两个 $n \times n$ 矩阵 A 和 B，目标是计算矩阵乘积 $C = A \times B$。一个悬而未决的重要问题是：计算 C 需要多少次域运算？长期以来，人们认为基于矩阵乘积定义的朴素算法是最优的，其运行时间为 $O(n^3)$。Strassen 的开创性工作表明，采用分治方法可以在 $O(n^{2.808})$[30] 时间内完成计算。此后一系列研究不断降低运行时间，最终得到 Coppersmith-Winograd算法（CW），将运行时间

* 本文的扩展摘要曾发表于 SAT 2020 [5] 会议论文集。

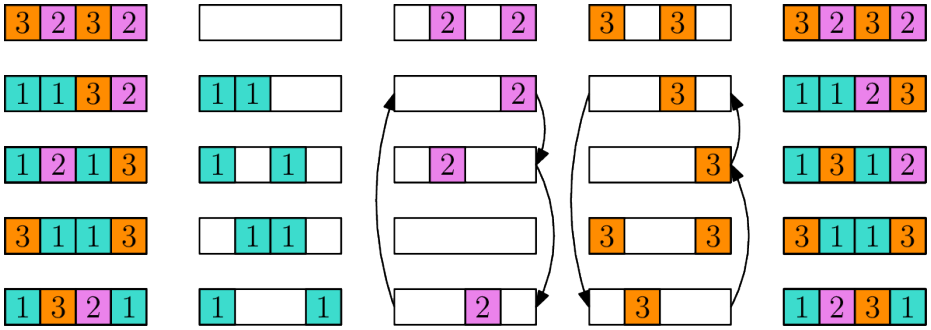

图 1：最左侧的示意图是一个宽度为 4、规模为 5 的谜题 P。中间的三个示意图是 P 的三个子行集合。最右侧的示意图是将 P 的子行按箭头所示重新排列并重新组合后得到的谜题 P'。由于 P 可以在不重叠的情况下重排为 $P' \neq P$，因此 P 不是唯一可解的。

至 $O(n^{2.376})$ [26,28,31,13]。近年来，其他研究者借助计算机对 CW 进行了改进，将指数降低至 $\omega \leq 2.3728639$ [16,32,22]。

Cohn 和 Umans 提出的方法 [12]通过将矩阵乘法问题归约为搜索具有满足一种称为 三重积性质的子集的群，从而建立了一个开发更快矩阵乘法算法的框架；这种性质使得矩阵乘法能够嵌入群代数。他们的方法受到了用于相乘次数为 n 的单变量多项式的 O（n 对数 n）算法的启发；该算法通过嵌入快速傅里叶变换的群代数来实现，参见例如 [14，第 30] 章。后续工作 [11] 进一步阐述了这一思路，并提出了称为强唯一可解谜题 （强 USP）的组合对象概念。这些对象为矩阵乘法提供了群代数嵌入，因此也由此得到一种矩阵乘法算法。

宽度为 k 的谜题 P 是 $\{1,2,3\}^k$ 的一个子集，而 P 的基数就是该谜题的大小。 P 中的每个元素都称为 P 的一行，每一行由三个子行组成，分别是 $\{1,*\}^k$、 $\{2,*\}^k$、 $\{3,*\}^k$ 的元素。非正式地说，如果无法通过置换 P 的子行来构造一个不同的谜题 P'，且没有数字重叠的单元格，那么谜题 P 就是一个唯一可解拼图（USP）。图 1 展示了一个不是 USP 的谜题。若唯一可解拼图满足更严格的不重叠条件（见定义 3），则称其为强 USP。对于固定宽度 k，强 USP 的大小越大，它所对应的矩阵乘法算法就越快 [11]。事实上，Cohn 等人证明，存在一个强 USP 的无限族，可达到 $\omega < 2.48$。

我们遵循 Cohn 等人的方案，开发了：**(i) 验证算法**和启发式方法，用于确定一个谜题是否为强 USP；**(ii) 搜索算法**，以寻找规模较大的强 USP；**(iii) 这些算法的实用实现** [1]。

[1] 源代码可在此处获取：https://bitbucket.org/paraphase/matmult

算法；以及 (iv) 关于强 USP 大小的新版**上界**。我们最成功的验证算法通过三维匹配将问题归约为可满足性（SAT）和整数规划（IP）问题，再利用现有工具求解。我们开发的算法效率不高——在最坏情况下，其运行时间关于自然参数呈指数级。然而，我们的目标是找到规模足够大的强 USP，以便得到更快的矩阵乘法算法；所得算法的运行时间与我们这些算法的运行时间无关。我们算法效率低下，因此能够实际检查的搜索空间受到限制。

结果 我们的理论结果和实现为小宽度谜题中最大强 USP 的规模给出了新的界。对于小常数宽度，$k \leq 12$，我们超过了 [11, 命题3。8]所给出的最大规模。我们的最大规模下界由通过搜索找到的强 USP 见证。对于 $k \leq 5$，我们通过在消除共同对称性后穷举搜索所有谜题，给出了紧确的上界。对于 $k \leq 12$，我们改进了强 USP 规模的上界。尽管目前的结果尚未超过 [11] 无界 k时的结果，但它们表明，可能存在一些强 USP 家族，能够给出比目前已知算法更高效的矩阵乘法算法。我们能够构造出的最佳强 USP 所蕴含的矩阵乘法算法具有 $\omega \leq 2.66$。

我们还基于强 USP 验证的归约构建了一个可满足／不可满足实例基准数据集，并考察了 2021 年 SAT 竞赛 [6]中求解器的性能。

相关工作 关于矩阵乘法问题算法方面的背景，可参见例如 [9]。目前也已知一些否定性结果。直观来看，输出矩阵 C 的维度意味着该问题至少需要 $\Omega(n^2)$ 的时间。一般情形下以及针对专门计算模型，也已知稍好的下界，可参见例如 [29,20]。此外，针对多种矩阵乘法算法方法，也已知一些下界。Ambainis 等人证明，单靠激光法无法得到一个具有 $\omega \leq 2.3078$ [4] 的算法。帽集等差数列方面的一项最新突破 [15]，结合关于 Erd¨os-Szemeredi 向日葵猜想的一项条件性结果 [3]，意味着 Cohn 等人的强 USP 方法在某些 $\epsilon > 0$ [10] 下无法达到 $\omega = 2 + \epsilon$。后续工作将这一障碍推广到了更大一类算法技术 [1,2]。尽管如此，据我们所知，这些否定性结果并未推出关于 ϵ 的具体下界。LeGall、Williams 和 Stothers 通过正向改进所取得的成果，与强 USP 方法无法证明 $\omega = 2$ 之间，仍存在着巨大的认知差距。

最近，Fawzi 等人展示了如何利用强化学习技术开发新的矩阵乘法算法 [17]。他们的工作生成了具有 $\omega < 2.77$ 的矩阵乘法算法，其速度快于 Strassen 的

原始算法（$\omega < 2.81$），但远不及 Coppersmith-Winograd 的改进（$\omega < 2.372$）或本工作所取得的结果。

组织 第 2 节 首先给出强 USP 和 Cohn-Umans 框架的正式定义。第3 &4节分别讨论我们用于验证某个谜题是否为强 USP 的算法和启发式方法，以及如何搜索此类谜题。第 5 节 介绍了强 USP 大小的若干上界。第 6& 7 节讨论了我们的实现和实验结果。

2 预备知识

对于一个整数 k，我们用 $[k]$ 表示集合 $\{1, 2, \ldots, k\}$。对于集合 Q，Sym$_Q$ 表示作用在 Q元素上的对称群，即作用于 Q的置换群。Cohn 等人提出了谜题 [11]的概念。

定义 1（谜题）。对于 $s, k \in \mathbb{N}$，一个(s, k)-谜题 是一个满足$|P| = s$的子集 $P \subseteq [3]^k$。我们称 s 为 P的大小，称 k 为 P的宽度。
我们称一个（s， k）-谜题有 s 行和 k 列。谜题的列具有固有的顺序，并按 $[k]$ 编号。谜题的行没有固有顺序，但通常为方便起见，假设它们按自然数集合 $[s]$ 排序并编号。

Cohn 等人确立了谜题的一种特定组合性质，使得可以推导出能够高效嵌入矩阵乘法的群代数。此类谜题称为强唯一可解谜题。不过，为了便于理解，我们首先解释一种更简单的性质，称为唯一可解拼图。

定义 2（唯一可解拼图（USP））。一个（s、 k）-谜题 P 若对于所有 π_1、 π_2、 $\pi_3 \in \mathrm{Sym}_P$：则要么（*i*） $\pi_1 = \pi_2 = \pi_3$，要么（*ii*）存在 $r \in P$ 和 $c \in [k]$，使得以下至少两个条件成立：（π_1（r））$_c = 1$、（π_2（r））$_c = 2$、（π_3（r））$_c = 3$。

通俗地说，如果谜题的每一行都能拆分成一号、二号和三号棋子，然后以另一种方式重新组合这些行，使每个新行都由一个一号、一个二号和一个三号棋子组成，并且 $[3]$ 的每一列恰好有一个元素，那么这个谜题就**不是**唯一可解的。注意，唯一可解的谜题最多只能有 2^k 行，因为每个一号棋子、二号棋子和三号棋子都必须是唯一的，否则就可以交换重复的棋子，使谜题变得不再唯一可解。

下面给出了强唯一可解拼图的定义。它与原定义几乎相同，不同之处在于，它要求某一列中恰好有两个棋子发生碰撞，而原定义允许两个或更多棋子发生碰撞。

定义 3（强唯一可解谜题（SUSP））。若一个(s, k)谜题 P 满足：对于所有 $\pi_1, \pi_2, \pi_3 \in \mathrm{Sym}_P$，要么 ***(i)*** $\pi_1 = \pi_2 = \pi_3$，要么 ***(ii)*** 存在$r \in P$ 和 $c \in [k]$，使得以下条件恰有两个成立：$(\pi_1(r))_c = 1, (\pi_2(r))_c = 2, (\pi_3(r))_c = 3$，则称其为强唯一可解。

最后，Cohn 等人定义了 SUSP 的一种强化形式，要求每三行都见证必要的重叠。

定义 4（局部 SUSP）。局部强唯一可解谜题是一个（s, k）-谜题，其中对于每组三行 $u, v, w \in P$，只要 u, v, w 不全相等，就存在 $c \in [k]$，使得 (u_c, v_c, w_c) 属于

$$\mathcal{L} = \{(1,2,1), (1,2,2), (1,1,3), (1,3,3), (2,2,3), (3,2,3)\}.$$

每个 SUSP P 都对应一个大得多的局部 SUSP P'；通俗地说，它是通过串接和复制 P 的行得到的，目的是明确展示 $\forall \pi_1$、π_2、π_3 部分定义 3。

命题 1（[11, 命题 6。3]）设 P 是一个 $(s,\ k)$-*SUSP*，则存在一个局部 $(s!\ s \cdot k)$-*SUSP* P'。

请注意，在所有定义中，局部性、强可解性和唯一可解性均不受谜题各行排列顺序的影响，因为 P 是一个集合——我们在此处隐式地使用了这一事实。

Cohn 等人展示了强 USP 的存在性与矩阵乘法指数上界之间的以下联系 ω。

引理 1 ([11, 推论 3。6]**)**。设 $\epsilon > 0$，若存在一个强唯一可解(s, k)谜题，则存在一种算法，可在时间$O(n^{\omega+\epsilon})$ 内完成 n×n 矩阵乘法，其中

$$\omega \leq \min_{m \in \mathbb{N}_{\geq 3}} \left(\frac{3 \log m}{\log(m-1)} - \frac{3 \log s!}{s \cdot k \log(m-1)} \right).$$

这一结果促使人们寻找规模较大的强 USP，以实现更快的矩阵乘法算法。在同一篇文章中，作者还证明了存在一个强唯一可解谜题的无限族，其宽度 k 可被 3 整除，并且该族对 ω 达到了非平凡上界。

