arXiv:2303.01478v1 [cs.DS] 2 Mar 2023

通过推送-重标记实现更快速的拟阵打包与覆盖精确算法和近似算法*

Kent Quanrud†

摘要

拟阵是组合优化中的一个基础研究对象。涉及拟阵的三个密切相关且重要的问题包括：最大化独立集并集的大小，即 k个独立集（即 k重拟阵并）；计算 k 个不相交基（又称 拟阵基打包）；以及用 k 个基覆盖元素（又称 拟阵基覆盖）。这些问题自然地推广到元素具有整数容量和实值容量的情形。本文为这些问题以及最优增容、拟阵成员判定等其他密切相关的问题，研究了更快速的精确算法和/或近似算法。对于独立性预言机模型中的一般拟阵以及图拟阵，我们都改进了运行时间。改进的核心在于，基于 Frank 和 Miklós [FM12]先前的工作，为这些问题的整数容量版本开发了一种更快速且统一的 推送-重标记 算法。随后，我们从两个方向进一步发展这一算法。首先，我们为 k重拟阵并开发了一个更快速的增广路子程序；将其嵌入推送-重标记算法的近似版本后，可针对 k的某些参数得到更快速的精确算法。特别地，对于上述三个基本问题，在无容量限制情形下，我们获得了次二次查询运行时间。通过利用随机舍入将问题化归为小整数容量问题，我们还为具有实值容量的这些问题获得了更快速的近似算法。为此，我们针对拟阵中的基覆盖问题开发了一种新的随机舍入技术，该技术本身也可能具有独立研究价值。

1 引言

拟阵是组合优化中的基本研究对象。涉及拟阵的三个密切相关且重要的问题是：最大化 k 个独立集的并集大小（也称为拟阵并，即取同一拟阵的 k 个副本）、求出 k 个两两不相交的基（也称为拟阵基打包），以及用 k 个基覆盖底集（也称为拟阵基覆盖）。正如我们稍后将要说明的，这些问题可自然推广到整数容量和实值容量的情形。本文为这些问题及其他一些密切相关的问题开发更快的精确算法和／或近似算法。我们改进的核心在于为这些问题的整数容量版本开发了一种更快且统一的推送-重标记 算法。

推送-重标记框架最常与最大流联系在一起。由 [Gol85] 提出、随后经 [GT88] 改进并推广的流推送-重标记框架（参见 [Gol87]），是一种优雅的框架，为最大流带来了新的算法和视角（例如 [AO89；CM89；Tun94]），也适用于最小费用流 [GT90], 、参数流 [GGT89] 和有向最小割 [HO94；HRG00] 等相关问题。实证研究也表明，推送-重标记算法在实践中表现良好（例如 [CG97；CGM98]）。

*稿件于2022年7月提交至某会议。在2022年9月的答辩阶段进行了少量修改。

†普渡大学。 krq@purdue.edu。部分研究由美国国家科学基金会资助，资助编号为 CCF-2129816。

宽泛地说，流的推送-重标记算法与以往算法的不同之处在于，它通过局部改进，每次沿一条边“推送”流，而不是沿路径增广。推送-重标记算法并不维护一个 (s,t)-流（即在非终端点处保持流量守恒），而是维护一种称为预流 的流松弛形式，允许非终端顶点存在正盈余。该算法引入顶点标号，为每个顶点分配整数层级，以指导推送操作并实现多项式运行时间。

推送-重标记这一概念已被扩展（形式抽象得多），应用于更一般的组合优化问题，包括次模流 [FZ96], 次模交 [FZ92], 次模最小化 [FI00；FI03；IO09], 以及其他相关问题 [IMS00；IMS97]。最近，Frank 和 Miklós [FM12] 针对从拟阵划分到次模流等抽象组合优化问题，开发了更简单的强多项式时间推送-重标记算法。他们的工作建立在以往部分成果的基础上，并有助于将这些成果统一起来。我们的工作直接建立在 [FM12]的基础上。我们聚焦于上述拟阵问题，针对他们的框架提出了结构性见解和算法技术，既加速了精确算法，也扩展了该框架，从而得到快速近似算法。

我们从两个不同方面补充这些技术（它们属于推送-重标记框架）。首先，我们设计了一个专用于整数容量 k-重拟阵并的增广路子程序，可用于将推送-重标记框架（极快地）得到的近似解增广为最优解。我们指出，该增广路子程序需要推送-重标记框架所提供的额外结构才能发挥作用。对于某些参数范围，增广路径能够带来更快的运行时间；在无容量限制情形下，查询运行时间甚至低于二次。第二项扩展基于随机抽样，用于将实值容量问题的近似问题化简为具有（较小）整数容量的近似问题。其中一些随机舍入技术是新的，另一些则来自先前的工作 [Kar98]。化简后的整数容量情形尤其适合我们开发的推送-重标记算法，由此得到了一系列针对实值容量、具有近线性预言机调用和运行时间复杂度的近似算法。

1.1 结果概述

现在，我们概述本文所取得的改进后的运行时间。为简洁起见，关于每个问题的更详细描述，我们将推迟到论文后文对其进行完整分析时再给出。对于每个问题，我们说明新的运行时间，以及文献中最具竞争力且可直接比较的运行时间（据我们所知）。我们提供了指向每项结果的完整定理表述的链接。更多背景知识见附录A。我们讨论的问题都是拟阵优化中的基本且研究充分的问题，相关内容可见于 [Sch03]，我们将参阅该文献以获取更多背景知识。对于一般拟阵，我们在预言机模型中提出这些问题；对于图拟阵（即无向图的森林），我们也提出这些问题。

我们简要介绍描述这些结果所需的一些预备知识；包括更多记号和相关定义在内的其他预备知识见第 2 节。对于拟阵问题，令 n 表示元素的数量，令 r 表示拟阵的秩。我们假设可以访问一个独立性预言机，以查询元素集合 S 是否独立。我们采用标准且便于简化的约定：将通常需要组装并传输集合 $O(|S|)$ 至预言机的工作，也计入独立性查询的开销。对于图问题，令 m 表示（不同）边的数量，令 n 表示图中顶点的数量。当图的边具有整数容量时，令 U 表示总容量。

结果可 有三种形式：(1) 针对整数容量的精确算法，可生成 ce

整数解；(2) 针对整数容量的整数近似算法，可生成整数解；(3) 针对实值容量的近似判定算法。类型 (1) 通常有两种（无法比较优劣的）运行时间：一种由推送-重标记算法直接给出，另一种则结合推送-重标记与增广路径。第二类算法通常是相应的类型 (1) 推送-重标记算法的截断版本。第三类算法通常通过随机抽样，将问题归约为第二类问题而得到。所有针对整数容量的算法——包括类型 (1) 和类型 (2)——都是确定性的，并构造整数的原始解和对偶解，二者能够相互验证。针对实值容量的算法（类型 (3)）是随机蒙特卡洛算法，以高概率成功。它们构造近似的整数对偶解，但不构造原始解。

下面介绍我们的结果。我们首先介绍独立性预言机模型下拟阵的结果，随后介绍一组关于图拟阵的对应结果。

在无容量限制情形，k 重拟阵并问题要求找出 k 个基 B_1、……、B_k，使它们的并集大小最大。自然地，这一问题可推广到整数容量情形，此时我们将 B_1、……、B_k 中的每个元素按其容量计入，最多计入相应容量次。我们首先通过推送-重标记算法得到 $O(n+$最优值$\,r\log(kr))$ 次独立性查询的运行时间，其中最优值表示最优解的值（定理 3.6）。此外，结合推送-重标记算法与增广路径，我们还得到 $O\Big(n+$最优值$\,\sqrt{n'}\log(kr)\Big)$ 次独立性查询的运行时间，其中 $n'=\min\{n+$最优值$\log(r)\ rk\log(kr)\}$。注意，在无容量限制情形下，最优值 $\le n$ 且 $n'\le n\log(r)$，因此无论参数 k 和 r 如何选择，第二种运行时间至多为 $\tilde{O}\left(n^{3/2}\right)$ 次查询。这些运行时间可与无容量限制情形下的 $O\Big(\Big($最优值$^{3/2}+k\Big)nQ+$最优值$^{1/2}\,kn\Big)$ 相比较，其中 [Cun86]，中的 Q 表示对独立性预言机进行一次查询所需的时间。我们还可以将无容量限制的 k 重拟阵并归约为无权拟阵交（含有 nk 个元素且秩为最优值）；这将得到 $\tilde{O}\Big(nk\sqrt{}$最优值$\Big)$ 次查询的随机化运行时间[Bli21]。

我们还研究了具有容量限制的 k 重拟阵并近似算法。设 $\epsilon\in(0,1)$ 为给定参数；目标是达到至少为 $(1-\epsilon)$ OPT 的目标值。对于整数容量，我们获得了 $O(n+\text{OPT}\log(kr)/\epsilon)$ 次独立性查询的运行时间（定理 3.7）。该算法同时生成整数原始解和对偶解，并且二者相互证明其为 $(1\pm\epsilon)$-近似最优解。关于可比的运行时间，尽管 [Cun86] 未明确考虑近似问题，但 [Cun86] 隐含地包含了一个适用于无容量限制情形的 $(1-\epsilon)$-近似算法，其运行时间为 $O(knQ+(\text{OPT}\,nQ+kn)/\epsilon)$。此外，还可以将无容量限制的 k 重拟阵并归约为近似拟阵交，从而得到 $\tilde{O}\Big(nk\sqrt{\text{OPT}}/\epsilon\Big)$ 次查询的运行时间 [Bli21]。k 重拟阵并问题可推广到实值容量，此时需要在 k 基的分数线性组合上进行优化。对于这种情形，我们设计了一种随机化算法，其随机运行时间上界为 $O\big(n+(\text{OPT}/k)\log(n)\log\big(r\log(n)/\epsilon^2\big)/\epsilon^3\big)$ 次独立性查询。我们尚未发现文献中存在可比的算法。

接下来我们考虑基打包问题，并针对整数容量对其进行描述。目标是计算 k 个基，使得任一元素出现于这些基中的次数不超过其容量。显然，（精确求解）该问题可归结为 k重拟阵并，因此我们得到与上文所列相同的运行时间。除了上文提到的 [Cun86] 结果外，还可以将其与一种针对无容量限制情形的拉斯维加斯随机算法进行比较；该算法以高概率运行 $O\Big(n+r^{3/2}k^{5/2}\log^{3/2}(kr)\Big)$ 次独立性查询 [Kar98]。 [Kar98] 还给出了一种确定性算法，其运行时间为 $O\Big(n+(rk)^3\log^{O(1)}(kr)\Big)$。