引理 2 ([11, 命题3。8]**)**。存在一个强唯一可解谜题的无限族，可达到 $\omega < 2.48$。

最后，他们猜想强唯一可解谜题能够为实现二次时间矩阵乘法提供一条途径。然而，正如引言中所述，这一猜想已被证明是错误的。

引理 3 ([10]**)**。强唯一可解谜题不可能显示 $\omega < 2 + \epsilon$，其中 $\epsilon > 0$。

话虽如此，即使无法达到 $\omega = 2$，唯一可解拼图方法仍有望超越 Coppersmith-Winograd 的改进版本。

算法 1：穷举验证

输入： 一个 $(s,\ k)$-谜题 P。
输出： 如果 P 是强 USP，则输出“是”，否则输出“否”。

函数 VerifyBruteForce(P)
  对于 $\pi_2 \in$对称群$_P$ **执行**
    对于 $\pi_3 \in$对称群$_P$ **执行**
      如果 $\pi_2 \neq 1 \vee \pi_3 \neq 1$ **则**
        $found = false.$
        对于 $r \in P$ **执行**
          对于 $i \in [k]$ **执行**
            如果 $\delta_{r_i,\ 1} + \delta_{(\pi_2\ (r))_i,\ 2} + \delta_{(\pi_3\ (r))_i,\ 3} = 2$ **则** $found = true$。
        如果不是 $found$，**则返回** NO。
  返回 YES。

3 验证强 USP

本文的核心关注点是验证强 USP 的问题，即给定一个 (s, k)-谜题 P，若 P是强 USP，则输出 YES，否则输出 NO。本节将讨论如何根据自然参数 s 和 k设计算法来解决这一计算问题。

本节开发的所有精确算法，其最坏情况运行时间都是指数级的。不过，渐近最坏情况运行时间并非我们真正关注的指标。我们关注的是算法的实际性能，以及它们发现新的大型强 USP 的能力。我们最终开发的算法是由若干较简单的算法和启发式方法组成的混合型算法。

我们首先讨论一种基于强 USP 定义的朴素¨暴力算法（第 3.1 小节），了解它如何启发我们归约到三维匹配问题（第 3.2 小节），然后探讨如何将归约表述为可满足性问题和整数规划问题（第 3.4和 3.5 小节）。接着，我们介绍几种基于强 USP 性质的验证启发式方法（第 3.6 小节），并将它们与验证算法结合，得到一种混合算法 Verify （第 3.7 小节）。正如我们在第 7.2 小节所述，我们的混合算法能够快速检验给定谜题是否为强 USP，并有助于寻找强 USP。

3.1 暴力搜索

验证的显然算法直接来自强 USP 的定义。非正式地说，我们考虑相对于一号棋子置换二号和三号棋子的所有方式，并检查 定义 3 中的不重叠条件是否满足。该算法的形式化描述见算法 1。

算法 1 第 4 行中的 ones 表示 Sym_P 中的恒等置换，而 $\delta_{a,b}$ 是克罗内克δ函数：当 $a = b$ 时其值为 1，否则为 0。请注意，算法 1 并未涉及定义 3 的 π_1。这是因为强 USP 性质对行的置换具有不变性，因此 π_1 可以被视为任意相位。于是，我们固定 $\pi_1 = 1$ 以简化算法。鉴于 $|\mathrm{Sym}_P| = s!$，我们得出该算法的运行时间为 $O((s!)^2 \cdot s \cdot k \cdot \mathrm{poly}(s))$，其中最后一个因子计入了对 s个元素的置换进行操作的开销。运行时间的主导项来自遍历所有置换对。最后请注意，如果 P是强 USP，则算法的运行时间为 $\Theta((s!)^2 \cdot s \cdot k \cdot \mathrm{poly}(s))$；而如果 P不是强 USP，算法会提前终止。然而，该算法性能不佳，因而无法在我们的实现中使用；不过，它的简单性以及与定义的直接联系，使其实现成为针对后续更复杂算法的宝贵合理性检查（同时也为参与该项目的本科生提供了有效的入门实践）。

尽管算法 1 的表现不佳，但考察一个看似简单的优化的结构，能带来效率高得多的算法。考虑定义在三行 a、b、$c \in P$ 上的以下函数：$f(a, b, c) = \vee_{i \in [k]} (\delta_{a_i,0} + \delta_{b_i,1} + \delta_{c_i,2} = 2)$。我们可以用语句 $found = found \vee f(r, \pi_1(r)\ \pi_2(r))$ 替换算法 1 第 7 和第 8 行中的最内层循环。注意，f既不依赖于 P、r，也不依赖于置换，而且算法 1 不再直接依赖于 k。为了稍微加快算法 1 的速度，我们可以在算法开始前预先计算并缓存 f，然后在算法运行时查找相应的值。我们预先计算针对谜题 P 中各行特化的 f，并将其称为 f_P。

3.2 强 USP 验证到三维匹配

事实证明，与 P 相比，使用 f_P 更有用。将 f_P 视为函数 f_P：$P \times P \times P \to \{0, 1\}$ 很方便；它是三部超图 $H_P = \langle P \sqcup P \sqcup P, f_P \rangle$ 的关系示性函数的补函数，其中顶点集是 P 的三个副本的不相交并集，而 f_P 表示 H_P 中不存在的超边。

设 $H = \langle P \sqcup P \sqcup P, E \subseteq P^3 \rangle$ 是一个三部三超图。我们称 H具有一个 三维匹配 (3DM)，当且仅当存在一个子集 $M \subseteq E$满足 $|M| = |P|$，且对于任意不同的超边 e_1、$e_2 \in M$、e_1和 e_2均 顶点不相交，即 $e_1 \cap e_2 = \emptyset$。判定一个超图是否具有三维匹配是一个众所周知的 NP 完全问题（例如参见 [18]）。如果一个三维匹配不是集合 非平凡的，则称其为 $\{(r、r、r) \mid r \in P\}$。 图 2 展示了一个具有非平凡三维匹配的三超图。

H_P 中是否存在非平凡三维匹配，直接关系到P 是否为强 USP。

引理 4. 谜题 P 是强 *USP*，当且仅当 H_P 不存在非平凡三维匹配。

证明。 我们先证明反方向。假设 H_p 存在非平凡的三维匹配M。我们利用 M构造 π_1，π_2，$\pi_3 \in$，从而证明 P不是强 USP

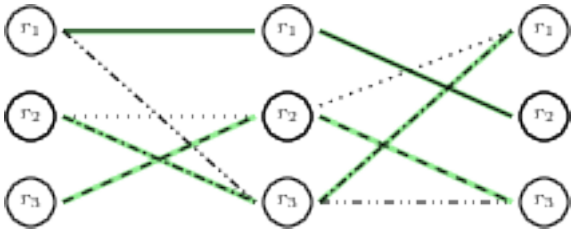

图 2：一个超图 G，其超边为 $E = \{(r_1, r_1, r_2), (r_1, r_3, r_3), (r_2, r_2, r_1), ($ $r_2\ r_3\ r_1\ ,\ r_3\ r_2\ r_3\ \}$ 3 M =,,) (,,)。高亮的超边构成一个 $\{\ r_1\ r_1\ r_2\ ,\ r_2\ r_3\ r_1\ ,\ r_3\ r_2\ r_3\ \}$ G 非平凡的 D 匹配 (,,) (,,) (,,)。

Sym$_P$，它们是该见证。令 π_1 为恒等置换。对于每个 $r \in P$，定义 $\pi_2(r) = q$，其中 $(r, q, *) \in M$。注意，q定义良好且唯一，因为 M 是三维匹配，因此其超边的顶点互不相交。类似地，定义$\pi_3(r) = q$，其中 $(r, *, q) \in M$。注意，根据构造

$$M = \{(\pi_1(r), \pi_2(r), \pi_3(r)) \mid r \in P\}.$$

由于 M是 H_P的一个匹配，$M \subseteq f_P$。因为 M是一个非平凡匹配，所以 $(a$、b、$c) \in M$中的至少一条超边具有 $a \neq b$、$a \neq c$或 $b \neq c$。这分别意味着，按照构造，$\pi_1 \neq \pi_2$、$\pi_1 \neq \pi_3$或 $\pi_2 \neq \pi_3$。在每种情况下，我们都确定了 π_1、π_2和 π_3 并非完全相同。因此，我们确定了置换，使得对于所有 $r \in P$，$f(\pi_1(r)\ \pi_2(r)\ \pi_3(r)) = 0$。这违反了定义 3 的条件 (ii)，因此 P 不是强 USP。

正向的证明是对称的。假设 P 不是强 USP。我们证明 H_P 存在一个三维匹配。若 P 不是强 USP，则必定存在π_1、π_2、$\pi_3 \in$ Sym$_P$，它们并非全部相同，使得定义 3 的条件 (ii) 不成立。定义 $e(r) = (\pi_1(r)\ \pi_2(r)\ \pi_3(r))$，以及 $M = \{e(r) \mid r \in P\}$。由于条件 (ii) 不成立，我们有 $f_P(e(r)) = false$ 对所有 $r \in P$ 都成立。这意味着对所有 $r \in P$，$e(r) \in f_P$，因此 $M \subseteq f_P$。由于 π_1是一个置换，$|M| = |P|$。注意，M 是非平凡的，因为这些置换并非全部相同，因此必定存在某个 $r \in P$，使得 $e(r)$ 的坐标不全相同。因此，M 是一个非平凡的三维匹配。□

根据定义 3，强 USP 验证属于余 NP。请注意，尽管三维匹配是一个NP完全问题，引理 4 并不能立即推出强 USP 的验证是余 NP完全的，因为 H_P并非任意超图。强 USP 验证是否为余 NP完全问题仍未解决。引理 4 意味着，要验证 P是否为强 USP，只需判定 H_P 是否存在非平凡的三维匹配。在接下来的小节中，我们将考察解决后一个问题的算法。回过头来看，我们可以将算法 1 视为一种求解三维匹配的算法。

我们注意到，参数 s 和 k 并非完全独立。首先，$s \leq 3^k$因为宽度为 k 的谜题最多有 $|[3]^k| = 3^k$行。其次，我们通过将一个 (s, k)-谜题

算法 2：双向动态规划验证

输入： 一个 $(s,\ k)$-谜题 P。

输出： 若 P 是强 USP，则输出 YES；否则输出 NO。

函数 VerifyDynamicProgramming (P)
  设 $T = \emptyset$。
  构造三维匹配实例 H_P。
  函数 SearchHalf$(\ell,\ Q,\ \ell_Q,\ R,\ \ell_R,\ \delta,\ t)$
    如果 $\ell = t$ **则**
      如果 $\delta = 1$ **则** ▷ 正向基本情形
        将 $(Q,\ R)$ 插入 T。
        返回 $false$。
      else ▷ 反向基本情形
        若 $(P - Q, P - R) \in T$ **则**
          返回 $true$。
        else
          返回 $false$。
    $res = false$。 ▷ 递归情形
    当 $\ell'_Q = \ell_Q + 1$ 到 s **时**
      当 $\ell'_R = \ell_R + 1$ 到 s **时**
        若 $(p_\ell,\ p_{\ell'_Q}\ p_{\ell'_R}) \in H_P \wedge \neg res$ **则**
          $res = \text{SearchHalf}(\ell + \delta, Q \cup \{p_{\ell'_Q}\}, \ell'_Q, R \cup \{p_{\ell'_R}\}, \ell'_R, \delta, t)$.
    返回 res。
  SearchHalf$(1,\ \emptyset,\ 0,\ \emptyset,\ 0,\ 1,\ \lfloor s/2 \rfloor + 1)$。
  返回 SearchHalf$(s,\ \emptyset,\ 0,\ \emptyset,\ 0,\ -1,\ \lfloor s/2 \rfloor)$。

转换为顶点集为 $[s]^3$上的三维匹配实例，从而完全消除了对 的依赖。不过，这种转换并非没有代价，因为 H_P 的大小是 s 的立方函数，而不是与谜题 $s \cdot k$的大小成线性关系。

3.3 动态规划

认识到强 USP 的验证是三维匹配的一种特化，从而得到一种用于验证的动态规划算法，其运行时间为线性指数时间 $O(2^{2s}\text{poly}(s) + \text{poly}(s, k))$。归约使我们能够用 P 的子集替换来自 Sym P 的置换，并有效降低 算法 1 外层循环的成本，从 $s! = \Theta(2^{s \log s})$ 降至 2^s。

算法 2 描述了一种用于强 USP 验证的递归双向动态规划算法，该算法使用三维匹配实例。该算法包含两个相位。令 $t = \lfloor s/2 \rfloor$。第一阶段确定所有可能的集合 Q, $R \subseteq P$，满足 $|Q| = |R| = t$，使得存在三维匹配 M_1，其属于 H_P，并且限制在顶点 $\{p_1, p_2, \ldots, p_t\} \sqcup Q \sqcup R$上。第一阶段第 7 行会将满足要求的集合 Q, R存储在表 T 中。第二阶段确定所有可能的集合 $Q, R \subseteq P$，满足 $|Q| = |R| = s - t$

使得存在一个三维匹配 M_2，其属于 H_P，并且限制在顶点 $\{p_{t+1}, p_{t+2}, \ldots, p_s\} \sqcup Q \sqcup R$上。对于算法在第二阶段考虑的每一对 (Q, R)，它都会检查 $(P - Q, P - R)$ 是否已在第一阶段插入 T中。如果该对存在，就意味着存在一个三维匹配 H_P，它是 $M = M_1 \cup M_2$。这是因为，根据第 10 行，M_1和 M_2分别是 $\{p_1, \ldots, p_t\} \sqcup (P - R) \sqcup (P - Q)$ 和 $\{p_{t+1}, \ldots p_s\} \sqcup R \sqcup Q$ 上的部分三维匹配，这意味着 M_1 和 M_2 互不相交。第一阶段总是返回 $false$，但该结果会被忽略；第二阶段则返回是否找到了完整匹配，因此，根据引理 4，返回 P 是否为强 USP。

该算法的运行时间主要取决于它检查的集合对 (Q, R) 的数量。注意，在第 15 行和第 16 行中，P 的各行是按顺序考虑的。此外，该算法分别在 ℓ_Q 和 ℓ_R 中跟踪添加到 Q和 R 中的最后一个元素的索引。该算法只会向 Q 或R 添加索引高于先前已添加元素的新元素。综上可知，在一个相位中，每对集合 (Q, R) 至多只会被考虑一次。由于 $Q, R \subseteq P$，集合对 (Q, R) 至多有 $\sum_{i=0}^{t} \binom{s}{i} \cdot \binom{s}{i} \leq (\sum_{i=0}^{t} \binom{s}{i})^2 \leq (2^s)^2 = 4^s$ 个。这意味着每个相位中，SearchHalf 至多会被调用 4^s 次。因此，该算法的运行时间为 $O(4^s \cdot s^2 \cdot \text{poly}(s) + T_{3DM}(s, k))$，其中 s^2 因子来自内层循环，$\text{poly}(s)$ 是操作集合并以哈希表形式跟踪 T内容所需的时间，而 $T_{3DM}(s, k)$ 则表示构造 H_P所需的时间。算法 2 的内存需求同样很高——第一阶段使用 $O(4^s \cdot s)$ 位来存储 T。

请注意，算法 2 不会在遇到 P 为强 USP 时提前终止，因为它必须搜索完所有配对，才能确定不存在符合条件的配对。可以修改该算法，使其在 P 不是强 USP 时提前终止：一旦找到第一个三维匹配见证，就让搜索的第二阶段立即在第 18 行返回。不过，这仍然要求第一阶段运行至完成。一种解决办法是并行运行两个阶段，并让它们相互检查。我们没有采用这种做法，因为它会大幅增加实现的复杂度，而且最终不太可能提升我们组合算法的性能。

作为比较，像 Bj¨orklund 等人提出的更先进技术，可以实现更优的渐近时间复杂度 $O(2^s \text{poly}(s))$ [8]。我们选择不实现他们的算法，因为我们认为这不会显著扩大可进行验证的范围。

3.4 从三维匹配到可满足性

根据引理 4，可以通过构造图 P 并判断其是否存在非平凡三维匹配，来确定谜题 H_P 是否为强 USP。这里，我们将三维匹配问题归约为合取范式公式的可满足性（SAT）问题，然后使用最先进的 SAT 求解器来解决归约后的问题。为进行归约，我们将图 H_P 转换为合取范式公式 Ψ_P，即一个由以下内容取 AND 的深度为 2 的公式：

布尔文字的 OR。我们构造 Ψ_P，使得 Ψ_P 可满足当且仅当 H_P 存在非平凡三维匹配。

令 $H_P = \langle V = P \sqcup P \sqcup P, E \subseteq P^3 \rangle$ 为与谜题 P相关联的三维匹配实例。我们的目标是确定是否存在非平凡的三维匹配 $M \subseteq E$。一种朴素的归约方法是使用变量 $M_{u,v,w}$，分别表示是否将每条超边 $(u, v, w) \in P^3$ 纳入匹配。这会得到一个含有 s^3 个变量、规模为 $\Theta(s^5)$ 的公式Ψ_P，因为纳入一条超边 $e \in P^3$ 会排除 $\Theta(s^2)$ 条与 $e \cap e' \neq \emptyset$相关的超边 e'。为了减小 Ψ_P的规模，我们改用变量集合来表示第二部分和第三部分中与第一部分各顶点匹配的顶点。具体而言，对于所有 $u, v, w \in P$，我们都有布尔变量 $M^1_{u,v}$以及 $M^2_{u,w}$；这些变量按以下方式映射为朴素方案中的赋值：$M^1_{u,v} \wedge M^2_{u,w} \Leftrightarrow M_{u,v,w}$。

现在我们写出三维匹配的合取范式公式。首先，我们加入子句，阻止非边出现在匹配中：

$$\Psi_P^{\text{non-edge}} = \bigwedge_{(u,v,w) \in \overline{E}} (\neg M^1_{u,v} \vee \neg M^2_{u,w}). \tag{1}$$

其次，我们加入子句，要求 H_P 中的每个顶点都与某条边匹配：

$$\begin{aligned} \Psi_P^{\geq 1} = &\left(\bigwedge_{u \in P} (\vee_{v \in P}\ M^1_{u,v}) \wedge (\vee_{w \in P}\ M^2_{u,w}) \right) \\ &\wedge \left(\bigwedge_{v \in P} (\vee_{u \in P}\ M^1_{u,v}) \right) \wedge \left(\bigwedge_{w \in P} (\vee_{u \in P}\ M^2_{u,w}) \right). \end{aligned} \tag{2}$$

第三，我们要求每个顶点至多与一条边匹配，因此加入子句，排除在一个或两个坐标上重叠的匹配边。

$$\Psi_P^{\leq 1} = \bigwedge_{i \in \{1,2\}} \bigwedge_{(u,v),(u',v') \in P^2} (u = u' \vee v = v') \wedge (u, v \neq u', v') \Rightarrow \neg M^i_{u,v} \vee \neg M^i_{u',v'}. \tag{3}$$

第四，我们要求至少有一条对角边不被使用，以排除平凡的三维匹配：$\Psi_{P}^{\text{非平凡}} = \bigvee_{u \in P} \neg M^1_{u,u} \vee \neg M^2_{u,u}$。最后，我们将这些子句合取，得到整体的合取范式公式：$\Psi_P = \Psi_{P}^{\text{非边}} \wedge \Psi_P^{\leq 1} \wedge \Psi_P^{\geq 1} \wedge \Psi_{P}^{\text{非平凡}}$。合取范式公式 Ψ_P 的规模为 $\Theta(s^3)$，含有 $2s^2$ 个变量，比朴素方法小 s^2 倍。因此，我们通过将实例 H_P 转换为合取范式公式 Ψ_P，将三维匹配归约为可满足性问题。

3.5 三维匹配到整数规划

与上一小节类似，我们利用强 USP 验证与三维匹配之间的联系，将前者归约为整数规划这一另一种众所周知的 NP完全问题（例如，参见 [21]），然后应用

最先进的求解器来解决它。同样，令 $H_P = \langle V, E\rangle$ 为与 P相关联的三维匹配实例。我们构造一个关于 $\{0,1\}$的整数规划 Q_P，使其不可行当且仅当 P 是强 USP。这里的归约比前一个更简单，因为线性约束可以自然地表达匹配。

我们用 $M_{u,v,w}$ 表示一个取值属于 $\{0,1\}$ 的变量，用于指示边 $(u,v,w) \in P^3$ 是否存在于匹配中。为确保 M 是 E 的子集，我们向 Q_P添加以下边约束： $\forall u,v,w \in P, \forall (u,v,w) \notin E, M_{u,v,w} = 0$. 我们还要求，图的三个部分中的每个顶点都恰好关联 M 中的一条边。这由 Q_P 中的以下顶点约束表示： $\forall w \in P, \sum_{u,v\in P} M_{u,v,w} = \sum_{u,v\in P} M_{w,u,v} = 1$. 3 $w = \sum_{u,v\in P} M_{u,v,w}$, 最后，由于我们要求 D 匹配是非平凡的，因此添加以下约束：$\sum_{u\in P} M_{u,u,u} < |P|$.

要检查 P 是否为强 USP，我们需要确定 Q_P 是否不可行，即不存在任何对变量 M 的赋值能够满足所有约束。我们注意到，从三维匹配到整数规划的归约可在多项式时间内完成，且 Q_P 中有 s^3 个变量，约束的总规模为 $s^3 \cdot \Theta(1) + 3s \cdot \Theta(s^2) + 1 \cdot \Theta(s^3) = \Theta(s^3)$，与 SAT 归约中 Ψ_P 的规模类似。

3.6 启发式方法

尽管前几节介绍的精确算法较暴力方法有了显著改进，但其性能仍不实用。为解决这一问题，我们还开发了若干快速验证启发式方法，它们可能会给出不确定这一非确定性答案，而不是肯定或否定。接下来，为了验证一个谜题 P，我们运行这组快速启发式方法；如果其中任何一种给出明确的肯定或否定答案，就立即返回。当所有启发式方法都得出不确定时，我们再运行此前讨论过的一种较慢的精确算法。这些启发式方法形式各异，但都依赖于强唯一可解谜题的结构性质。

向下封闭性 我们考虑的最简单启发式方法基于这样一个事实：强 USP 具有向下封闭性。

引理 5。 如果 P 是强 *USP*，那么每个子谜题 $P' \subseteq P$ 也是强 *USP*。

证明。设 P 是强 USP，且 $P' \subseteq P$。根据定义 3，对于每个 $(\pi_1, \pi_2, \pi_3) \in \mathrm{Sym}_P^3$ 不全为恒等置换的情形，都存在 $r \in P$ 和 $i \in [k]$，使得以下三项中恰有两项成立：$(\pi_1(r))_i = 1$、$(\pi_2(r))_i = 2$、$(\pi_3(r))_i = 3$。考虑将这些置换限制为固定 $P \backslash P'$ 中元素的置换。对于这些置换，必然有 $r \in P'$，否则 $r \in P \backslash P'$，并且恰有一个 $j \in [3]$ 使得 $(\pi_j(r))_i = j$ 成立。因此，我们可以删去 $P \backslash P'$ 中的元素，并得出：对于 $\mathrm{Sym}_{P'}$ 中的每个置换元组， P' 满足定义 3 的条件，从而 P' 是强 USP。□

由此得到一种多项式时间启发式方法，可判断某个谜题不是强 USP。非正式地说，该算法接受一个 (s, k)-谜题 P 和 $s' \leq s$,

算法3：向下闭包启发式

输入： 一个 $(s,\ k)$-谜题 P，以及大小 $s' \leq s$。

输出： 如果 P 有一个由 s' 行组成的集合，且这些行不能构成强 USP，则输出“否”，并且否则输出“不确定”。

```
函数 HeuristicDownwardClosed(P, s')
    对 P' ⊆ P, |P'| = s' 执行
        如果 P' 不是强 USP，则返回NO。
    返回 不确定。
```

并验证所有大小为 $P' \subseteq P$的子集 $|P'| = s'$都是强 USP。如果某个子集 P' 不是强 USP，该启发式方法就返回“不确定”，否则返回“不确定”。为完整起见，该算法详述于 算法3。

该算法的运行时间为 $O(\binom{s}{s'} \cdot T(s', k))$，其中 $T(s', k)$ 是验证一个 (s', k)-谜题的运行时间。实际上，对于大于 3 的 s'，我们没有采用这种启发式方法。当 s' 是某个常数 d 时，使用暴力算法（算法 1）验证谜题 P' 的运行时间变为 $O(s^d \cdot T(d, k)) = O(s^d k)$。

唯一子行每个强唯一可解谜题都是唯一可解拼图。谜题要成为 USP 的一个必要条件是，对于 $[3]$，中的每个元素，子行集合中都没有重复项。

引理 6（隐含于 [11]）。如果 P 是 *USP*，那么对于所有 $e \in [3]$，以及不同的行 r_1，$r_2 \in P$，存在一列 $c \in [k]$ 其中一行 r_1 或 r_2 有一个e 而另一行没有。

证明。反设结论不成立，则不同的行 $r_1, r_2 \in P$ 在某个 $e \in [3]$ 的恰好相同的列中都有 e。我们证明 P 不是 USP。选取 $\pi_e = (r_1 r_2)$，即对行 r_1 和 r_2 中的 e 的子行进行转置的置换。将 $[3] \backslash \{e\}$ 中元素对应的另外两个置换取为恒等置换。由于这些置换并非全为恒等置换，定义 2 的后半部分适用。然而，由这些置换得到的谜题与 P 相同；对于所有 $c \in [k]$，以及每一行 $r \in P$，恰好存在一个 $i \in [3]$ 使得 $(\pi_i(r))_c = i$。因此不满足唯一可解的定义，矛盾。□