我们还研究近似基打包。目标是打包 $(1-\epsilon)k$ 个基，或证明

不存在 $(1+\epsilon)k$ 个基的打包，其中参数 $\epsilon \in (0,1)$。这里，近似 k-基打包不能直接归约为近似 k 重拟阵并，不过我们采用了类似的技术。我们得到的运行时间为 $O(n + kr\log(n)\log(kr)/\epsilon)$（定理 4.2）。可以将其与用于无容量限制情形的随机蒙特卡洛算法进行比较；该算法运行 $\tilde{O}(n + r^3k/\epsilon^3)$ 次独立性查询 [Kar98]。对于实值容量，我们得到一个随机蒙特卡洛算法，运行 $O\left(n + r\log^2(n)(\log\log(n) + \log(1/\epsilon))/\epsilon^3\right)$ 次独立性查询，并以高概率成功（定理 7.4，第 25 页）。可以将其与 [CQ17] or 中的一个确定性 $\tilde{O}(nk/\epsilon^2)$ 查询时间算法，或运行 $O\left(n + r^3\log^{O(1)}(kr)/\epsilon^5\right)$ 次独立性查询的随机蒙特卡洛算法 [Kar98]进行比较。我们认为，可以结合 [CQ17; Kar98] 中的思路，得到一个运行 $\tilde{O}(n + r/\epsilon^4)$ 次独立性查询的随机蒙特卡洛算法。对于固定的 k，我们的随机化算法可以通过改进的二分搜索转换为一个随机化算法，用于计算最大值 k（我们称之为拟阵强度）。我们得到的运行时间略优于直接应用二分搜索所得的运行时间（定理 7.5，第 25 页）。

接下来我们讨论基覆盖。目标是计算 k 个基，使得对于每个元素，包含该元素的基的数量至少等于该元素的容量。如果无法做到这一点，我们期望能得到一个可行性的对偶证书。与打包一样，精确基覆盖可以通过 k 重拟阵并求解，因此我们也继承了相应的运行时间。据我们所知，当前可比的最佳运行时间是上文提到的 [Cun86] 针对无容量限制情形提出的 k 重拟阵并算法。（尤其是，我们不了解基覆盖领域有类似于 [Kar98] 或 [CQ17] 的进展；这些工作聚焦于基打包。）对于容量为整数的 $(1-\epsilon)$-近似基覆盖——值得注意的是，它不能归约为近似 k 重拟阵并——我们得到的查询运行时间为 $O(n\log(n)\log(kr)/\epsilon)$（定理 4.4）。（注意，对于基覆盖，$n \le kr$不失一般性。）对于实值容量，我们可以不失一般性地假设 $k = 1$。在这种情况下，我们面对的是拟阵成员资格问题，其目标是判断一个实值是否属于某个拟阵的独立集多面体。[Cun84] 给出了第一个强多项式时间算法，此外还有一个只需 $O(n^3r^2)$ 次独立性查询的精确算法 [Nar95]。我们得到了一种随机蒙特卡罗算法，其查询次数为 $O(n + r\ln(n)\ln(r/\epsilon)/\epsilon^3)$（定理 7.6）。同样，明确研究实数容量下近似基覆盖或拟阵成员资格的文献并不多。不过，我们认为 [CQ17]中的技术也可能得到一种运行时间为 $\tilde{O}(nk/\epsilon^2)$ 次查询的确定性算法；需要指出的是，当容量为整数时，这种算法不会产生整数打包。

我们讨论的拟阵领域的最后一个问题是最优增容。在整数版本中，给定一个具有整数元素容量、实值元素成本和整数参数 k 的拟阵，目标是计算一个最小成本扩展，以扩展容量，使所得的带容量拟阵具有拟阵强度 k。 [Cun85] 在图拟阵 ˜ 中提出并分析了这个问题。将其推广到拟阵后，他的算法可归约为 $O(n)$ 次 k 重拟阵并调用。结合上述 [Cun86] 的运行时间（此处 OPT $= kr$），得到运行时间 $O\left(\left((kr)^{3/2} + k\right)n^2Q + k^{3/2}rn^2\right)$。([Cun85] 也考虑实数容量，但我们不讨论这一情形。) 我们展示了如何通过一次调用我们的首个用于 k 重拟阵并的推送-重标记算法来计算最优增容；该算法运行时间为 $O(n + \mathrm{OPT}\, r\log(kr))$ 次独立性查询。更准确地说，我们证明，只需对算法的初始化方式稍作修改，推送-重标记算法便可直接解决最优增容问题。

现在我们概述图拟阵对应的结果。一般而言，我们采用上述针对一般拟阵的算法，并用适当的数据结构替代独立性预言机，以便直接回答查询。通常，这会将运行时间中的查询开销替换为多对数开销（或更低，例如反阿克曼函数 $\alpha(n)$。）对于 k-

折拟阵并——也就是说，最大化 k 个森林打包的总规模——仅使用推送-重标记即可得到运行时间 $O(m\alpha(n) + n$最优值$(\log(n) + \log(k)\alpha(n)))$（定理 5.2）；结合推送-重标记与增广路径，则可得到 $O\left(m\alpha(n) +$ 最优值$^{3/2}\log^2(nk)\alpha^2(n)\right)$（适用于连通图）（定理 8.15）。（注意，最优值 $\leq nk$。）可比的运行时间为 (a)$O(n^2k\log k)$，以及 (b) $O\left(\min\{U, nk\}\sqrt{k}(U + n\log n)\right)$。[GW92]。特别是，我们的第一种运行时间优于 (a)，第二种运行时间优于 (b)，因为最优值 $\leq \min\{U, nk\}$。对于 $(1-\epsilon)$-近似算法，我们得到整数容量下的 $O(m\alpha(n) +$ 最优值 $\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$，以及实数容量下的随机化 $O\left(m\alpha(n) + n\log^2(n)(\log(n) + \log(k)\alpha(n))/\epsilon^3\right)$ 时间。

基的打包可转化为生成树打包。这可以通过 k 重拟阵并算法求解，并且我们得到相同的运行时间。除了上文列出的 k 重拟阵并运行时间外，还可以与 $O(Un\log(m/n))$ [GW92] 以及 $O(kn\sqrt{U} + n\log n)$ [Gab91a]的运行时间进行比较。对于 $(1-\epsilon)$-近似算法，我们为近似整数树打包得到 $O(n +$ (最优值 $/k)\log(n)\log(r\log(n)/\epsilon^2))$。这可以与一种拉斯维加斯随机算法进行比较，该算法以高概率在 $\tilde{O}\left(kn^{3/2}/\epsilon^2\right)$ 时间内运行。可打包进一个图中的生成树最大数量称为网络强度。对于实值容量，我们得到一种随机化算法，可在 $/\epsilon^2))$ 时间内以 $(1\pm\epsilon)$-近似方式测试网络强度，其运行时间为 $O(n +$ (最优值 $/k)\log(n)\log(r\log(n)$，而将网络强度近似到 $(1\pm\epsilon)$-因子所需的运行时间略长。这个结果可以与一种运行时间为 $\tilde{O}(m/\epsilon^2)$ 的确定性算法，或一种在 [CQ17] 中运行时间为 $\tilde{O}(m + n/\epsilon^4)$ 的随机化算法进行比较。

按基覆盖对应于按生成树覆盖。同样，利用上文列出的 k重并算法，可获得整数容量下新的精确运行时间。除 k重并外，其他可比的运行时间还有 $\tilde{O}\left(U^{5/3}\right)$ 和 $O(Un\log(n))$[GW92]。对于整数容量下的$(1-\epsilon)$近似算法，我们得到的运行时间为$O(m\alpha(n) + nk\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$。覆盖一个图所需的最少生成树数量称为森林度。森林度可以在 $O(mn\log(n^2/m))$时间内精确计算 [Gab98]。我们得到一种随机化的 $O(m\alpha(n) + n\ln(n)(\log(n) + \log(\log(n)/\epsilon)\alpha(n))/\epsilon^3)$ 时间蒙特卡罗算法，用于近似测试森林度（达到$(1\pm\epsilon)$精度）；此外，还能以略长的运行时间将森林度近似到$(1\pm\epsilon)$因子以内（定理7.7和7.9）。与拟阵的情况一样，生成树近似覆盖方面的发展不如生成树打包方面多。不过，我们相信，[CQ17] 中的技术可以带来$\tilde{O}(m/\epsilon^2)$ 以及随机化的 $\tilde{O}(m + n/\epsilon^4)$ 时间算法，以$(1\pm\epsilon)$近似森林度（值）。

最后，采用与拟阵相同的归约，图中的最优加固问题可通过一次调用 k-重并的推送-重标记算法求解，其运行时间为$O(m\alpha(n) + n$最优值$(\log(n) + \log(k)\alpha(n)))$，其中最优值指 k-重并的最优值，且至多为 nk。这改进了 [Cun85] 将问题归约为调用 n 次 k-重并的做法，也改进了 $O(n^2m$ Gab98] 针对较小 k 的运行时间 $(n^2/m))$ log [。（[Gab98] 的算法和 [Cun85] 的归约也适用于实值容量。）

1.2 讨论与技术思路概述

（读者初次阅读时或许可以跳过本节，待对本文正文中介绍的算法有了一定技术了解后再回来阅读。）

如上文所述，我们改进后的运行时间始于一种共同的“拟阵推重标算法”，该算法建立在 [FM12] 的思想之上。在介绍我们的改进之前，先从高层次描述其框架可能会有所帮助，尤其是对于那些从流的角度出发的读者而言，这一视角颇为不同。

为了便于说明，我们非正式地描述其用于 k重拟阵并的算法。为简单起见，我们假设处于无容量限制情形；目标是计算一组 k 基 B_1、……、B_k，使其并的大小最大。若一个元素 e 不属于任何一个 B_i，则称其为未覆盖元素；若它属于某个 B_i，则称其为已覆盖；若它属于多个 B_i，则称其为超额元素。我们的目标是尽可能覆盖更多元素。一般而言，超额元素代表各个基之间被浪费的槽位。 [FM12] 的推送-重标记算法通过反复选择一个未覆盖元素 e ，并尝试将其交换到某个 B_i 中，使其保持未覆盖状态，从而操纵 B_1、……、B_k。若要取得直接进展，就必须将它换出一个超额元素 d；否则，并的大小将保持不变。然而，即使 B_1、……、B_k 尚未达到最优解，这种有利的交换也可能无法实现。虽然可以将 e 与其他已覆盖但并非超额的元素进行交换，但这种做法如何提供帮助并不明显。这类似于将流从一个具有盈余的非终端顶点推送到另一个非终端顶点；我们并不清楚这是否正在朝汇点推进。这正是推送-重标记框架中重标记部分发挥作用的地方。每个元素都被标记为一个整数层级，而超额元素保持在层级 0。交换受到限制：只有当某个元素 e 被交换到 B_i 中，以替换一个比 B_i低一层的元素 d 时，交换才被允许。从某种意义上说，由 e 处于未覆盖状态所代表的“盈余”向下移动一层，到达 d，因而更接近处于层级 0 的超额元素。这些思想最终促成了 [FM12]中关于该算法为何能在多项式时间内终止的更为复杂的论证。