注意，引理 6 的逆命题并不成立。 图 1中的谜题就是一个例子：它并非唯一可解，但每个元素对应的子行各不相同。

我们可以通过一种线性时间启发式方法，使引理 6 更加有效，该方法能够排除并非（强）USP 的谜题。虽然方法很直接，为求完整起见，我们在算法 4 中将其形式化。当集合以哈希表的形式实现时，该算法的期望运行时间为 $O(s \cdot k$ 时间，与谜题 P的规模呈线性关系。另一种最坏情况下运行时间为$O(s \cdot k$的实现方式是使用基数排序，将子行的特征序列作为二进制数进行排序

算法 4：唯一子行启发式

输入： 一个 (s,k)-谜题 P。

输出： 如果找到见证证明 P 不是（强）USP，则输出 NO；否则输出不确定

函数 HeuristicUniquePieces(P)
  初始化空集合 S_1, S_2, S_3。
  对 $r \in P$ **执行**
    对 $e \in [3]$ **执行**
      令 $h = \{c \in [k] \mid r_c = e\}$。
      如果 $h \in S_e$，**则返回** 否。
      $S_e = S_e \cup \{h\}$。
  return 不确定。

然后扫描相邻的行，以检测重复项。

对于大小为二的子谜题，唯一块启发式方法等价于向下闭包启发式。

引理 7. 设 P 是一个 (s,k)-谜题，则 HeuristicUniquePieces(P) = HeuristicDownwardClosed(P, 2)。

证明。我们证明两个方向。

假设 P 未通过唯一块启发式方法；不失一般性，设为 $e = 1$，那么存在不同的行 $r_1, r_2 \in P$，其中包含 1 的单元格都位于相同的列中。这意味着我们可以交换这些 1 所在的子行，而不会造成重叠或改变谜题。这说明 $P' = \{r_1, r_2\}$ 不是一个（强）USP。由于 $|P'| = 2$ 和 $P' \subseteq P$，针对 $s' = 2$ 的向下闭包启发式也会得出结论：P 不是一个（强）USP。

假设 P 未能通过针对 $s' = 2$的向下闭包启发式方法。那么存在一对不同的行 r_1、$r_2 \in P$，使得 $P' = \{r_1, r_2\}$ 不是强 USP。假设不存在任何 r_1 与 r_2 不同的列，那么对于所有元素，r_1、r_2 的子行都相同，因此 P 未能通过唯一块启发式。对于另一种情况，假设至少存在一列 $c \in [k]$，使得 r_1 与 r_2不同。不失一般性，令该列为 $((r_1)_c, (r_2)_c) = (1, 2)$。由于 P' 不是 SUSP，且该列为 (1, 2)，因此不能有其他列属于集合 {(1, 3)、(2, 3)、(3, 2)、(3, 1)}，否则它们会与列 (1, 2) 一起构成一个 SUSP。这意味着 P' 中的列只能属于集合{(1, 2)、(2, 1)、(1, 1)、(2, 2)、(3, 3)}。因此，包含 2 的列必须相同，并且 r_1 和 r_2 中 2 对应的子行完全一致。因此，P'，所以 P未能通过唯一块启发式。□

这个证明的一个推论是，对于大小为二的拼图，每个 USP 也是强 USP。

推论 1. 设 P 是一个$(2,k)$-谜题，若 P 是唯一可解拼图，则P 是强唯一可解谜题。

由于对于 $s' = 2$，唯一块启发式方法与向下闭包启发式方法等价，且唯一棋子的运行时间与谜题大小呈线性关系，$O(s \cdot k)$，而向下闭包的运行时间为 $O(s^2 \cdot k)$，因此我们用唯一块启发式方法代替 $s' = 2$中的向下闭包。

贪心 此启发式方法尝试利用引理 4，并为实例 H_P 贪心地搜索三维匹配。该启发式方法以迭代方式进行：找出三维匹配实例第一部分中超边数最少的顶点，然后随机选择一条与该顶点关联的超边，将其加入三维匹配。如果该启发式方法成功构造出三维匹配，就返回 NO，表示输入谜题 P 不是强 USP。如果先前的选择导致匹配不可行，该启发式方法便从头开始。此过程重复若干次，之后若仍未成功，则返回不确定。在我们的实现中，我们使用 s^2 次尝试，因为这与归约的运行时间相近；在包含 $k = 6$、7、8 的谜题领域中，实测表明这减少了需要完整验证的实例数量，同时不会使运行时间增加太多。贪心启发式算法形式化地描述于算法 5。

数组 cts 用于存储仍与第一坐标上顶点 $cts[u]$ 相关联的超边数量 u。算法的大部分工作都致力于维护这一不变量。集合 U、V、W 分别存储三个坐标上已纳入部分三维匹配的顶点。与算法 2 一样，我们不存储匹配本身，只存储其中涉及的顶点。第 10 行的中断条件会在部分三维匹配陷入死局、无法扩展为完整三维匹配时触发。第 23 行的条件在构造出完整三维匹配时成立，并使算法返回 P 不是强 USP。

该算法的运行时间为 $O(s^3t + T_{3DM}(s,\ k))$，其中 $T_{3DM}(s,\ k)$ 是从 $(s,\ k)$-谜题构造三维匹配实例所需的时间。该算法可能比向下闭包启发式方法慢得多，因此在实践中我们设置 $t = s^2$。不过，如果无法扩展三维匹配，主循环可以在第 10 行提前终止，从而使预期运行时间远低于最坏情况下的运行时间。对于一个是强 USP 的谜题 P，该启发式方法需要完整的 $\Omega(s^3t + T_{3DM}(s,\ k))$ 时间。

与向下闭包和唯一拼块启发式方法相比，这种启发式方法的效率要低得多。因此，只有在其他启发式方法都失败时，我们才会运行它。有关实验中这些启发式方法效果的比较，请参见第 7.2 小节。

3.7 混合型算法

我们的最终验证算法（算法 6）结合了多种精确算法和启发式方法。我们通过实验确定了规模阈值，以决定对较小的 k 采用哪种算法和启发式方法，并重点关注我们的强 USP 搜索算法能够处理的数值范围 $k \leq 6$ （或接近可处理的范围

算法 5： 贪心启发式算法

输入： 一个 $(s,\ k)$-谜题 P，以及迭代上界 t。
输出： 如果找到 P 不是强 USP 的见证，则输出 NO，否则输出不确定

函数 HeuristicGreedy(P)
  构造三维匹配实例 H_P。
  对于 $i = 1$ 到 t，**执行**
    对于 $u \in P$，**执行**
      $cts[u] = \sum_{v,w \in P} H_P(u, v, w)$. ▷ 与顶点 u关联的超边数量。
    令 $U,\ V,\ W = \emptyset$.
    令 $m = 0$. ▷ 匹配中的超边数量。
    当 $m < s$ **时执行**
      选择 $u \in \{w \in \bar{U} \mid cts[w] =$ 均匀随机地取最大值$_{v \in \bar{U}}\ cts[v]\}$。
      如果 $cts[u] = 0$，**则** 跳出。
      令 $D = \{(v, w) \in \bar{V} \times \bar{W} \mid H_P(u,\ v,\ w) = 1\}$。
      均匀随机地选择 $(v,\ w) \in D$。
      对于 $v' \in P$，**执行** ▷ 更新边的数量。
        对于 $w' \in P$，**执行**
          如果 $(v', w') \in \bar{V} \times \bar{W}$和 $H_P(u, v', w') = 1$ **则**
            $cts[u]$--。
          如果 $(v', w') \in \bar{U} \times \bar{W}$和 $H_P(v',\ v,\ w') = 1$和 $v' \neq u$ **那么**
            $cts[v']$——。
          如果 $(v', w') \in \bar{U} \times \bar{V}$和 $H_P(v',\ w',\ w) = 1$和 $v' \notin \{u, v\}$ **那么**
            $cts[v']$——。
      $U, V, W = U \cup \{u\}, V \cup \{v\}, W \cup \{w\}$。 ▷ 将边加入匹配。
      $m = m + 1$.
    如果 $m \geq s$，**则返回** 否。 ▷ 3找到 D 匹配，因此不是 SUSP，停止。
  返回 不确定。

可处理的范围 $k \leq 8$）。我们决定并行执行归约到可满足性和整数规划的两种归约，因为目前尚不清楚哪种算法总体表现更好。由于任一算法完成后验证便会停止，因此所浪费的计算量至多是单独运行表现较好的算法时的两倍。我们还选择这样做，是因为实验中观察到许多实例会令其中一种算法难以应对，而另一种算法却能轻松处理——因此，混合算法的平均表现优于单独使用各个精确算法。我们将在 第 7.2 小节 中展示，我们的混合算法和启发式方法在实践中能够快速验证宽度较小的强 USP k。此外，第 7.3 小节讨论了在基准实验中，针对不同类型实例，SAT 方法与 IP 方法的相对表现。

算法 6： 混合验证

输入： 一个 (s, k)-谜题 P。
输出： 如果 P 是强 USP，则返回 YES，否则返回 NO。

函数 VERIFY(P)
  如果 $s \leq 2$ **，则返回** VERIFYBRUTEFORCE(P)。
  如果 HEURISTICUNIQUEPIECES(P) 不是不确定，则返回结果。
  如果 $s \leq 7$ **则返回** VERIFYDYNAMICPROGRAMMING(P).
  如果 HEURISTICDOWNWARDCLOSED(P,3) 不是不确定，则返回结果。
  如果 HEURISTICGREEDY(P) 不是不确定，则返回结果。
  并行运行 VERIFYSAT(P) 和 VERIFYIP(P)，并返回第一个结果。

4 搜索强 USP

有了实用的验证算法后，我们来考虑搜索大型强 USP 的问题。由于强 USP 的集合是向下封闭的，一种自然的搜索策略是：从空集开始，反复尝试添加行，同时保持强 USP 的性质。不过，尽管这种策略会找到一个规模最大的强 USP，但不能保证找到规模最大的强 USP。这是因为强 USP 的集合并不构成拟阵，而只是一个独立系统（参见例如[25]）。

具体而言，(i) 空谜题是强 USP；(ii) 根据引理 5，强 USP 的集合是向下封闭的。成为拟阵所需的最后一项性质是增广性质，它要求对于任意一对强 USPP_1， P_2，若 $|P_1| \leq |P_2|$，则存在 $r \in P_2 \backslash P_1$ 中的一行，使得 $P_1 \cup \{r\}$ 也是强 USP。一个简单的反例是考虑强 USP $P_1 = \{32\}$和 $P_2 = \{12, 23\}$。利用引理 6，我们可以看到 $P_1 \cup \{12\} = \{12, 32\}$和 $P_1 \cup \{23\} = \{23, 32\}$都不是强 USP，因此增广性质不成立。其结果之一是，朴素的贪心算法可能无法有效地找到规模最大的强 USP。此外，目前我们还不知道有何高效算法能够给定一个强 USP P，并确定一行 r，使得 $P \cup \{r\}$ 是强 USP。

尽管如此，我们仍成功地将通用树搜索技术与基于强 USP 对称性的剪枝结合起来，并配合实用的验证算法，为较小的 k 构造出规模最大的强 USP。

4.1 谜题对称性

由于谜题被定义为行的集合，因此谜题中各行的顺序P 不会影响 SUSP 性质。类似地，虽然不那么显而易见，重新排列谜题的列也不会改变 SUSP 性质，因为所要求的存在性条件 $\exists c \in [k]$st. (...) 在定义 3 中与顺序无关

列的顺序。最后，通常用于表示谜题元素的字母表 [3] 完全是任意的，任何由三个不同值组成的集合都足以胜任。这些值除了便于简洁地表述 SUSP 定义之外，并无数学意义。这一逻辑可以形式化为以下引理。

引理 8。设 $\rho \in \mathrm{Sym}^{[k]}, \delta \in \mathrm{Sym}^{[3]}$。一个 (s,k)谜题 P是强 *USP*，当且仅当 $\{(\delta(r_{\rho(c)}))_{c\in[k]} \mid r \in P\}$ 是强 *USP*。

证明。由 定义 1 和 定义 3立即可得。 □

该引理意味着 SUSP 性质在这类谜题变换下保持不变。我们称以这种方式相关的两个谜题 P， P' 为同构，并用符号 $P \cong P'$ 表示。关系 $\cong$ 是一种等价关系，因为置换可逆，因此它将谜题集合划分为若干等价类。

这种同构概念与图中的同一概念自然相关。对于每个 (s,k)-谜题 P，我们都可以定义一个彩色无向图 G_P。该图由顶点组成，这些顶点被划分为四个颜色各异的集合：$V = \{row_r\}_{r\in[s]} \sqcup \{col_c\}_{c\in[k]} \sqcup \{e_i\}_{i\in[3]} \sqcup \{v_{r,c}\}_{(r,c)\in[s]\times[k]}$。$G_P$ 中有 $s+k+3+s\cdot k$ 个顶点。前三部分分别是表示 P 的行和列，以及 $[3]$，的元素的顶点；第四部分则是 P 中每个 $s\cdot k$ 单元格所对应的顶点。G_P 的边关系很简单：每个顶点 $v_{r,c}$ 都与三个顶点相连，这三个顶点分别对应于索引为 (r,c) 的单元格在 P 中包含的行、列和元素。特别地，附着于 $v_{r,c}$ 的三条边为 $(v_{r,c}, row_r)$、$(v_{r,c}, col_c)$、$(v_{r,c}, elt_{P(r,c)})$。总计，G_P 有 $3\cdot s\cdot k$ 条边。由于行、列和元素对应的顶点集各自具有唯一的颜色，且 P 的每个单元格都与表示其行、列和元素的顶点相连，因此 G_P 的自同构与 P 在行、列和元素置换下的自同构一一对应。这意味着，对于两个 (s,k)-谜题 P, P'，如果 $G_P \cong G_{P'}$，那么就存在对 P 的行、列和元素的置换，使其得到 P'。此外，根据引理 8，如果 $G_P \cong G_{P'}$，那么 $P \cong P'$，且 P 是 SUSP，当且仅当 P' 是 SUSP。

4.2 对称性剪枝的树搜索 h

搜索强 USP 的一种自然方法是采用广度优先搜索，并利用强 USP 向下封闭这一事实（引理 5）：为了找到可能的最大宽度k 强 USP，（i）从所有可能的第一行开始——即 $3^k(1,k)$-谜题；（ii）尝试用所有可能的行扩展所得谜题，只保留强 USP 且不同构于此前见过的强 USP 的谜题，以此形成新的搜索前沿；（iii）重复步骤（ii），直到搜索前沿为空。

为确保算法不会重复访问同构谜题，我们使用谜题图的规范图表示 $[G_p]$。G_P 规范图表示是图的一种二进制编码，具有这样的性质：对于任意两个图G_1， G_2，$[G_1] = [G_2]$ 当且仅当$G_1 \cong G_2$（例如，参见 [24]）。随着搜索算法运行，我们

算法 7：对称性剪枝广度优先搜索

输入： 一个整数 $k \geq 0$。
输出： 数字 b，即最大宽度为k 的强 USP 的大小。

函数 SP-BFS(k)
  令 Q 为空队列。
  令 I 为空集合。
  令 $b = 0$。
  ENQUEUE(Q, $\emptyset$)。
  当 Q 不为空时 **执行**
    P = DEQUEUE(Q)。
    对 $r \in [3]^k \backslash P$ **执行**
      令 $P' = P \cup \{r\}$。
      如果 VERIFY(P') 且 $[G'_P] \notin I$ **则**
        ENQUEUE(Q, P')。
        $I = I \cup \{[G'_P]\}$。
        $b = |P'|$。
  返回 b。

记录 I 每个不同谜题的规范图表示 $[G_P]$ 集合P，这些谜题已被加入搜索前沿。每当考虑将谜题 P' 加入搜索前沿时，我们首先检查其规范图表示 $[G_{P'}] \in I$；如果已存在，就不将其加入 P' 搜索前沿。使用谜题的规范表示，可以显著缩小搜索空间：只需从 $[P]$ 开始搜索，而不是遍历每个 $P' \cong P$，并且不会将$[P]$ 的重复项加入队列。该算法 SP-BFS 在算法 7中进行了形式化描述。

我们论证该算法的正确性。

引理9. 对于 $k \in \mathbb{N}$, SP-BFS*(k)*，返回满足存在一个 (s, k)-*SUSP* 条件的最大整数 s 。

证明。 暂且忽略 I 所执行的剪枝。显然可以论证，SP-BFS 的行为如同在所有强 USP 构成的树上执行通用广度优先搜索算法。这是因为强 USP 的向下闭包性质（引理 5）使得任何强 USP P 都可以从平凡强 USP $\emptyset$ 通过一系列行包含到达。 SP-BFS(k) 会对所有宽度为 k 的强 USP 进行穷举搜索，并返回此类强 USP 的最大规模 b 。

我们论证，在考虑 I 所参与的剪枝时， SP-BFS(k) 会为属于强唯一可解谜题的每个等价类恰好将一个元素加入队列。因此，根据 引理 8，该算法必须探索宽度为 k 的强唯一可解谜题的每个等价类。因此，它会探索一个包含规模最大的强唯一可解谜题的等价类，随后返回该规模，这正是预期的输出。

为完成论证并证明经过对称性剪枝的搜索覆盖了整个等价类搜索空间，反设存在某个最小的 {v1}，使得有一个 ({v2}<style id='3'>,</style>{v5})-谜题 {v6}，其等价类 {v7}未被搜索。

我们知道 $s > 1$，因为算法一开始会考虑所有可能的 $(1, k)$-谜题。令 P' 为从 P 中移除一行 r后得到的 $(s-1, k)$-谜题； P' 至少有一行，因为 $s > 1$。根据假设，SP-BFS 已访问 $[P']$ 的等价类，因为 P'的大小为 $s - 1 < s$。考虑 $[P]$，并移除与 r 对应的行，得到 $[P]'$。必有 $[P'] \cong [P]'$。这个同构可扩展到 $[P]$，因为必然存在一行 r'，使得 $([P'] \cup \{r'\}) \cong [P]$，其中 r' 替换了从 $[P]$中移除的行。因此，既然 $[P']$ 已被搜索，算法就必须考虑所有可能用于扩展的行，包括 r'。这意味着 $[P]$ 的等价类已被搜索，这与我们的假设矛盾。因此，SP-BFS 会搜索所有强唯一可解谜题的等价类。□

这种方法缩小了搜索空间，既缩短了搜索的运行时间，也减少了追踪边界谜题所需的空间。SP-BFS 的最坏情况运行时间为 $O(3^k \cdot \#EQUIV(k) \cdot (T_{\text{VERIFY}}(s_k+1, k) + T_{\text{CANONIZE}}(s_k, k))$，其中，$\#EQUIV(k)$ 是宽度为 k 的强 USP 的等价类数量， $T_{\text{VERIFY}}(s_k + 1, k)$ 是验证算法检查的最大尺寸 $(s_k + 1, k)$ 谜题所需的时间，而 $T_{\text{CANONIZE}}(s_k, k)$ 是计算算法考虑的每个谜题 P 的规范图表示所需的时间（假设 T_{VERIFY} 和 T_{CANONIZE} 关于其参数单调）。

参见 第 7.1 小节 ，了解运行 SP-BFS 的实验结果以及关于实现问题的讨论。

5 上界

尽管这条研究路线的重点是构造规模足够大的强 USP，从而实现更快的矩阵乘法算法，但我们的技术和方法也可用于寻找更紧的强 USP 大小上界。本节将介绍若干 SUSP 大小上界。

ω 界。 以往研究明确讨论了 USP 无限族的容量上界（参见 [11, 引理 3.2、定理 3。3]）。由于每个 SUSP 都是 USP，这些上界也适用于 SUSP，并可改写为适用于单个谜题的上界。第一个上界，我们称之为“ω 界”，由以下两点得出：(i)引理 1对于固定的 k，是单调不增的；(ii) $\omega \geq 2$。要计算这个界，我们在越来越大的 值上检验 s 引理 1 中的不等式，直到其推论即将推出 $\omega < 2$，这与 $\omega \geq 2$矛盾。

唯一拼块界。第二个界，我们称之为“唯一拼块界”，直接由引理 6得出。由于该引理要求（强）USP 的每一行都有一个唯一的 1 号、2 号和 3 号棋子，因此强 USP 的总行数不能超过 2^k。

USP 界。第三个界，我们称之为“USP 界”，由 [11, 引理 32]的证明得出。尽管该文章中没有明确说明，证明依赖于以下子论断，它直接将 s 界定为 k的函数。

命题2. 设 P 是一个 (s,k)-*USP*，则

$$s \leq \sum_{c_1=0}^{k} \sum_{c_2=0}^{k-c_1} \min\left(\binom{k}{c_1}, \binom{k}{c_2}, \binom{k}{k-(c_1+c_2)}\right) = O\left(k^2 \cdot \left(\frac{3}{2^{2/3}}\right)^k\right).$$

注意，当 $\frac{3}{2^{2/3}} \approx 1.8899 < 2$ 时，USP 界在渐近意义上比唯一拼块界更紧。

团界。第四个界称为“团界”，它源于强唯一可解谜题具有向下封闭性（引理 5）。具体而言，如果 P 是一个强唯一可解谜题，那么对于每个具有 2 行的 $P' \subseteq P$，它也必定是一个强唯一可解谜题。固定 $k \in \mathbb{N}$，并考虑图 G_k：其顶点对应于宽度为 k 的谜题可能具有的行，也就是 $[3]^k$ 中的字符串；若 $\{r_1, r_2\}$ 是一个强唯一可解谜题，则 r_1 与 $r_2 \in [3]^k$ 之间存在一条边。注意，根据向下封闭性，每个 (s,k)-强唯一可解谜题都对应于 G_k 中一个大小为 s 的团。这种方法可自然地从团问题推广到 h-超团问题，其中图 G_k^h 与 $G_k = G_k^2$ 具有相同的 3^k 个顶点，但其边为元数为 h 的超边 $\{r_1, r_2, \ldots, r_h\}$，这些超边是 (h,k)-强唯一可解谜题。

命题3。 设 P 是一个(s,k)-*SUSP*，且 $2 \leq h \leq s$。那么对于

$$G_k^h = \langle V = [3]^k, E = \{P' \subseteq V \mid P' \text{ is a strong USP and } |P'| = h\}\rangle,$$

$(G_k^h, s) \in h$-超图团。

因此， G_k^h 中最大超图团的大小是宽度为 k 的 SUSP 大小的一个上界。我们用“团界”表示该界针对 $h = 2$ 的具体实例。

穷举界。 对于第五个界（我们称之为“穷举界”），我们考虑 算法 7 在 k 的范围内运行时的结果，此时可以切实地穷举整个搜索空间。由于这些界基于穷举搜索，因此本质上是紧的。

向下闭合界。 我们考虑的最后一个界来自强唯一可解谜题的向下闭合性质。

命题3。 设 P 是一个(s,k)-*SUSP*，且满足 $k > 1$，那么存在一个$(\lceil \frac{s}{3} \rceil, k-1)$-*SUSP*。

证明。任取 $c \in [k]$，并考虑 P 的 c^{th} 列。根据平均原理，该列中必定有 $e \in [3]$ 的某个元素至少出现 $\lceil \frac{s}{3} \rceil$ 次。令 $P' \subset P$ 为 P 的子谜题，其各行在 c^{th} 列上的值均为 e。P' 是强 USP，因为 P 是强 USP，而强 USP 对向下取子谜题封闭

封闭（引理 5）。通过移除 c^{th} 列，构造 P'。 P'' 是强 USP，因为 P' 是强 USP，而且强 USP 性质在添加或移除常值列时保持不变。根据构造， P'' 是一个（$\lceil \frac{s}{3} \rceil$， $k-1$）- SUSP。 □

此界不像其他界那样能独立应用，但它可以将 $s \leq u$ 处的上界提升至 k 处，并将 $s \leq 3u$ 处的上界提升至 $k+1$处。

参见第 7.1 小节，了解对小宽度情况下上述界进行评估的结果，以及具体计算这些界所涉及问题的讨论。

6 实现

我们实现了验证算法、启发式方法和搜索算法，以及各种实用工具和适当的数据结构，用于在 C 语言中表示谜题等底层信息++。我们的实现源代码基于 MIT 许可证发布，地址为https://bitbucket.org/paraphase/matmult。

我们使用了若干外部库，其中的子程序对算法的运行至关重要。基于 IP 的验证器和团界计算器都使用商业闭源混合整数规划求解器 Gurobi，来求解由归约生成的整数规划 [19]。我们的基于 SAT 的验证器默认使用 `kissat-sc2021-sat` 求解器，该求解器由 A. Biere、M. Fleury 和 M. Heisinger 开发，参加了 2021 年 SAT 竞赛 [6, 页面 10]。请注意，本文的会议版本使用的是 MapleCOMSPS 求解器——参见第 7.3 小节，了解求解器基准测试、比较和选择的讨论。我们使用混合验证器实现了算法 7，并使用图自同构库 Nauty [24]作为子程序，对 G_P执行所需的图规范化。SP-BFS 实现的早期版本面向高性能计算集群环境，因为我们的暴力搜索和动态规划实现效率不足。后续对验证算法的改进使这一需求不复存在。尽管如此，我们的 SP-BFS 实现仍使用 MPI，并采用 MapReduce 框架[27] 维护分布式搜索前沿。