上面只是一个略去细节、正确性证明，甚至算法完整描述的概要（更具技术性的论述留待后文）。不过，它开始展现出一种类比：熟悉的最大流推送-重标记框架，与[FM12]所提出的抽象版本之间的类比。我们不再给图的顶点标记，而是给拟阵的元素标记。在流问题中，我们沿边将流从一个顶点推送到另一个顶点；而在拟阵中，我们“推送”元素交换，以一个元素换另一个元素，同时保持可行性。与流问题类似，推送仅限于向“更低”的层级进行；当无法再推送时，就会执行重标记操作，提高某些元素的层级。这样做可能会揭示由顶点水平集诱导的违反约束，这与流问题中由标签显现最小割的情形类似。

在分析其推送-重标记框架时，对于从拟阵划分到次模流等问题， [FM12] 着重证明算法具有强多项式性质，同时尽可能保持算法和分析简单。（从历史上看，对于这些抽象问题，获得强多项式运行时间并非易事。）为此，[FM12] 并未直接界定运行时间，而是给出了框架所执行“基本操作”（即推送和重标记操作）次数的最坏情况多项式界。（例如，上述 $O(n^5)$ 重拟阵并问题需要 k次基本操作。）可以证明，识别并执行一次基本操作需要多项式时间（以及对独立性预言机的查询），但需要指出的是，由于拟阵的抽象性质以及预言机模型，这远不如流问题中的情形简单。例如，在流问题中，边明确指出了我们可以在哪里“推送”；而在拟阵中，寻找一对可交换元素 e_1 和 e_2 （如上所述）可能需要对底集中的元素进行嵌套循环，并在每次内层迭代中进行一次独立性查询。还有一些 [FM12] 所处理的技术问题，在流问题中并没有明显的对应情形。

我们一直将这个框架称为拟阵推重标算法，以区别于流的推重标记算法。起初，我们被 [FM12], 在概念上的吸引力所打动，并开始针对一些更具体的问题开发算法，希望至少能找到比现有算法更简单或更实用的替代方案（类似于如今流的推重标记算法所扮演的角色）。鉴于 [FM12], 中的界很大，抽象层级也很高，因此，对于一些基础且研究已久的问题——这些问题还有看起来更直接的替代方法——能否通过拟阵推重标算法获得有竞争力的界，并不明朗。令我们惊讶的是，在开发出

拟阵推送-重标记框架中的若干新思路后，我们实际上能够改进其中几个问题目前已知的最佳界。

如上所述，推送-重标记算法会操作一组基，并为每个元素指定层级。算法在遵循一组“推送-重标记不变量”的同时修改这些基和层级；这组不变量由 [FM12], 提出，用来约束对这些基所做的交换。我们通过引入一个称为高度的整数值参数，对这些不变量稍作扩展。高度是任一未覆盖元素的最低层级。引入高度的一个动机是便于分析快速近似算法，下文将对此进行讨论。核心拟阵推送-重标记算法的运行时间表示为高度的函数。高度参数的选取取决于具体问题，以及我们寻求的是精确解还是近似解。

如上所述， [FM12] 中的算法并不十分具体（更谈不上高效），而且对这些算法的分析仅限于界定推送/重标记操作的次数，而非运行时间。我们补充了运行时间分析，并引入了若干进一步提升运行效率的思路。其中一些思路颇为微妙，且细致而局部，例如为 k重拟阵并中的高度推导更精细的界，或在生成树问题中更谨慎地应用数据结构。此外，还有一些更广泛、较容易单独阐述的思路，下面将重点介绍。

基的逐层递减顺序。 回想一下，推送-重标记算法会针对输入参数 k维护一组 k 基。每当我们想要将一个元素 e 交换进解中（姑且这么说）时，众多的基会带来一个算法问题：如何快速找出适合交换 e的基。一个直观的做法是遍历所有基，但这会非常慢，尤其是在凸组合较大的情况下。本文提出了一个重要的新思路，它似乎特别适用于拟阵和推送-重标记框架：按照“基的逐层递减顺序”维护这些基。该顺序的定义基于由标签确定的各基的上水平集。我们要求并维护这些基，使其按如下顺序排列：对于每个层级，一个基的水平集张成下一个基的水平集。直观而抽象地说，这种顺序的单调性与推送-重标记框架中元素重新标记和交换的方式配合得十分巧妙。具体而言，我们证明，若干简单且必要的贪心规则足以在选择不同的推送和重标记操作时维持递减顺序。此外，若基已按逐层递减顺序排列，我们就不必再像上文（宽泛地）描述的那样遍历这些基，而可以通过二分搜索，在对数时间内通过查询找出合适的基。递减顺序还允许我们在重新标记元素时沿层级进行二分搜索，从而进一步缩短运行时间。

通过截断进行近似。 在推送-重标记框架中，一种获得快速近似解的重要技术是：只需使用截断高度即可得到近似解，而截断高度主要取决于所需的精度。这类似于众所周知的联系：在边/点不相交路径、二分匹配和拟阵交等问题中，最短增广路径的长度与当前解的质量之间存在联系（例如，[Cun86； ET75； HK73]）。在本文中，拟阵推送-重标记算法以一种更适合运用这类论证的视角来构建拟阵优化问题。对于许多问题，我们证明，只需使用 $O(1/\epsilon)$ 层级，或 $O(\log(n)/\epsilon)$ 层级，即可获得一个 $(1 \pm \epsilon)$-因子近似。我们指出，导出 $O(1/\epsilon)$ 界或 $O(\log(n)/\epsilon)$ 界的论证各不相同，而且需要进行一些额外分析，才能确定每个问题适用哪种论证。

如前所述，除了对拟阵推送-重标记算法的改进外，我们还开发了另外两种技术，以拓展推送-重标记算法的适用范围。

随机舍入为将适用于整数容量的推送-重标记框架拓展到实数容量，并实现快速近似算法，我们采用随机舍入。更具体地说，我们通过随机抽样将容量离散化，从而有效地将容量缩减为较小的整数值（至多为 $O(\ln(n)/\epsilon^2)$）。对于装箱问题，我们所需的随机化技术已有 [Kar98]提供。对于 k重拟阵并和覆盖问题，此类技术尚不为人所知，而且[Kar98] 中的论证似乎也无法推广。我们针对这些尚未解决的问题提出了新的分析。有趣的是，这一分析还以一种可以说更简单的证明重新得出了 [Kar98], 的结果。（尤其是，新分析不依赖于随机收缩算法。）参见定理7.1、第7.1节。用于拟阵基覆盖和 k重拟阵并的随机化技术可能也具有独立的研究价值。

增广路径。 历来，解决此处所讨论问题最常用的方法是使用增广路径。增广路径也可用于将近似（整数）解扩展为最优解（这正是我们的应用场景）。对于（例如） k重拟阵并的增广路径，我们维护一个由 k 个独立集 $I_1, \ldots, I_k$ 组成的打包，并尝试每次扩展一个元素。扩展这样的打包并非易事，可能需要进行复杂的交换序列，才能为新元素腾出位置。可以证明，在此图中，指定源点与汇点之间的最短路径就是一条增广路径。最直接的方法是显式构建整个辅助图，逐一测试每条可能的弧是否存在，然后在所得图上运行广度优先搜索。然而，该图可能十分稠密，构建图会成为瓶颈。

相反，我们探索隐式寻找目标路径的技术。从较高层面来看，要识别从指定源点可达的所有顶点，并不需要遍历所有弧。事实上，对于头部已被“标记”为已探索的顶点，我们无需测试相应弧是否存在。话虽如此，我们确实需要能够识别指向“未标记”辅助顶点的弧。为了帮助我们搜索这些有用的弧，我们引入额外的不变量。其中一个不变量是让 $I_1, \ldots, I_k$ 保持“递减顺序”，其方式类似于拟阵推送-重标记算法中递减基的逐层次序。这限制了我们在每次迭代中允许进行的增广。第二个不变量适用于单次增广路径搜索。在探索过程中标记辅助顶点时，我们还要求 $I_1, \ldots, I_k$中对应于已标记辅助顶点的元素子集也按“递减顺序”排列。为此，我们引入一个“预搜索”子程序，在标记辅助顶点之前对其调用。借助这一额外结构——让 $I_1, \ldots, I_k$ 以及 $I_1, \ldots, I_k$ 中的已标记元素（可以说）都按递减顺序排列——隐式遍历辅助图便容易得多。一次搜索的总运行时间约为辅助图中每个顶点对应对数数量的独立性查询。（引理8.1，第8节。）

对于适当选择的 ϵ，我们使用增广路子程序，将推送-重标记算法生成的一个 $(1-\epsilon)$-近似解扩展为最优解。然而，要应用我们的增广路径算法，我们要求推送-重标记算法的输出满足某些不变量：也就是说，基 $B_1, \ldots\ldots, B_k$ 应包含一个打包 $I_1, \ldots\ldots, I_k$，且两者总容量相同，并按递减顺序排列。幸运的是，确实如此，而证明的关键在于 $B_1, \ldots\ldots, B_k$ 本来就按层级递减顺序排列。

值得注意的是，近期已有一些与隐式遍历辅助图相关的类似思路，被用于加速拟阵交的增广路径算法 [BBMN21；Bli21；CLSSW19；Ngu19], ，而拟阵交是一个密切相关的问题。不过，要达到我们的运行时间，还需要针对k重拟阵并的额外专门思路，以及拟阵推重标算法所带来的额外结构。

未来工作。 我们认为，这里的一些思路可用于实数容量下的精确推送-重标记算法，并将此留待未来工作。我们还从推送-重标记的角度，继续为更抽象的问题开发算法，例如多拟阵交和次模流。这些主题需要更为抽象的工具，其概念重点也不同于本文的论述；本文的内容则更专注于拟阵。这些进展将留待未来工作。

1.3 结构安排。

本文其余部分安排如下。

- 在第 2 节中，我们介绍初步定义和符号。
- 在第 3节中，我们介绍并分析适用于整数容量的拟阵推送-重标记算法，该算法用于 k重拟阵并。
- 在第 4节中，我们分析整数容量下的基打包与覆盖问题。
- 在第 5节中，我们为图拟阵实现拟阵推送-重标记算法。
- 在第 6 节中，我们分析最小成本强化问题。
- 在第 7 节中，我们开发适用于实值容量的随机化算法。
- 在第 8 节中，我们开发增广路径算法。
- 更多背景知识见A。