我们的代码库还包含多种受深度优先搜索启发的算法实现，用于寻找强 USP。这些算法采用了我们的混合验证实现和谜题对称性剪枝技术，相关内容见 第 4 节。为简洁起见，也为了让本文聚焦于强 USP 验证，我们决定不讨论这些算法，而将其留待后续文章介绍。尽管如此，我们在下一节中找到并报告的一些具体谜题，确实是由这类算法生成的。这些谜题找到后，我们使用第 3 节中详细讨论的技术，对其进行了实验验证，确认它们是强 USP。

		k											
		1	2	3	4	5	6	7	8	9	10	11	12
[11]	$s \geq$	**1**	**2**	**3**	4	4	10	10	16	36	36	36	136
	$\omega \leq$	3.00	2.88	2.85	2.85		2.80			2.74			2.70
这项工作	$s \geq$	**1**	**2**	**3**	**5**	**8**	14	21	30	42	64	112	196
	$\omega \leq$	3.00	2.88	2.85	2.81	2.78	2.74	2.73	2.72	2.72	2.71	2.68	2.66

表1：与 [11] 相比，宽度为 k 的强 USP 的最大规模下界及其所蕴含的上界。粗体表示该 k的紧确结果。

7 实验结果

对于小常数宽度的 k，我们的实验结果包括以下几类：（i）以找到的谜题为见证，给出宽度为 k 的强 USP 最大规模的构造性下界；（ii）宽度为 k 的强 USP 最大规模的上界；（iii）宽度为 k 时强唯一可解谜题的数量及其等价类数量；（iv）用于比较验证算法运行时间并区分启发式方法有效可能性的实验数据；（v）一个可满足／不可满足实例基准数据集，我们用它来比较不同可满足求解器作为验证器中基于可满足部分的子程序时的效果。

本节的所有结果均通过在同一台搭载 Ubuntu 20.04、3.00 GHz Intel Core i9-10980XE 处理器和 128 GB 内存的 PC 上运行我们的算法实现得出。

7.1 强 USP 规模的新上界和下界

新的下界。表1汇总了强 USP 最大规模的新下界，并与 [11]进行了比较。 [11] 的下界来自命题 3.1 和 3.8 中的构造，这些构造给出了k 为偶数或 k 能被三整除时的强 USP 族。对于 k不能被二或三整除的情况，我们通过添加新列来外推其构造，这样做可以保留 SUSP 性质。表中 ω 的上界是将 s 和k 代入引理 1并对 m进行优化后计算得出的。为清楚起见，我们省略了会大于先前各列的 ω。表中的结果是通过运行 SP-BFS 和其他搜索算法得出的；这些算法验证了最终结果是强 USP。由于 SP-BFS 具有穷举性质，对于所有 $k \leq 5$，我们的界都是紧确的；此外，我们还以构造性方法改进了 $4 \leq k \leq 12$的已知下界。

图3展示了我们找到的具有最大规模的强 USP 的代表性示例，适用于 $k \leq 6$。我们找到的强可唯一求解的 (14，6) 谜题，在小 ω 的情况下，相较于 [11] 的构造，在 k方面取得了最大的改进。此外，我们针对 $k = 12$ 构造的谜题，是通过取两个强可唯一求解的 (14，6) 谜题的笛卡尔积得到的。注意， [11]

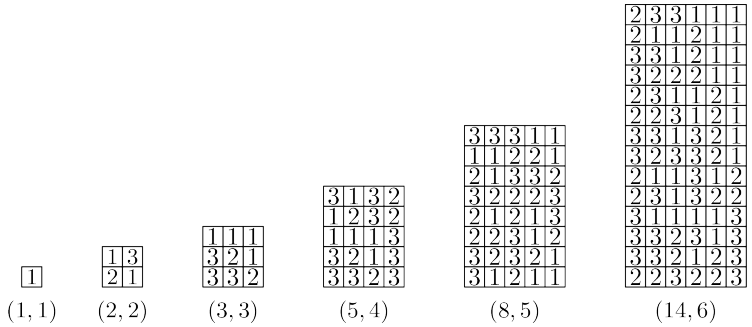

图 3：宽度为 $k = 1$、2、.....、6 的代表性最大规模强 USP。

给出了一个强 USP 的无限族，当 k 趋于无穷时，其达到 $\omega < 2.48$ ，这比我们的结果能够直接达到的更强。

新的上界。表 2 总结了对宽度为 $k \leq 12$ 的谜题计算第 5 节中的界的结果。除团界外，计算过程都很常规；团界需要构造 G_k，将其转换为混合整数规划，并使用 Gurobi [19]求解该规划。在我们的测试系统上，计算可行的上限为 $k = 11$。我们还尝试计算 3-超图团界的上界，但发现对于 $k \geq 5$ 而言无法计算，因此省略了相关结果。表格的最后一行列出了我们得到的最佳上界，其中包括应用向下闭合界，将相邻的界提升到 $k = 6$ 和 $k = 12$。这些上界比 [11] 直接推出的上界更强。

请注意，穷举搜索得出了最佳且最紧的界，而团界明显强于唯一拼块界、USP 界和 ω界。唯一拼块界似乎强于 USP 界，但我们知道，这是 k取值较小所造成的假象。随着 k 增大，USP 界将比唯一拼块界更紧。根据我们在 $k = 6$ 上花费的处理时间，我们推测 $s = 14$ 对于 $k = 6$ 是紧的，而我们对 $k > 6$ 的下界并不紧。我们的结果表明，强 USP 的构造仍有很大的改进空间，而且可能存在适用于 $k = 7$，8，9的大型谜题，它们能胜过 [11]的构造，甚至可能接近 Coppersmith-Winograd 改进。尽管如此，要继续推进 $k > 6$，似乎还需要对 SUSP 搜索问题取得新的见解。

强 *USP* 计数。 表 3 展示了使用 SP-BFS 穷举计算的强 USP 数量以及 SUSP 等价类数量，计算时采用了对称剪枝和未采用对称剪枝两种方式。请注意，即使对于 (3，3)-SUSP，强 USP 的数量也比强 USP 等价类的数量高出许多个数量级。即使采用谜题对称性剪枝，穷举搜索也变得不可行

	k											
界	1	2	3	4	5	6	7	8	9	10	11	12
ω	3	7	15	31	62	120	230	438	831	1,575	2,890	5,637
唯一	2	4	8	16	32	64	128	256	512	1,024	2,048	4,096
USP	3	6	12	24	45	87	168	312	597	1,140	2,112	4,023
团	**1**	3	5	9	17	30	55	105	186	348	654	
穷举	**1**	**2**	**3**	**5**	**8**							
Best	**1**	**2**	**3**	**5**	**8**	24	55	105	186	348	654	1,962

表 2：宽度为 $k \leq 12$ 的 SUSP 大小上界。粗体表示该界是紧的，空白表示无法计算该谜题宽度对应的结果。

对于 $k \geq 6$ 而言，算法 7 用于存储搜索前沿的内存用量超过了测试系统可用的 128GB。

7.2 算法性能

为了衡量验证算法和启发式方法的性能，我们在参数空间中针对宽度 $k = 5 \ldots 12$ 和大小 $s = 1 \ldots 60$的每个点进行遍历，并在每个点上对 10,000 个随机谜题运行这些方法。我们选择通过随机抽样来测试性能，因为我们无法获得大量已求解实例。该领域与搜索空间的前沿相吻合，我们调整了混合算法中启发式方法和算法的参数，使其在该领域表现良好。我们没有深入研究该领域之外的性能特征。在图 4、5 和 6 中，为简洁起见，我们仅绘制了参数空间中具有代表性的结果，范围为 $k \in \{6, 9\}$。

运行时间。 图 4展示了我们的验证算法的平均运行时间（以秒为单位）。除规模非常小的情况外，暴力搜索和动态规划算法表现不佳， $s \leq 8$，其曲线大致符合我们预期的指数时间上界。两种基于归约的算法（SAT 和整数规划）的曲线表现相似。当 s较小时，它们比暴力搜索和动态规划算法慢；而当 s 较大时，速度则快得多。我们推测，前者是由于构造归约后的实例所需的成本以及第三方工具带来的开销。还可以看到，对于较大的规模，SAT 归约在$k = 6$时明显优于整数规划归约；但随着 k 增大，这一差距会缩小。我们还注意到，在 k 的不同设置下，整数规划归约的运行时间基本相同，且不受 k影响。这很可能是因为整数规划实例的规模只取决于 s。当 s较小时，混合算法通常表现最好或接近最好；而当 s较大时，它显然更快。请注意，当 s 较小时，它的表现与动态规划算法十分接近，而当

	k					
s	1	2	3	4	5	6
1	**13**	**2**	9	**3**	27	**4**	81	**5**	243	**7**	729
2		**224**	**9**	408	**33**	4,848	**91**	50,160	**229**	486,024
3			**9**	1,800	**240**	182,304	**2,429**	8,361,000	**16,971**	291,347,280
4					**728**	2,445,120	**59,149**	992,377,400	**1,611,648**	?
5					**190**	3,248,640	**707,029**	?	?	?
6							**2,337,715**	?	?	?
7							**1,359,649**	?	?	?
8							**89,196**	?	?	?
9									?	?

表 3：按（s，k）维度的谜题所对应的等价类数量（左侧粗体）与编码的强唯一拼图总数（右侧常规字体）。使用算法 7计算。空单元格表示 SUSP 和等价类的数量为零。? 表示因计算不可行而未知的数值。

基于归约的算法和启发式方法会在 s较大时启动。请注意，对于较大的 s，混合算法的运行时间基本为常数，不过出现这种情况的规模会随着 k而增大。我们认为，这是因为强唯一拼图的密度会随着 s增大而迅速降低，而且随机选取的拼图很可能不满足定义 3，因此会被唯一拼块启发式方法迅速拒绝。进一步的证据是，对于较大的 k，混合算法的运行时间会趋近于唯一块启发式的运行时间。

启发式方法的有效性。图 5展示了各个单独的启发式方法区分基准测试集中随机谜题的概率。可以看到，对于 $s' = 2$，向下闭包启发式与唯一拼块启发式的区分能力一致，这表明实验结果与引理 7相符。此外，出于同样的原因，对于 $s' = 3$，向下闭包启发式的区分概率至少与唯一拼块启发式一样高。在图中，这三种启发式方法在谜题规模达到 $s = 30$时，区分随机谜题的概率几乎达到 100%。贪心启发式的表现不如其他方法，且随着 k 增加，表现显著变差。此处未绘出这些启发式方法的运行时间，但其表现符合前文分析的预期。正如前文所述，唯一拼块启发式的运行时间与谜题规模呈线性关系，是这些启发式方法中最快的。图 4 展示了混合算法与唯一拼块启发式的运行时间如何趋于一致：对于基准测试集所考察的大规模随机谜题，几乎所有谜题都由该启发式方法验证为非 SUSP。

运行时间变化。 最后，我们来看看图 6中混合算法运行时间的变化。对于较小的 s，运行时间分布远非正态分布——平均值远高于中位数

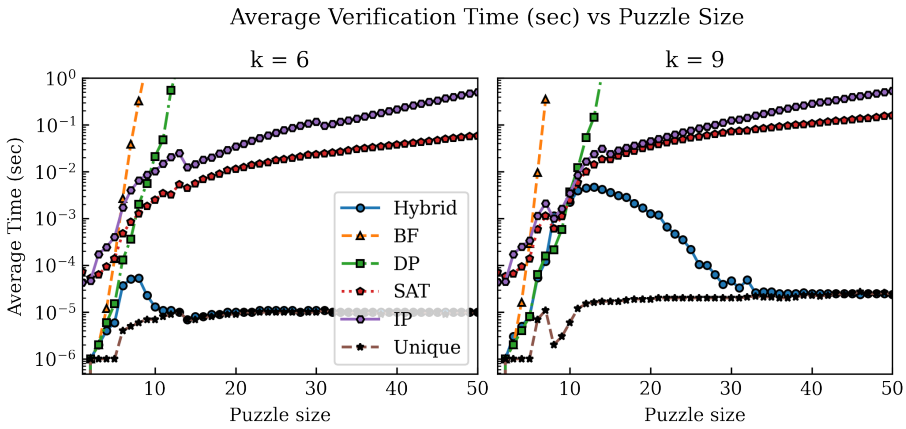

图 4：验证每个 $s \in [50], k \in \{6, 9\}$ 个随机 (s, k) 谜题时的平均运行时间对数图。图中描述了五种验证算法的行为：暴力搜索（BF）、动态规划（DP）、归约到可满足性（SAT）、归约到整数规划（IP）以及我们的混合算法（Hybrid）。同时还列出了唯一子行启发式方法的运行时间。

以及运行时间中间 50% 的范围。随着 k 增大，这种现象会变得更加明显。不过，我们发现，随着 s 增大，运行时间中位数会逐渐接近唯一块启发式的运行时间中位数；当 s 进一步增大时，平均运行时间也会趋于一致。这是因为，当快速启发式方法无法解决实例时，混合算法不得不运行慢上几个数量级的基于归约的算法。虽然此处未作图展示，但我们发现，可满足性验证器的运行时间分布范围比基于整数规划的验证器更大，尽管基于整数规划的验证器平均运行速度更慢。

总的来说，尽管经过 NP 完全问题归约，我们的混合验证算法在随机实例上仍具有相当良好的实际表现。

7.3 SAT 求解器的选择

在本文的会议版本中，我们仅考察了一种用于实现的 SAT 求解器：MapleCOMSPS。这是一种采用学习率分支启发式方法的冲突驱动求解器，并且在 2016 年 SAT 竞赛中表现出色 [7,23,5]。本文使用 SUSP 验证归约，对各种谜题（回顾

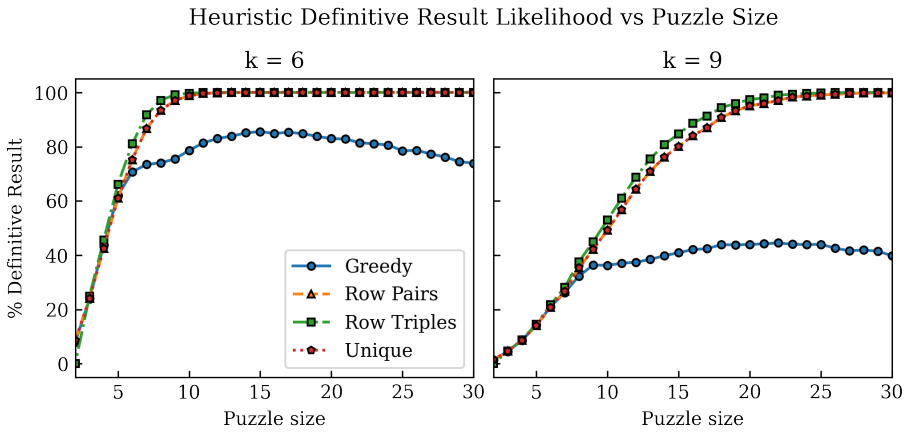

图 5：对于每种大小和宽度，展示各启发式方法在 10,000 s k $s \in [50]$ $k \in$ random (,)-谜题上产生确定性结果的可能性。其中，“行对”是 {6, 9} HeuristicDownwardClosed P 2 (,)，“行三元组”是 HeuristicDownwardClosed(P, 3)。图中绘制了行对的点，但由于唯一子行的点与其重合，因此难以看清。

第 3.4 小节），并考察了 35[2] 参加 2021 年 SAT 竞赛主赛道的求解器 [6]的性能。

我们选择了由大小取自集合的 (s, k)-谜题组成的基准测试实例

$$\{(2,2),(3,3),(5,4),(8,5),(14,6),(21,7),(30,8),(42,9)\}.$$

我们选择这些规模，是因为我们需要正例和负例，而这些规模代表了我们通过搜索找到的各个宽度下最大的强 USP。对于每种规模，我们创建了十个强 USP 谜题和十个非强 USP 谜题。为创建十个非 SUSP，我们随机生成该规模的谜题，并验证它不是强 USP。为创建十个强 USP，我们对每种规模都使用了搜索算法的结果。随后，我们将所有谜题通过 SAT 归约处理，为每个实例生成 .dimacs 文件。请注意，SUSP 对应不可满足实例，而非 SUSP 对应可满足实例。这个基准测试集总计包含 160 个实例。然后，我们对这 160 个实例文件分别运行全部 35 个求解器，并将每次运行的输出与预期结果进行核对。对于每次试验，我们记录 Linux 的 time 命令报告的用户CPU时间；如果程序运行超过 5000 秒仍未停止，则记录为超时（以模拟真实 SAT 竞赛的规则）。作为比较，我们还在同一组基准谜题上运行 MapleCOMSPS 求解器（来自

[2] 共有 39 个可满足求解器提交参加主赛道。我们对每个提交均使用默认构建配置。我们无法构建其中三个，而其中一个虽然构建成功，却在所有基准测试上反复崩溃，未生成结果。我们测试了其余 35 个。

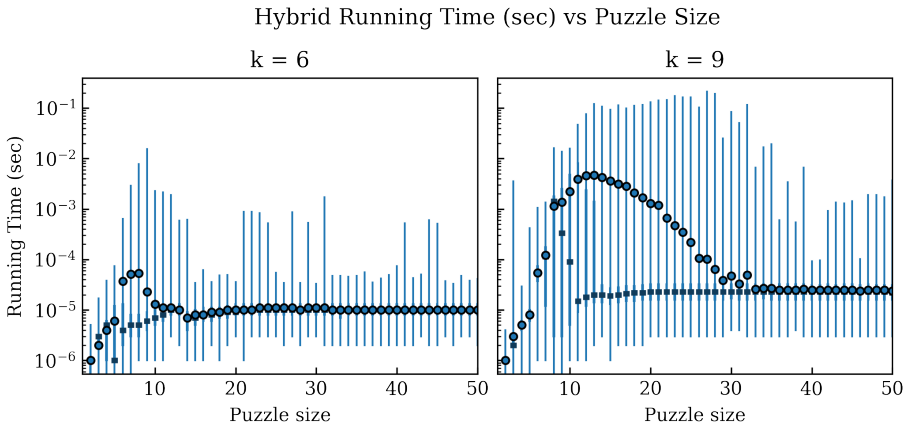

图 6：对数箱线图显示了混合验证算法在每个 $s \in [50], k \in \{6, 9\}$ 的 10,000 个随机 (s, k) 谜题上的运行时间分布。蓝色圆圈表示混合算法的平均运行时间。深蓝色方块表示中位时间。粗竖线表示中间 50% 的运行时间，细竖线表示每个 s 上运行时间的完整范围。

本文的早期版本）、我们基于整数规划的验证器（回顾第 3.5 小节）以及最终的混合验证算法。

为了比较每个求解器的结果，我们计算所有运行中完成每个实例所需的最长时间；如果某次运行超时，则该时间为 5000 秒。然后用该最长时间作除数，将所有运行时间归一化到区间 $[0, 1]$。我们通过对所有实例的相对运行时间求和，计算每个求解器的基准分数。 表 4列出了每个求解器的基准分数。

我们在本文会议版本中使用的求解器 MapleCOMSPS，其表现与 2021 年竞赛中得分最高的求解器相近。所有求解器记录的超时几乎都来自由 (30,8)-强唯一可解谜题和 $(42, 9)$-强唯一可解谜题导出的不可满足实例。基于 Gurobi 的验证器在可满足实例（非 SUSP）上的表现明显不如性能最佳的可满足性求解器，但在不可满足实例（SUSP）上的表现则显著更好。

图 7 展示了基于 Gurobi 的验证器与可满足性分数最高的五个求解器之间的性能对比。在该图中，每个求解器的实例完成时间按升序排列，因此曲线越靠左，表现越好。如果这不是一张对数图，曲线左侧的面积就会与表 4中的基准分数成正比。请注意，对于可满足实例，包括 MapleCOMSPS 在内的可满足求解器都遵循相似的轨迹。Gurobi 在所有可满足实例上的表现都差了一个数量级。虽然图中绘出了混合算法，但由于启发式方法在识别随机可满足（非 SUSP）实例方面非常有效，因此看不到它的曲线。对于不可满足

实例，情况则有所不同。对于规模较小、较容易的实例，Gurobi 的速度相对较慢；但对于规模较大、较困难的实例，其表现则显著优于可满足求解器。在较容易的不可满足实例上，各求解器的表现差异比对应的可满足实例更大，但这并未导致基准分数出现明显差异，因为相对完成时间的数量级较小。

对于不可满足实例，基准分数主要由超时次数决定，每次超时实际上都会使分数增加 1。事实上，可满足求解器的图表在实例编号 60 到 70 之间截断，因为剩余实例都会导致超时。最后请注意，对于规模较小的不可满足实例，混合算法的表现优于其他算法——这些实例适合采用暴力搜索和双向搜索算法。对于规模较大的实例，混合算法的表现比基于 Gurobi 的验证器差一个数量级。这是因为我们的算法经过调优，旨在处理远多于不可满足实例（SUSP）的可满足实例（非 SUSP）。此外，由于单边启发式方法在实践中能快速排除可满足实例，因此对于不可满足实例，混合算法会先运行这些启发式方法，但随后必须退回到基于 Gurobi 的验证器，从而产生一些额外开销。

最终，这些基准测试实验的结果表明，使用 2016 MapleCOMSPS 与使用 2021 年竞赛中的最佳求解器之间并无显著差异。尽管如此，我们仍在实现中选择 kissat-sc20221-sat 作为默认求解器，因为它在我们的可满足实例基准测试集上表现最佳。采用当前方法，Gurobi 对于可行地验证强唯一可解谜题至关重要。

基准测试实例和谜题，以及全部原始计时数据，均可在我们的代码仓库中找到[3]。

8 结论

我们首次研究了强 USP 的验证问题，并开发了用于验证和搜索强 USP 的实用软件。我们给出了宽度为 k 的强 USP 最大规模的精确结果（针对 $k \leq 5$），并改进了 $k \leq 12$ 的最大强 USP 规模的上界和下界。我们证明了强 USP 的若干与验证和搜索相关的性质。此外，我们还为可满足求解器生成了一组新的基准测试实例。

尽管我们的结果并未给出矩阵乘法运行时间的新上界，但它们表明这种方法前景可期。仍有许多开放问题：强 USP 验证是否是余 NP 完全问题？强 USP 容量的最大值是多少？有没有办法弥合由单个 SUSP 和无限 SUSPs 族所隐含的 ω 值之间明显的差距？对于 $k \geq 6$，最大规模强 USP 的紧确界是多少？这些界能否带来渐近更快的矩阵乘法算法？

[3] https://bitbucket.org/paraphase/matmult/src/main/data_集合/

我们工作的主要瓶颈是搜索空间的规模——要大幅缩小搜索空间，似乎需要新的见解。是否存在更便于有效搜索的强 USP 子类？有没有更适合在这一搜索空间中使用的搜索策略？

致谢

作者感谢匿名审稿人提出的详尽而周到的建议，帮助改进这项工作。

第二作者和第三作者感谢 Union College 提供本科生暑期研究奖学金，为他们的工作提供资助。第一作者感谢多年来以各种形式参与该项目的众多本科生，包括：Jonathan Kimber、Akriti Dhasmana、Jingyu Yao、Kyle Doney、Quoc An、Harper Lyon、Zachary Dubinsky、Talha Mushtaq、Jing Chin、Diep Vu、Hung Duong、Vu Le、Siddhant Deka、Baibhav Barwal、Aavasna Rupakheti。

参考文献

1. Alman, J., Williams, V.V.：已知矩阵乘法方法的进一步局限。载于：第9届理论计算机科学创新会议（ITCS）。LIPIcs。Leibniz 国际信息学会议论文集，第94卷，第25篇文章，共15页。达格施图尔城堡。Leibniz 信息学中心，德国瓦德恩（2018年）。https://doi.org/1025 242302018./LIPIcs.ITCS... Alman, J., Williams, V.V.：所有已知（以及一些未知）的矩阵乘法方法的局限。载于：第59届IEEE计算机科学基础年会（FOCS）。第580–591页（2018年10月）。https://doi.org/1000061 311092018./FOCS... Alon, N., Shpilka, A., Umans, C.：关于向日葵与矩阵乘法。计算复杂性**22**(2), 219–243（2013年）。https://doi.org/https://doi.org/101 41007000370130060./s---. Ambainis, A., Filmus, Y., Le Gall, F.：快速矩阵乘法：Coppersmith-Winograd 方法的局限。载于：第47届ACM理论计算研讨会（STOC）。第585–593页。ACM（2015年）。https://doi.org/102746554 511452746539./.. Anderson, M., Ji, Z., Xu, A.Y.：矩阵乘法：验证强唯一可解谜题。载于：Pulina, L., Seidl, M.（编）《可满足性测试理论与应用》（SAT）。第464–480页。施普林格国际出版，查姆（ 2020 10 1007 978 3 030 51825 7 32 ）。 6 https://doi.org/https://doi.org/./----. Balyo, T., Froleyks, N., Heule, M., Iser, M., Jarvisalo, M., Suda, M.（编）：《2021年SAT竞赛论文集：求解器与基准测试描述》。计算机科学系报告系列B，赫尔辛基大学计算机科学系，芬兰（ 2021 10138 333647 ）， 7 http://hdl.handle.net//. Balyo, T., Heule, M.J., Jarvisalo, M.：SAT竞赛：近期进展。载于：第31届人工智能促进协会人工智能大会（AAAI）（2017年）。https://doi.org/https://doi.org/10.1609/aaai.v31i1.10641 2016