致谢。 感谢 Chandra Chekuri 提供宝贵的反馈，也感谢审稿人提供有益的反馈和额外的参考文献线索。

2 预备知识

我们简要介绍拟阵，更多背景知识请参阅 [Sch03] 。拟阵 M =(N , I) 由有限底集 N 和一族独立集 I $\subseteq 2^{\mathcal{N}}$构成，并满足以下三个性质：(i) $\emptyset \in$ I (ii) $A \in$ I 且 $B \subset A$ 蕴含 $B \in$ I；(iii) A、$B \in$ I 且 $|B| > |A|$蕴含存在 $e \in B \setminus A$，使得 $A \cup \{e\} \in$ I。拟阵的秩函数是定义在 N 的子集上的整数值函数，其中 rank(S) 是包含于 S中的最大独立集的基数。我们令 span(S) $= \{e :$ rank($S + e$) $=$ rank(S)$\}$ 表示由 S张成的元素集合。对于独立集 $I \in$ I 和元素 $e \in$ span(I) $\setminus I$， $I + e$中存在唯一的极小集合，称为 e 在 I 中的回路，记作 $C(I + e)$。

独立多面体是向量集合 $x \in \mathbb{R}^{\mathcal{N}}_{\geq 0}$ ，其中的向量可表示为独立集示性向量的凸组合。我们用 $P_{\mathcal{I}}$ 表示独立多面体。对于 $k > 0$，我们将 $kP_{\mathcal{I}}$ 按因子 k放大；等价地， $kP_{\mathcal{I}}$ 表示满足 $x/k \in P_{\mathcal{I}}$ 的向量集合x 。

在我们所有的问题中，元素都配有容量 $u \in \mathbb{R}^{\mathcal{N}}_{\geq 0}$。容量通常为整数，只有在第7 节中我们考虑实值容量。对于集合 $S \subseteq \mathcal{N}$，为简化记号，我们用 S 表示其容量之和

$$u(S) \stackrel{\text{def}}{=} \sum_{e \in S} u(e).$$

我们研究的许多问题都会处理一组 k 基 B_1，……，B_k 这些基可能包含重叠的元素。在这种情况下，对于一个元素 $e \in \mathsf{N}$，令

$$x(e) \stackrel{\text{def}}{=} |\{i \in [k] : e \in B_i\}|$$

如果 $x(e) < u(e)$，则称一个元素为未覆盖；如果 $x(e) \geq u(e)$，则称其为已覆盖；如果$x(e) \leq u(e)$，则称其为可行装填；如果 $x(e) > u(e)$，则称其为超额装填。

3 整数容量下的拟阵推重标算法

本节介绍拟阵推送-重标记算法。我们主要讨论 k重拟阵并问题，并给出一种能够加速 k重拟阵并算法的算法，该算法见于 [FM12]。 k重拟阵并问题曾在第 1.1节中简要介绍过，我们在此重新介绍该问题。

输入由一个拟阵 $\mathcal{M} = (\mathcal{N}, \mathcal{I})$、整数容量 $u : \mathcal{N} \to \mathbb{N}$以及一个整数 k组成。该输入定义了以下对偶极小极大问题， [Nas67] 已证明它们的目标值相等：

$$\text{maximize} \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \text{ over } k \text{ bases } B_1, \ldots, B_k \in \mathcal{I}; \tag{1}$$

$$\text{minimize } k \operatorname{rank}(S) + \sum_{e \in \bar{S}} u(e) \text{ over all sets } S \subseteq \mathcal{N}. \tag{2}$$

（$x(e)$ 的定义见第2节。）(1) 称为 k 重拟阵并问题；我们将 (2) 简称为其对偶问题。如前所述， [Cun86] 给出了一个针对该问题无容量版本（即对于所有 e， $u(e) = 1$ ）的算法，运行时间为 $O\Big((\text{最优值}^{3/2} + k)nQ + \text{最优值}^{1/2}\, kn\Big)$。此外，[FM12] 在无容量限制情形下给出了 $O(n^5)$ “基本操作”（定义如下）的上界。我们注意到， [Cun86; FM12] 考虑的是更一般的拟阵并情形，其中每个 B_i 都是不同拟阵（定义在同一底集上）中的一个基。

我们首先在第[FM12], 3.1 节介绍拟阵推送-重标记框架的高层组件，该框架基于3.1。随后，我们分析如何利用该框架实现最优性，改进 [FM12] 中的界，并在第3.2节将分析扩展到近似算法。最后，我们在第3.3节介绍并分析更快的算法。

3.1 拟阵推送-重标记框架的组成部分

拟阵推送-重标记算法维护一个映射 $\ell : \mathsf{N} \to \mathsf{Z}_{\geq 0}$，为每个元素分配一个层级，初始时所有层级均设为 0。算法还维护 k 个基 B_1，…，$B_k \in \mathsf{B}$，初始时它们都设为任意选取的同一个基。

我们的讨论经常会根据元素的层级对其分组，因此引入以下记号会很方便。对于固定层级 j，我们用 N^j 表示层级为 j 的元素集合。我们还用 $\mathsf{N}^{\leq j}$ 表示层级小于或等于 j 的元素集合；类似地，还有 $\mathsf{N}^{<j}$、$\mathsf{N}^{>j}$ 和 $\mathsf{N}^{\geq j}$。

算法通过插入元素来更新基 B_1、……、B_k。对于每个基 B_i 和每个元素 $e \in B_i$，算法会跟踪 e 被插入 B_i 时的层级。对于 $i \in [k]$ 和 $j \in \mathsf{Z}_{\geq 0}$，令 $B_{i,j} \subseteq B_i$ 表示 e 中那些在 e 处于层级 j 时被插入 B_i 的元素子集。（注意，$B_{i,j}$ 不等于 $B_i \cap \mathsf{N}^j$。）令 $B_{i,\geq j}$ 表示在 $\ell(e)$ 至少为 j 时插入 B_i 的元素 e 所组成的集合。类似地，我们还有 $B_{i,\leq j}$、$B_{i,>j}$ 以及 $B_{i,<j}$。

在拟阵推重标算法中， k 基 B_1、……、B_k 构成最大化问题（1）的候选解。次水平集 $\mathcal{N}_{\leq j}$（其中 $j \in \mathbb{N}$）表示最小化问题（2）的候选解。该框架旨在确保

$$\sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \leq k \operatorname{rank}(\mathcal{N}_{\leq j}) + u(\mathcal{N}_{>j})$$

对所有 j。如下面第 3.2 节所示，算法终止时会确定一个层级 j，使上述不等式恰好或近似取等。因此， B_1、……、B_k 和 $\mathcal{N}_{\leq j}$ 相互证明对方在各自的问题上恰好或近似最优。

拟阵推送-重标记算法满足 [FM12]提出的以下不变量。这里，如果一个元素 e 被包含在至少 个基中，则称其为 (e) 覆盖的；否则称其为 未覆盖。

(I) $\ell(e) = 0$ 对于所有严格多于 $u(e)$ 个基中的元素 e 。

(II) 对于 $i = 1$、……、k 以及所有层级 j、 $B_{i,\ \geq j}$ 张成 $\mathcal{N}_{>j}$。

(III) 所有未覆盖元素 e 都有 $\ell(e) \geq h$，其中参数为 $h \in \mathbb{Z}_{\geq 0}$。

给定基的配置和层级赋值，高度 定义为任一未覆盖元素的最小层级。也就是说，高度是满足不变量 h 的最小值 (III)。

拟阵推重标算法本质上由两个操作组成，[FM12]称为基本操作。

1. 推送（交换）：给定一个未覆盖元素 e、一个基 B_i 和一个元素 $d \in B_{i,\ \ell(e)-1}$，使得 $B_i - d + e \in \mathrm{I}$，将 B_i 替换为 $B_i - d + e$。[1]
2. 重标记：给定一个未覆盖元素 e，将 $\ell(e)$ 增加为 $\ell(e) + 1$。

为保持不变量 (II)，只有在没有可用推送操作时， e 未覆盖元素才能被重新标记。如第 1.2 节所述，为元素 e 识别可行的推送是一个计算瓶颈，这一点不同于流。[FM12] 证明了推送和重标记操作会保持不变量 (I)–(III)，我们将其视为已知事实。

3.2 通过拟阵推重标不变式实现最优性

我们最终将设计一种算法，尝试尽快获得一个由基和层级组成的配置，使其对于给定的高度参数 $h \in \mathbb{N}$ 满足上述不变量 (I)–(III)。首先，我们将展示 h 的特定取值如何与式（1）和（2）中的对偶极小极大问题的良好解相关联。这里我们分别分析精确解和近似解。

3.2.1 精确解

我们首先考虑方程 (1) 和 (2) 的精确解。此前， [FM12] 证明，在拟阵并（拟阵不同）这一更一般的情形下，高度 $\Omega(n)$ 足以导出一个精确解。对于 k重拟阵并这一特定情形，我们有以下更强的界 $r + 2$。

[1]请注意， d 可能与 e相同；后者是在 e 处于较低层级时插入的。

引理 3.1. 假设 $B_1, \ldots, B_k$ 和 $\ell : \mathcal{N} \to \mathbb{R}_{\geq 0}$ 满足不变量不变量 *(I)–(III)*，且高度为 $h > r + 2$。那么存在一个层级 j 使得

$$\sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \geq u(\mathcal{N}_{<j}) + k \operatorname{rank}(\mathcal{N}_{\geq j}).$$

这证明了 $B_1 \cup \cdots \cup B_k$ 是一个最大解，而 $\mathcal{N}_{\geq j}$ 是一个最小对偶解 .