8. Bj¨orklund, A., Husfeldt, T., Kaski, P., Koivisto, M.：用于参数化路径与打包问题的窄筛法。《计算机与系统科学杂志》87，119–139（ 2017 10 1016 2017 03 003 ）。9 ¨ 5 https://doi.org/https://doi.org/ . /j.jcss. . . . Blaser, M.：《快速矩阵乘法》。研究生综述第号，计算理论丛书，（2013 005 10 10 4086 2013）。https://doi.org/ . /toc.gs. . . Blasiak, J., Church, T., Cohn, H., Grochow, J.A., Umans, C.：哪些群有助于证明矩阵乘法的指数为二？arXiv预印本 arXiv:1712.02302（2017年）11. Cohn, H., Kleinberg, R., Szegedy, B., Umans, C.：用于矩阵乘法的群论算法。载于：第46届IEEE计算机科学基础年会（FOCS）。第379–388页（2005年10月）。https://doi.org/10 39 12 1109 2005 . /SFCS. . . Cohn, H., Umans, C.：快速矩阵乘法的群论方法。载于：第44届IEEE计算机科学基础年会（FOCS）。第438 1238217 13 449 2003 10 1109 2003 –页（10月）。https://doi.org/ . /SFCS. . . Coppersmith, D., Winograd, S.：通过等差数列进行矩阵乘法。《符号计算杂志》9(3)，251–280（1990年）。https://doi.org/https://doi.org/10 2 14 1016 0747 7171 08 80013 . /S - ( ) - . Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.：《算法导论》第三版。麻省理工学院出版社，美国，第3版（2009年）15. Croot, E., Lev, V.F., Pach, P.P.：中的无等差数列集合呈指数级小。《数学年刊》第331–337页（2017年）。https://doi.org/https://doi.org/10 7 16 4007 2017 185 1 . /annals. . . . . Davie, A.M., Stothers, A.J.：改进矩阵乘法复杂度的界。《爱丁堡皇家学会会刊A辑：数学》143(2)，351–369（2013年）17. Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F.J., Schrittwieser, J., Swirszcz, G. 等：利用强化学习发现更快的矩阵乘法算法。《自然》610(7930)，47 4 18 53（2022年）。https://doi.org/https://doi.org/ . /s - - - . Garey, M.R., Johnson, D.S.：《计算机与难解性：NP完全性理论指南》（1979年）19. Gurobi Optimization LLC：Gurobi优化器参考手册（2018年），http://www.gurobi.com 20. Kaminski, M.：有限域上多项式乘法复杂度的下界。《SIAM 计算杂志》34(4)，960–992（2005年）。https://doi.org/https://doi.org/10 9 40 21 1007 978 3 540 31856 2 . / - - - - . Korte, B., Vygen, J.：《组合优化》，第。卷。施普林格，柏林，海德堡（2012年）22. Le Gall, F.：张量幂与快速矩阵乘法。载于：第39届国际符号与代数计算研讨会（ISSAC）。第296–303页。ACM（2014 2608664 23 10 1145 2608628）。https://doi.org/ . / . . Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.：基于学习率的SAT求解器分支启发式方法。载于：可满足性测试理论与应用国际会议（SAT）。第123–140页。施普林格（2016年）。https://doi.org/https://doi.org/10 2 9 24 1007 978 3 319 40970 . / - - - - . McKay, B. D., Piperno, A.：实用图同构（II）。《符号计算杂志》60，94–112（2014年）。https://doi.org/https://doi.org/10.1016/j.jsc.2013.09.003, https://www.sciencedirect.com/science/article/pii/S0747717113001193

25. Oxley, J.G.：《拟阵理论》，第 3 卷。牛津大学出版社，美国 (2006) 26. Pan, V.Y.：Strassen 算法并非最优：用于构造矩阵运算快速算法的聚合、合并与消去三线性技术。载于：第 19 届计算机科学基础年度研讨会 (FOCS)。第 166–176 页 34 27 1978 10 1109 1978 。IEEE ( )。https://doi.org/https://doi.org/ 。/SFCS. . . Plimpton, S.J., Devine, K.D.：用于大规模图算法的 MPI 中的 MapReduce。Parallel Computing 37(9), 610–632 (2011)。https://doi.org/https://doi.org/

10 1016 2011 02 004 。/j.parco. . . .

28

Schonhage, A.：部分矩阵乘法与完全矩阵乘法。SIAM 计算杂志 10 0210032 29 3 434 455 1981 10 1137 ( ), – ( )。https://doi.org/ 。/ . Shpilka, A.：矩阵乘积的下界。SIAM 计算杂志 32(5), 1185 959910 30 1200 2003 10 1109 2001 – ( )。https://doi.org/ 。/SFCS. . . Strassen, V.：高斯消元并非最优。《数值数学》13(4), 354 02165411 31 356 1969 10 1007 – ( )。https://doi.org/https://doi.org/ 。/BF . Strassen, V.：张量的渐近谱与矩阵乘法指数。载于：第 27 届计算机科学基础年度研讨会 (FOCS)。第 49 52 32 54 页 1986 10 1109 1986 – 。IEEE ( )。https://doi.org/ 。/SFCS. . . Williams, V.V.：比 Coppersmith-Winograd 更快地进行矩阵乘法。载于：第 44 届 ACM 计算理论年度研讨会 (STOC)。第 887–898 页。ACM (2012)。https://doi.org/10.1145/2213977.2214056

求解器	SAT	不可满足	超时总数	
cadical-hack-gb	17.51	15.97	33.48	15
cadical-less-UP	19.81	16.14	35.95	15
cadical-PriPro	19.49	15.62	35.11	15
cadical-PriPro 无二进制文件	16.55	15.73	32.28	15
cadical-rp	19.08	15.78	34.85	15
cadical-sc2021	18.82	16.80	35.62	16
Cadical SCAVEL01	33.49	16.73	50.23	15
Cadical SCAVEL02	40.97	27.28	68.26	15
cleanmaple	30.44	18.93	49.37	17
CleanMaple PriPro	30.70	20.18	50.87	18
hCaD	19.70	16.52	36.22	16
hKis	13.15	17.30	30.45	16
kissat bonus	13.04	16.59	29.63	15
kissat cf	12.06	16.19	28.26	14
kissat gb	12.52	17.27	29.79	17
kissat-MAB	15.28	16.07	31.36	15
kissat-sat crvr gb	13.37	16.64	30.01	16
kissat-sc2021	12.32	16.08	28.40	14
kissat-sc2021-sat	**12.02**	16.06	**28.08**	14
kissat-sc2021-sweep	12.82	16.24	29.07	16
lstech maple	15.13	14.83	29.96	12
Maple_MBDR BJL6 Tier2	19.46	16.02	35.47	14
Maple_MBDR BJL7 Local	19.98	15.49	35.47	13
Maple_MBDR Cent PERM 10K	25.20	15.96	41.16	**12**
Maple_MBDR Cent PERM 75K	25.07	16.00	41.06	**12**
Maple simp21	12.53	16.72	29.26	15
MapleSSV	15.56	16.68	32.24	16
parafrost-nomdm-sc2021	18.11	15.56	33.67	14
parafrost-sc2021	24.15	15.61	39.76	14
Relaxed LCFTP	12.80	17.55	30.35	16
Relaxed LCFTP V2	13.97	16.17	30.14	**12**
Relaxed_LCMDCBDL BLB	15.38	15.95	31.33	14
Relaxed_LCMDCBDL SCAVEL01	13.95	16.08	30.03	15
Relaxed_LCMDCBDL SCAVEL02	25.45	79.43	104.88	17
slime	17.26	**14.73**	31.99	13
MapleCOMSPS	12.98	17.42	30.40	16
Gurobi	30.20	0.00	30.20	0
混合型	0.00	0.01	0.01	0

表 4：各求解器在我们的 SUSP 验证基准测试集上的得分。可满足和不可满足得分满分为 80，总分和超时次数满分为 160。得分越低越好，每列中各 SAT 求解器的最小值以粗体标出。表格上半部分列出了我们测试的 2021 年 SAT 竞赛中的 SAT 求解器 [6]。

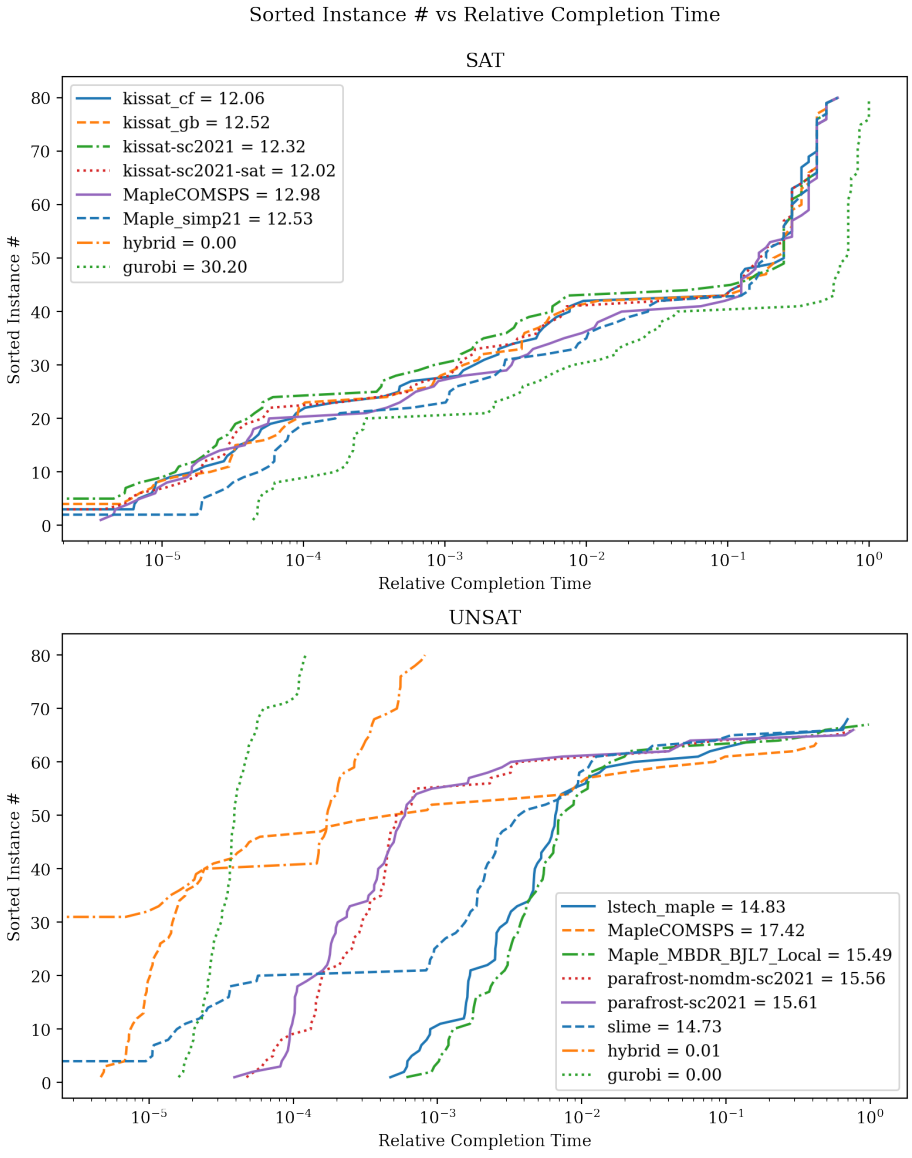

图 7：该实例类型中得分最高的五个求解器在可满足实例和不可满足实例上的排序后相对完成时间图。