证明。 作为 $j \in \mathbb{Z}_{\geq 0}$ 的函数，$\operatorname{rank}(\mathbb{N}_{\geq j})$ 是一个整数，并从 0 单调递增至 r。根据鸽巢原理，存在一个层级 $j \in \{1, \ldots, h\}$，使得 $\operatorname{rank}(\mathbb{N}_{\geq j}) = \operatorname{rank}(\mathbb{N}_{>j})$。 我们将证明在这一选择下的断言。根据不变量 (II)，$B_{i,\geq j}$ 张成 $\mathbb{N}_{>j}$，因此

$$\sum_{i=1}^{k} |B_{i,\geq j}| \geq k \operatorname{rank}(\mathcal{N}_{\geq j}) = k \operatorname{rank}(\mathcal{N}_{>j}).$$

根据不变量 (I)，$\mathcal{N}_{\geq j}$ 中没有元素超额装填，因此

$$\sum_{i=1}^{k} |B_{i,\geq j}| \leq \sum_{i=1}^{k} |B_i \cap \mathcal{N}_{\geq j}| = \sum_{e \in \mathcal{N}_{\geq j}} \min\{x(e), u(e)\}.$$

最后，根据不变量 (III)，$\mathcal{N}_j$ 中的所有元素都已覆盖，因此

$$u(\mathcal{N}_{<j}) = \sum_{e \in \mathcal{N}_{<j}} \min\{u(e), x(e)\}.$$

综上可得

$$u(\mathcal{N}_{<j}) + k \operatorname{rank}(\mathcal{N}_{\geq j}) \leq u(\mathcal{N}_{<j}) + \sum_{i=1}^{k} |B_{i,\geq j}| \leq \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\},$$

如所期望的。 □

3.2.2 近似解

现在我们转向近似算法。这里我们证明，高度 $O(1/\epsilon)$ 足以为原问题和对偶问题都得到 $(1+\epsilon)$-近似算法。

引理 3.2. 设 $\epsilon \in (0,1)$。假设 $B_1, \ldots, B_k$ 和 $\ell : \mathcal{N} \to \mathbb{R}_{\geq 0}$ 满足不变量 *(I)–(III)*，其高度为 $h > 1/\epsilon + 2$。那么存在一个层级 j 使得

$$u(\mathcal{N}_{<j}) + k \operatorname{rank}(\mathcal{N}_{\geq j}) \leq (1+\epsilon) \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\}.$$

这证明 $B_1, \ldots, B_k$ 是一个 $1/(1+\epsilon)$-近似最大解，并且 $\mathcal{N}_{\geq j}$ 是一个 $(1+\epsilon)$-近似最小对偶解。

证明。 由于 $h > 1/\epsilon + 2$，存在一个索引 $j \in \{1, \ldots, h-1\}$ 使得

$$|B_{1,j}| + \cdots + |B_{k,j}| \leq \epsilon(|B_{1,\geq 1}| + \cdots + |B_{k,\geq 1}|). \tag{3}$$

We 将证明对于这一选择，以下断言成立： j。由于每个 $B_{i,\ \geq j}$ 张成 $\mathcal{N}_{>j}$ （依据 （II） ），

$$k \operatorname{rank}(\mathcal{N}_{>j}) \leq \sum_{i=1}^{k} |B_{i,\geq j}|.$$

此外，根据 j 方程（3）的选择，我们有

$$\sum_{i=1}^{k}|B_{i,\geq j}| \leq \sum_{i=1}^{k}|B_{i,>j}| + \epsilon\sum_{i=1}^{k}|B_{i,\geq 1}| \leq x(\mathcal{N}_{>j}) + \epsilon x(\mathcal{N}_{\geq 1}).$$

现在，由于 $x(e) \leq u(e)$ 对所有 $e \in \mathcal{N}_{>0}$ 以及 $u(e) \leq x(e)$ 对所有 $e \in \mathcal{N}_{<h}$ 根据不变量 (I) 和 (III)，我们有

$$u(\mathcal{N}_{\leq j}) + k\,\mathrm{rank}(\mathcal{N}_{>j}) \leq u(\mathcal{N}_{\leq j}) + x(\mathcal{N}_{>j}) + \epsilon x(\mathcal{N}_{\geq 1}) \leq (1+\epsilon)\sum_{e\in\mathcal{N}}\min\{u(e), x(e)\},$$

正如所期望的。 □

3.3 更快的拟阵推送-重标记算法

现在已经证明，推送-重标记不变量 (I)–(III)如何导出精确解或近似解，接下来我们转向算法问题：如何计算满足这些不变量且高度为给定值 h的配置。我们得到的运行时间关于 h的函数关系，将在下述定理中说明。下文中，令 OPT 表示方程（1）和（2）的共同最优值。

定理 3.3。 给定拟阵 $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ 以及参数 k， $h \in \mathbb{N}$，存在一种算法，其运行时间上界为 $O(n + h\mathrm{OPT}\log(kr))$ 次独立性预言机调用，可计算基B_1、...、B_k 以及层级 ℓ： $\mathcal{N} \to \mathbb{Z}_{\geq 0}$，使其满足推送-重标记不变量 *(I)–(III)*，且高度为 h。

作为比较， [FM12] 证明了 $O(nh)$ 次基本操作足以在无权情形下达到高度h （这并未考虑其他计算因素，例如识别基本操作）。本小节其余部分将用于证明定理 3.3。

如上所述，该算法初始化 $B_1, \ldots, B_k$为任意一个基。在这里，贪心算法可以通过 $O(n)$ 次独立性查询，在与之成正比的时间内计算出一个基。起初，我们设定 $\ell(e) = 0$ 对所有元素 e均成立。除了不变量 (I)–(III)之外，我们还对 $B_1, \ldots, B_k$和 ℓ施加以下单调性条件。

定义 3.4。令 $B_1, \ldots, B_k$ 为一列独立集，并令 $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ 为每个元素指定整数层级。若对于所有索引 $i = 1, \ldots, k-1$ 以及所有层级 $j \in \mathbb{Z}_{\geq 0}$， $B_1, \ldots, B_k$ 满足 $B_{i,\geq j}$ 张成 $B_{i+1,\geq j}$，则称其为单调递减 （或简称递减）。

给定 h，算法的目标是达到这样一种配置：对于所有未覆盖元素 e， $\ell(e) \geq h$。为此，推送-重标记算法（由 [FM12] 描述）会反复选择一个满足 $\ell(e) < h$ 的未覆盖元素 e，原则上，它希望将 e 推送到某个 B_i 中以覆盖e，或者重标记 e，使 $\ell(e)$ 更接近 h。对于更快的算法，我们描述了一种称为贪心插入的新过程，它沿层级和基进行二分搜索，以更积极地将 e 放入第一个可用的基中（可以这么说）。要论证这一过程模拟了合法的推送-重标记操作序列，还需要一些说明；我们将在描述该过程后证明这一点。

贪心插入元素 e：令 $e \in \mathcal{N}$ 为未覆盖，且满足 $\ell(e) < h$。

1. 如果 e 由 $B_{k,h-1}$ 张成，则设置 $\ell(e) = h$ 并返回。2. 否则，找出第一个满足 $B_{k,>j}$ 不张成 e的层级 j。3. 否则，搜索第一个满足 $B_{i,>j}$ 不张成 e的索引 i。设置 $\ell(e) = j+1$，并将 e 交换到 B_i 中，以替换满足 $B - d + e \in \mathrm{I}$ 的元素 $d \in B_{i,j}$。

综合起来，整体算法如下。我们最初将所有基 $B_1,\dots,B_k$ 设为任意一个基，并对所有 e设置 $\ell(e)=0$。只要存在一个未覆盖元素 e，满足 $\ell(e)<h$，我们就贪心地插入 e；这要么将 e 放入一个基中，要么设置 $\ell(e)=h$。

接下来需要分析该算法的正确性和运行时间。我们先讨论正确性，特别是首先证明 `greedy-insertion`保持不变量(I)–(III)。证明需要用到这样一个事实：贪心插入之前，各基按递减顺序排列；因此，我们也会证明贪心插入能保持基的递减顺序。

引理 3.5。假设 $B_1,\dots,B_k$ 和 $\ell:\mathcal{N}\to\mathbb{Z}_{\geq 0}$ 满足不变量*(I)–(III)*，且 $B_1,\dots,B_k$按递减顺序排列。那么，贪心插入元素 e会保持不变量*(I)–(III)* 并维持基的递减顺序。

证明。我们先证明贪心插入能保持递减顺序。只需考虑在步骤3中执行交换的情况，因为否则基不会发生变化。因此，假设我们将元素 e 换入层级 B_i 上的一个基 j，以交换出层级 d 上的一个元素 $j-1$。令 $B_1,\dots,B_k$ 表示交换前的各个基。令 $B_i'=B_i-d+e$ 表示交换后的基；这是基序列中唯一的变化。只需将 B_i' 与其前后的基 B_{i-1} 和 B_{i+1} 进行比较 B_i （假设 B_i 分别不是第一个或最后一个基）。无需检查严格大于 j' 的层级 j，因为这些水平集没有变化。

先考虑 B_{i-1} （当 $i>1$ 时）。对于任意层级 $j'\leq j$，我们有

$$B'_{i,\geq j'}\subseteq B_{i,\geq j'}\cup\{e\}\overset{(a)}{\subseteq}\operatorname{span}(B_{i-1,\geq j'}),$$

对两边取张成，便得到所需的子集不等式。这里，(a) 是因为 $B_{i,\ \geq j'}\subseteq$张成$(B_{i-1,\ \geq j'})$，这是由 $B_1,\dots,B_k$ 按降序排列这一事实得出的；同时也因为根据索引 i 的选择，$e\in$ 张成 $(B_{i-1,\ \geq j})$。

现在考虑 B_{i+1} （当 $i<k$ 时）。在层级 j，我们有

$$B_{i+1,\geq j}\overset{(b)}{\subseteq}\operatorname{span}(B_{i,\geq j})\overset{(c)}{\subseteq}\operatorname{span}\Big(B'_{i,\geq j}\Big),$$

对两边取张成，便得到所需的子集不等式。这里，(b) 是由已有的降序排列得出的。(c) 是因为 $B'_{i,\ \geq j}=B_{i,\ \geq j}+e$。对于层级 $j'<j$，我们有

$$B_{i+1,\geq j'}\overset{(d)}{\subseteq}\operatorname{span}(B_{i,\geq j'})\subseteq\operatorname{span}(B_{i,\geq j'}+e)=\operatorname{span}\Big(B'_{i,\geq j'}+d\Big)\overset{(e)}{=}\operatorname{span}\Big(B'_{i,\geq j'}\Big),$$

对两边取张成，便得到所需的子集不等式。这里，(d) 是由降序排列得出的，(e) 是由单调性得出的，(f) 是因为 $B'_{i,\ \geq j'}+d=B_{i,\ \geq j'}+e$，而 (g) 是因为 d 由 $B'_{i,\ \geq j'}=B_{i,\ \geq j'}+e-d$ 张成。

接下来，我们证明贪心插入能够维持推送-重标记不变量 (I)–(III)。更具体地说，我们将证明贪心插入模拟了一个合法的基本操作序列；如前所述，[FM12] 已经证明基本操作能够维持这些不变量。

给定 e，假设我们反复尝试对 e 进行推送或重标记，直到 (a) 执行一次推送，或 (b) 将 $\ell(e)$ 增加到 h。在事件 (a) 中，在所有可以于固定层级交换 e 的基 B_i 中，我们特意选择索引最小的基 B_i。i这描述了一个有效的基本操作序列，因此会保持不变量 (I)–(III)。我们将证明贪心插入模拟了这一过程。

固定一个未覆盖元素 e，其中 $\ell(e)=j$。首先，我们断言：当且仅当 e 不被 $B_{i,\ \geq j}$张成时，元素 e 才能被推送到一个基 B_i （在该层级）。事实上，若 $B_{i,\ \geq j}$ 张成 e，则所有

可与 e 交换的元素都在 $B_{i,\ \geq j}$中，特别地，它们不在 $B_{i,\ j-1}$ 中，这正是进行一次推送所要求的。反之，若 $B_{i,\ \geq j}$ 不张成 e，则 $B_i + e$ 的唯一回路必定包含一个元素 d，其中 $d \in B_{i,\ <j}$。此外，根据不变量 (II)，$B_{i,\ \geq j-1}$ 张成 e，因此该回路也是 $B_{i,\ \geq j-1} + e$的唯一回路。这意味着 $d \in B_{i,\ j}$，并且可以与 e交换。

其次，我们注意到，对于所有 i，如果 $e \notin$ 张成($B_{i,\ \geq j}$)，那么对于所有 $i' \geq i$，$e \notin$ 都张成$\left(B_{i',\ \geq j}\right)$。这是由基的递减顺序所推得的，该顺序意味着集合 span($B_{i,\ \geq j}$) 构成一个嵌套递减的集合序列。这一观察带来以下两点。首先，如果 $e \in$ 张成($B_{k,\ \geq j}$)，那么所有 $B_{i,\ \geq j}$都张成 e，因此我们可以安全地增加 $\ell(e)$。其次，如果对于某个 B_i，$e \notin$张成($B_{i,\ \geq j}$)，那么我们可以通过二分搜索找到索引最小的基 B_i，使得 $e \notin$张成($B_{i,\ \geq j}$)。

综合以上各点，回顾一下，我们要论证的是：贪心插入模拟了一个推送重标记过程，该过程会反复重标记 e，直到出现以下情况之一：$\ell(e) = h$，或者我们可以将 e 交换进一个基 B_i，以替换元素 $d \in B_{i,\ell(e)-1}$；此时，它会将其交换进第一个满足条件的 B_i。如上所述，当且仅当 $e \notin$ 张成$\left(B_{i,\geq \ell(e)}\right)$时，这种交换才可行。此外，如上所述，当且仅当 $e \notin$ 张成$\left(B_{k,\geq \ell(e)}\right)$时，后一种情况才可行。由于集合 $B_{k,\geq j}$ 在 j 中构成嵌套递减的集合序列，而 span$(\cdots)$ 是单调递增的集合函数，因此我们可以通过二分搜索找到第一个（最小的）索引 j，使得 $e \notin B_{k,\geq j}$。这个索引 j 恰好就是模拟的推送-重标记过程最终会将 $\ell(e)$ 设为的层级。假设 $j < h$，模拟的推送-重标记过程随后会找到第一个可以将 e交换进去的 B_i。根据上述观察，这个基 B_i 与步骤 3 中通过二分搜索找到的基相同。 □

引理 3.5 通过不变量 (I)–(III)证明了该算法的正确性。为了完成定理 3.3 的证明，还需证明运行时间上界。

每次我们贪心插入一个未覆盖元素 e 时，都可以将其归因于以下两种情况之一：(a) 将 $\ell(e) = h$设为某个值；或 (b) 增大某些 $i \in [k]$ 和 $0 \leq j < h$对应的 $B_{i,>j}$ 的大小。每个元素的层级只会被设为 h 一次，因此类型 (a) 的插入共有 n 次。每次类型 (a) 的插入都需要进行一次独立性查询。为了界定类型 (b) 插入的次数，我们注意到，对于固定层级 $j > 0$，跨越 i 的集合 $B_{i,>j}$ 构成了式 (1) 的一个可行解，因此其总大小至多为最优值。由于共有 h 个层级，类型 (b) 的插入次数至多为 hOPT。综上，第一类贪心插入至多有 n 次，第二类至多有 O(OPT h)次。

第一类贪心插入只需进行一次预言机调用。下面考虑第二类。使用二分搜索时，步骤 2 中的第一次搜索需要 $O(\log h)$ 次探测。更好的是，通过标准的倍增技巧，我们可以调整二分搜索，使第一次搜索也至多需要 $O(1 + \ell)$ 次探测，其中 ℓ 是该元素向前移动的层级数。我们可以将这些探测的代价归因于至少 ℓ 个集合的秩 $B_{i\text{增加},\ >j}$。步骤 3 中的搜索需要 $O(\log k)$ 次探测。在这两种情况下，每次探测都需要进行一次独立性查询。

执行交换时，我们还需要快速确定要移除的元素 d。为此，我们可以在 $B_{i,j}$ 上按照插入顺序维护一棵平衡二叉树，并使用二分搜索快速确定 d的最后一个可能选择。（这是第一个元素 d，使得 $B_{i\text{中的所有元素}j-1}$ 之前的 d元素与 $B_{i\text{中的元素}\geq j}$一起都不张成 e）。这需要进行 log(r) 次独立性查询。我们注意到，步骤 3 只会针对类型 (b) 的 OPT h 查询中的一次调用。因此，这就得到了总运行时间。

至此，整数容量下更快的拟阵推送-重标记算法介绍完毕。

3.4 统合起来

结合定理 3.3 的运行时间与引理 3.1 和 3.2所需的高度，我们得到以下整数容量下精确和近似 k重拟阵并的运行时间。

定理 3.6. 对于整数容量，可以通过 $O(n + \text{最优值}\, r\log(kr))$ 次独立性查询，计算最大 k 重拟阵并及其对偶解。

定理 3.7. 对于整数容量，可以在不超过 $O(n + \text{最优值}\log(kr)/\epsilon)$ 次独立性查询的时间内，计算一个 $1-\epsilon$)-近似最大 k 重拟阵并，以及一个 $1+\epsilon$)-近似最小对偶解。

4 基打包与覆盖

现在我们转向将拟阵打包和覆盖为基的问题。这些问题已在第 1.1 节简要介绍，下面我们将更详细地描述它们。

在基打包问题中，给定一个整数 k，目标是计算 k 个基 $B_1, \ldots, B_k$，使得对于所有 $e \in$ N， $x(e) \leq u(e)$。若对于所有 e， $B_1, \ldots, B_k$ 满足 $x(e) \leq u(e)$，则称其为一个打包。Edmonds [Edm65a] 证明，存在一个由 k 个基组成的打包 $B_1, \ldots, B_k$，当且仅当

$$u\left(\bar{S}\right) \geq k(r - \operatorname{rank}(S))$$

对于所有集合 S。

在基覆盖问题中，目标是计算 k 个基 $B_1, \ldots, B_k$ 使得 $x(e) \geq u(e)$对于所有 $e \in \mathcal{N}$。这样的 $B_1, \ldots, B_k$ 称为一个覆盖。Edmonds [Edm65b] 证明，存在覆盖 $B_1, \ldots, B_k$ 由 k 个基组成，当且仅当

$$k \operatorname{rank}(S) \geq u(S)$$

对于所有集合 $S \subseteq$ N 。

上述关于基打包和基覆盖的两种对偶刻画，都可以通过 k重拟阵并的对偶刻画得到，其中 [N个67] 拟阵并的对偶刻画已在第 3 节介绍。

如前所述，精确基打包与覆盖可直接归约为 k重拟阵并。例如，对于打包，存在由 k 个基组成的基打包，当且仅当存在 k 个基，其并集的大小为 kr。对于覆盖，存在由 k 个基组成的基覆盖，当且仅当存在 k 个基，其并集的大小为 n。当找不到打包或覆盖时， k重拟阵并算法给出的对偶解也为基打包或覆盖问题提供了不可行性证书。

人们或许会认为， k重拟阵并的近似算法也能用于打包和覆盖的近似算法，但事实并非如此。以 k 基的无容量限制打包为例。 k重拟阵并的近似算法会输出 k 个基，其并集大小为任意并集最大大小的$(1-\epsilon)$倍。如果存在 k 个不相交基，那么该并集的大小至少为$(1-\epsilon)kr$ 个元素总数。然而，这并不等同于$(1-\epsilon)k$ 个不相交基。大小为$(1-\epsilon)kr$ 的并集既不能证明至少存在$(1-\epsilon)k$ 个不相交基，也不能否定存在 k 个不相交基。

有容量打包以及有容量和无容量覆盖也存在类似差异。对于所有这些问题，我们给出稍强的分析，以获得所需形式的近似。这里的主要区别在于，要获得 $(1 \pm \epsilon)$-近似，高度需要增加一个对数因子。因此，打包和覆盖近似算法的所有运行时间都比近似 k 重拟阵并的运行时间多一个对数因子。

4.1 打包

我们首先考虑无容量基打包的近似算法，定理 4.2 声称其运行时间为 $O(n\log(1/\epsilon) + n\log(kr/\epsilon))$ 次独立性查询。下面我们证明，高度$O(\log(n)/\epsilon)$ 意味着可获得 $(1-\epsilon)$-近似。于是，根据定理 4.2，其运行时间可通过运行高度为 $O(\log(n)/\epsilon)$ 的无容量拟阵推送-重标记算法得到；依据定理 3.3。

引理 4.1. 设 $\epsilon \in (0,1)$，且 $k \in \mathbb{N}$。设 $B_1, \ldots, B_k$ 是一族 k 基，且 $\ell : N \to \mathbb{Z}_{\geq 0}$ 是满足拟阵推重标不变式（I）–（III）的层级赋值，其高度为 $h = O(\log(n)/\epsilon)$。那么，要么 (a) 基 $B_1, \ldots, B_k$ 构成 k 基的可行打包；要么 (b) (M, u) 不存在 $(1+\epsilon)k$ 基的可行打包。

证明. 假设 $B_1, \ldots, B_k$ 不是正规打包。由不变量（I），这意味着 $N_0 \neq \emptyset$。现在，$u(N_{\leq j})$ 在 $j \in \{0, \ldots, h\}$ 上单调不减；对于 $j = 0$，其下界为 1（如上所述），对于 $j = h$，其上界为 n。因此，必定存在一个层级 $j \in \{1, \ldots, h-1\}$，使得 $u(N_j) < (\epsilon/(1+\epsilon))u(N_{\leq j})$。固定这样的 j。根据 j 的选取，我们有

$$u(\mathcal{N}_{\leq j}) + (1+\epsilon)k \operatorname{rank}(\mathcal{N}_{>j}) < (1+\epsilon)(u(\mathcal{N}_{\leq j}) + k \operatorname{rank}(\mathcal{N}_{>j})) - (1+\epsilon)u(\mathcal{N}_j) \tag{4}$$

根据不变量(III)，$\mathcal{N}_{\leq j}$ 中的所有元素都已覆盖，因此

$$u(\mathcal{N}_{\leq j}) \leq (1+\epsilon)u(\mathcal{N}_{<j}) \leq (1+\epsilon)x(\mathcal{N}_{\leq j}).$$

根据不变量(II)，每个 $B_{i,\geq j}$ 都覆盖 $N_{>j}$，且 $N_{\geq j}$ 中的元素均未超额装填，因此

$$k \operatorname{rank}(\mathcal{N}_{>j}) \leq \sum_{i=1}^{k} |B_{i,\geq j}| = x(\mathcal{N}_{\geq j}).$$

综上，我们有

$$u(\mathcal{N}_{\leq j}) + (1+\epsilon)k \operatorname{rank}(\mathcal{N}_{>j}) \leq (1+\epsilon)x(\mathcal{N}) \leq (1+\epsilon)kr.$$

重新排列后，我们有

$$u(\mathcal{N}_{\leq j}) \leq (1+\epsilon)k(r - \operatorname{rank}(\mathcal{N}_{>j})),$$

因此，$N_{>j}$ 证明强度至多为 $(1+\epsilon)k$。 □

应用高度参数为 $h = O(\log(n)/\epsilon)$的拟阵推送-重标记算法（定理 3.3），根据引理 4.1，可得到以下 $(1-\epsilon)$-近似基打包算法。

定理 4.2。 一个$(1-\epsilon)$-近似基打包，或一个$(1+\epsilon)$-近似的不可行性证书，可在运行时间上界为 $O(n + kr\log(n)\log(kr)/\epsilon)$ 次独立性查询的时间内计算得出。

4.2 覆盖

现在我们转向覆盖问题，首先讨论无容量覆盖。在这里我们证明，$O(\log(r)/\epsilon)$ 的高度足以获得 $(1+\epsilon)$-近似；请注意，这一界限比上面对无容量限制打包进行近似时的界限略好。

引理 4.3。 设 $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ 是一个拟阵，且 $k \in \mathbb{N}$。设 B_1，……， $B_k \in \mathcal{B}$ 是由 k个基组成的集合，并且 $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ 是满足推送-重标记不变量 *(I)–(III)* 且高度为$h = O(\log(r)/\epsilon)$的一组层级。那么以下两种情况必有其一：

1. B_1,… , B_k 是一个覆盖，或者
2. 对于某个 索引 j，$\mathcal{N}_{>j}$ 证明任何覆盖都需要超过$(1-\epsilon)k$ 个基 *ng.*

证明。假设 $B_1, \ldots, B_k$ 不是一个覆盖。那么至少存在一个未覆盖元素；此外，根据不变量 (III)，任何未覆盖元素都属于 $\mathbf{N}_{\geq h}$。作为 $j \in \{0, \ldots, h\}$的函数，$\mathbf{rank}(\mathbf{N}_{\geq j})$单调不增，在 $j = 0$处至多为 r，并且由于 $\mathbf{N}_{\geq h}$非空，在 $j = h$ 处至少为 1。由于 h 至少为 $O(\log(r)/\epsilon)$，因此必定存在一个层级 $j \in \{0, \ldots, h\}$，使得 $\mathbf{rank}(\mathbf{N}_{>j}) \geq (1-\epsilon)\,\mathbf{rank}(\mathbf{N}_{\geq j})$。取这样的 j。我们有

$$u(\mathcal{N}) > \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \tag{5}$$

因为 $B_1, \ldots, B_k$ 不是一个覆盖。由于 $\mathbf{N}_{\geq j}$ 中的元素没有超额装填，我们有

$$\sum_{e \in \mathcal{N}_{\geq j}} \min\{u(e), x(e)\} = \sum_{e \in \mathcal{N}_{\geq j}} x(e) = \sum_{i=1}^{k} |B_{i,\geq j}|.$$

根据不变量(II)，每个 $B_{i,\geq j}$ 张成 $\mathbf{N}_{>j}$ 因此

$$\sum_{i=1}^{k} |B_{i,\geq j}| \geq k \operatorname{rank}(\mathcal{N}_{>j}).$$

另一方面，由于 $\mathbf{N}_{<j}$ 中的所有元素都已覆盖，我们有

$$\sum_{e \in \mathcal{N}_{<j}} \min\{u(e), x(e)\} = u(\mathcal{N}_{<j}).$$

代入式（5），现在得到

$$u(\mathcal{N}) > k \operatorname{rank}(\mathcal{N}_{>j}) + u(\mathcal{N}_{<j}).$$

最后，由于根据 j 的选择，$\mathbf{rank}(\mathbf{N}_{>j}) \geq (1-\epsilon)\,\mathbf{rank}(\mathbf{N}_{\geq j})$，我们得到

$$u(\mathcal{N}) > (1-\epsilon)k \operatorname{rank}(\mathcal{N}_{\geq j}) + |\mathcal{N}_{<j}|.$$

整理后可得 $(1-\epsilon)k\mathbf{rank}(\mathbf{N}_{\geq j}) < |\mathbf{N}_{\geq j}|$。根据上面的对偶刻画，这意味着任何覆盖都需要超过 $(1-\epsilon)k$ 个基。 □

将 $h = O(\log(r)/\epsilon)$ 代入拟阵推送-重标记算法（定理 3.3），我们得到以下基覆盖近似算法。（代入定理 3.3时，注意到 最优值 $= n$，且 $n \leq kr$。）

定理 4.4. 对于整数容量，可以在不超过 $O(n \log(n) \log(kr)/\epsilon)$次独立性查询的时间内，计算出 $(1+\epsilon)k$ 个基的覆盖，或为任何 $(1-\epsilon)k$ 个基的覆盖问题提供不可行性证书。

5 图形拟阵推重标

在本节中，我们考虑图形拟阵的特殊情形下的拟阵推重标算法框架；也就是说，无向图的森林。这将为第 1.1 節中提到的若干图问题带来算法。

定理 *5.1.* 给定一个具有整数容量的图以及整数 k 和 h，在 $O(m\alpha(n) +$ 最优值 $h(\log(n) + \log(k)\alpha(n)))$ 时间内，可以计算出一系列生成树 k，...，T_k 和层级 $\ell: E \to Z_{\geq 0}$，使其满足高度为 h 的推标不变量（*I*）–（*III*）。

证明。 我们使用以下数据结构来实现拟阵推送-重标记算法。对于每棵生成树 T_i，我们维护一棵动态树 [ST83], ，其中的边标记有各自的层级。给定一条边 e 和一棵树 T_i，我们可以通过查询由 e 所形成的环上层级最低的边，查询满足 $T_{i,\geq j}$ 张成 e 的最大层级 j。这也使我们能够在 O(log n) 时间内检索出一条要交换出去的边 d。我们总共进行 O(最优值 h) 次此类交换。

我们还会针对每个基 B_i 和每个层级 $j \in$ {1、.....、h}，维护一个表示 $B_{i,\ \geq j}$ 连通分量的并查集数据结构。这样，我们可以在 $\alpha(n)$ 时间内查询边 e 是否被森林 $B_{i,\ \geq j}$ 张成。对于所有 i 和 j，我们在这些并查集数据结构中总共进行 O(OPT h) 次插入。此类查询最多进行 $O(m +$ OPT $h\log(k))$ 次。 □

定理 5.1，结合关于 3.1、3.2、4.1 和4.3 的 k重拟阵并、基打包和基覆盖的最优性条件（包括精确算法和近似算法），可得图拟阵的以下运行时间。

我们首先讨论 k 重拟阵并问题。对于图而言，更自然的表述是计算森林 F_1、.....、F_k 的最大总容量打包。[2] 我们有以下精确算法和近似算法的运行时间。

定理 5.2。 k 森林的最大容量打包以及对偶最小化问题，可以在 $O(m\alpha(n) + n$最优值$(\log(n) + \log(k)\ \alpha(n)))$ 时间内求解，其中 最优值 表示最优规模。

定理 5.3。 可以在$O(m\alpha(n) +$ 最优值$\log(n(\log(n) + \log(k)\alpha(n))/\epsilon)$ 时间内，计算出$1-\epsilon(1-\epsilon)$-近似的k 森林最大容量打包，以及$1+\epsilon(1+\epsilon)$-近似的最小对偶解。

对于生成树的打包与覆盖，除了定理 5.2所隐含的精确算法外，我们还有以下近似算法。

定理 5.4。对于整数容量图，存在一种算法，可在$O\left(n +\right.$(最优值 $/k)\log(n)\log\left(r\log(n)/\epsilon^2\right))$ 时间内输出 k 生成树的打包，或输出网络强度小于 $(1+\epsilon)k$的证书。

定理 5.5。对于整数容量图，存在一种算法，可在$O(m\alpha(n) + nk\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$ 时间内，输出由 k 棵生成树构成的覆盖，或输出一份证书，证明其强度小于 $(1+\epsilon)k$。

6 最小成本强化

本节讨论第 1.1 节中介绍的最小成本强化问题。我们主要讨论更一般的拟阵情形；图情形是其特例。

[2]正如 wi th 树，如果一个森林集合中没有任何元素 e 出现在超过 $u(e)$ 个森林中，则该集合是一个打包。

设 M $=(N, I)$ 是一个具有 n 个元素、秩为 r 的拟阵，设 $u: N \to Z_{\geq 0}$ 是一组整数容量，设 $c: N \to R_{>0}$ 是一组实值成本，并设 $k \in N$。成本 $c(e)$ 表示将 $u(e)$ 增广 1 的成本。目标是计算使 u 达到强度 k 所需的最小成本增广。

我们的论证与 Cunningham 算法密切相关 [Cun85], ，该算法是针对这一问题的首个强多项式时间算法。Cunningham 研究的是无向图的网络强度；在此，我们将介绍其算法向拟阵的一种直接推广。

1. 通过 k 重拟阵并，计算满足 $y \leq u$ 的最大点 $y \in kP_{\mathcal{I}}$。

2. 通过贪心方式，将 y 扩展为 $kP_{\mathcal{I}}$ 中的一个整数点 $y+z$，其中 $z \in Z_{\geq 0}^{\mathcal{N}}$ 的计算方法如下。初始时，对所有元素 e，设 $z(e) = 0$。然后，按成本递增的顺序依次处理每个元素 e，将 $z(e)$ 设为满足 $y + z \in kP_{\mathcal{I}}$ 的最大值。$z(e)$ 的最大值可通过二分搜索得到；每次探测都会调用一个 k 重拟阵并算法，以判断候选值 $z(e)$ 是否可行。

关于该算法的完整论证，请参阅[Cun85]。请注意，该算法需要多次调用拟阵划分算法：首先计算初始点 y，然后对每个元素 e调用对数次，以得到正确的 $z(e)$。

或许令人惊讶的是， k重拟阵并算法在第3 节中提出，却实际上可以一步（可以这么说）解决加固问题。只需做一个小小的调整：初始基（在第3节中原本可以任意选取）都必须设为最小费用基，我们将其记为 B_0。回想一下，推送-重标记算法应用于一组整数容量 u 和一个参数 k 时，会生成一组 k 基 $B_1, \ldots, B_k$，使

$$\sum_{e \in \mathcal{N}} \min\{u(e), x(e)\},$$

其中， $x(e)$ 表示包含 e 的基 B_i 的数量。由 $y(e) = \min\{u(e)\ x(e)\}$（对所有 e）定义的向量 y 满足 Cunningham 算法的第 1 步。下文将证明的关键断言是：如果 k 重拟阵并算法中的初始基全部设为最小费用基 B_0，那么算法结束时， x 描述了某个最优强化解 z 对应的 $y+z$。因此，可以直接从 x 和 y 得到 z。这样一来，总运行时间恰好与 k 重拟阵并相同。与 [Cun85] 相比，这里有两项提速：一项来自更快的 k 重拟阵并算法，另一项则来自完全省去第二阶段。下面的引理将关键断言形式化。

引理 6.1。 令 B_0 为相对于 c的最小费用基，并考虑经过调整、使所有初始基均设为 B_0的精确推送-重标记 k重拟阵并算法。令 $B_1, \ldots, B_k$ 为推送-重标记拟阵分割算法输出的k 个基，并令 $x \in \mathbb{Z}_{\geq 0}^{\mathcal{N}}$ 为向量，其中$x(e)$ 表示对每个 $e \in \mathcal{N}$，包含 e 的基的数量。定义 $z \in \mathbb{R}_{\geq 0}^{\mathcal{N}}$ 为

$$z(e) = \min\{0, x(e) - u(e)\} \text{ for } e \in \mathcal{N}.$$

那么， z 是一个最小费用强化。

证明。除上述所述的 $B_1, \ldots, B_k$和 x外，令 ℓ为容量约束 k重拟阵并算法生成的层级集合。则 $B_1, \ldots, B_k$、x和 ℓ满足第 3.1 节中的不变量（I）–（III）。令 $C = \{e : x(e) > u(e)\}$。根据不变量（I），对所有 $e \in C$，均有 $\ell(e) = 0$。

断言 *1*. $C \subseteq B_0$。

为此，请注意，初始时，只有初始基 $e \in B_0$ 中的元素才有 $x(e) > u(e)$。此后，只有当 $x(e) < u(e)$ 时，坐标 $x(e)$ 才会增加，且绝不会超过 $u(e)$。

断言*2.* 对任意元素 $e \in B_0$ 和任意基 $B_i \in \{B_1, \ldots, B_k\}$，要么 $e \in B_i$，要么$e \in \operatorname{span}(B_i \setminus C)$。

固定任意元素 $e \in B_0$ 和来自 $B_1, \ldots, B_k$ 的一个基 B_i。若 $e \notin B_i$，则它曾被一个满足 $\ell(d) = 1$ 的元素 d 交换出去，其中 $B_{i,\geq 1} = B_{i,>0}$ 张成 e。也就是说，若 $e \notin B_i$，则 $e \in \operatorname{span}(B_{i,>0})$。由于 $C \subseteq \mathrm{N}_0$, ，我们有 $B_{i,>0} \subseteq B_i \setminus C$，因此 $e \in \operatorname{span}(B_i \setminus C)$。

现在，令 y 为 x 和 u 的逐点最小值；在满足 $y \leq u$ 的条件下， y 是 $kP_{\mathcal{I}}$ 中的最大值。回想一下，可以通过贪心算法按费用非递减顺序添加可行元素，从而得到一个最小费用基。将元素 $\mathrm{N} = \{e_1, e_2, \ldots\}$ 按费用非递减顺序编号，并在费用相同时进行排序，使得贪心算法按此顺序处理元素时得到 B_0。在 Cunningham 贪心增广算法中，通过依次处理各个 e_i，并将设置 $z(e_i)$ 为满足 $y + z \in kP_{\mathcal{I}}$ 条件下的最大值，得到一个最小费用强化 z 。（这等于满足条件 $(y + z)/k$ 在独立集多面体$P_{\mathcal{I}}$中张成 e_i 时的最小值；在 Cunningham 算法中，通过二分搜索并针对每次探测调用一次 k 重拟阵并来确定该值。）Cunningham 已证明该算法能产生一个最优解。因此，只需证明以下断言。

断言*3. Cunningham* 的贪心增广对所有 $e \in \mathcal{N}$ 选择 $z(e) = \max\{x(e) - u(e), 0\}$。

我们按照贪心顺序分析每个元素。考虑第 i 次迭代（其中 $i \in [n]$） ，在这一步中，贪心算法处理 e_i。根据归纳法，我们假设 $(y + z)/k$ 张成 $\{e_1, \ldots, e_{i-1}\}$（在$P_{\mathcal{I}}$ 中）。（基例 $i = 1$ 显然成立。）如果 $e_i \notin B_0$，那么 $e_i \in$ 张成$(\{e_j \in B_0 : j < i\})$，所以 e_i 由 $(y + z)/k$张成。

现在假设 $e_i \in B_0$。我们断言，贪心增广算法将 $z(e_i) = x(e_i) - u(e_i)$ 设为相应取值。为此，令 z' 为从 z 得到的向量，并将 $z'(e_i) = x(e_i) - u(e_i)$ 设为相应取值。我们知道 $y + z'$ 是可行的，因为 $y + z' \leq x$ 且 $x \in kP_{\mathcal{I}}$。为了证明 $(y + z')/k$ 张成 e_i，我们首先注意到，可以将这些 k 个独立集装入 $y + z'$

$$B'_j \stackrel{\text{def}}{=} B_j \setminus (C - e_i) \text{ for } j = 1, \ldots, k.$$

我们断言，对于每个 B'_j，均张成 e_i，这表明 j $(y + z')/k$ 张成 e_i。我们分两种情况讨论。第一种情况，若 $e_i \in B'_j$，则显然 $e_i \in$ 张成$\left(B'_j\right)$。第二种情况，若 $e_i \notin B'_j$，则 $e_i \notin B_j$。根据断言 2， $e_i \in$ 张成$(B_j \setminus C)$。由于 $B'_j \subseteq B_j \setminus C$， $e_i \in B'_j$。

这表明 Cunningham 贪心增广算法需要 $z(e_i) = x(e_i) - u(e_i)$。这就确立了该断言，并完成了证明。□

7 一般容量问题的近似算法

本节针对一般容量的拟阵问题开发快速近似算法。本节中的所有算法都基于使用随机舍入，以高概率将具有实值容量和实值参数 k 的问题，归约为具有整数容量和数量级为 $\ln(n)/\epsilon^2$ 的整数参数 k 的问题。随后，我们应用前几节中开发的近似推送-重标记算法；这些算法尤其适合该归约后的情形。

这种便利带来的代价之一是，我们将不再获得原始输入对应的原始解。不过，我们仍然能够获得对偶解，它们至少可以为相应判定问题的一方提供证书。

7.1 实值容量的随机舍入

设 M = , I) 是一个具有 n 个顶点、秩为 r 的拟阵，并设 u: N $\to$ R$_{>0}$ 是一组容量。设 $k > 0$ 是由上下文指定的参数。另设 $\tau > 0$ 是一个给定参数，且对于足够小的常数 c，有 $\tau \le ck\epsilon^2/\ln(n)$。根据需要减小 c ，我们可以不失一般性地假设 k/τ 是整数。设 $\tilde{u} \in$Z$^{\mathcal{N}}_{\ge 0}$ 是通过将 u/τ 随机舍入为整数向量而得到的一组随机化整数容量。也就是说，对于每个元素 $e \in$ N，我们独立地设置

$$\tilde{u}(e) = \begin{cases} \lceil u(e)/\tau \rceil & \text{with probability } u(e)/\tau - \lfloor u(e)/\tau \rfloor, \\ \lfloor u(e)/\tau \rfloor & \text{with (remaining) probability } \lceil u(e)/\tau \rceil - u(e)/\tau. \end{cases}$$

缩放后的容量拟阵 (M u) 具有一些显而易见的优良性质。首先，对于所有 $e \in$ N，都有 $\mathbf{E}[\tilde{u}(e)] = u(e)/\tau$。根据期望的线性性，对于所有集合 S，也有 $\mathbf{E}[\tilde{u}(S)] = u(S)/\tau$~。其次，(M $\tilde{u}$) 的容量为整数，且其容量（在期望意义下）比 (M u) 小 $O(k)$ 倍。我们希望将这种随机舍入应用于这样的问题：最大化打包若干 k 独立集所覆盖的总容量、打包 k 基，或由 k 基进行覆盖。这些问题特有的 k 值用于确定随机舍入到 $\tilde{u}$时的 k 值。因此，对于 (M $\tilde{u}$) 上相应的缩放问题， k/τ 为整数会很有帮助。此外，对于所有这些问题， k 都是容量的自然上界或近似上界，因此 $\tilde{u}$ 的容量相对较小，期望上界为 $O(\ln(n)/\epsilon^2)$（且以高概率成立）。

我们希望证明，对于这些问题，(M $\tilde{u}$)（高概率下）是 (M u) 的良好代表性样本。现在，尽管 $\tilde{u}$ 反映了 u/τ 的期望，但一般而言， $e \in$ N 的值 $\tilde{u}(e)$ 和 $S \subseteq$ N 的值 $\tilde{u}(S)$ 数量过多，无法假定它们在期望意义上都集中于各自的期望值。（事实上，当 $u(S)$ 显著小于 τ 时，值 $\tilde{u}(S)$ 永远不可能在乘法意义上接近期望值。）尽管如此，我们仍有以下定理，它利用这些问题的对偶刻画，证明 (M $\tilde{u}$) 在高概率下是 (M u) 的良好（针对具体问题的）表示。

定理 7.1。 根据上述设定，以下结论均以高概率成立。

(i) 令 M 表示 $(\mathcal{M}\ u)$ 中任意由 k 个独立~集构成的分数打包的最大总容量，令 M 表示 $(\mathcal{M}\ \tilde{u})$ 中任意由 k/τ 个独立集构成的分数打包的最大总容量，则有 $\left|\tilde{M} - M/\tau\right| \le \epsilon M/\tau.$

(ii) 若$(\mathcal{M}\ u)$ 可以分数打包 k 个基，则$(\mathcal{M}\ \tilde{u})$ 可以打包至少$(1-\epsilon)k/\tau$个基。

(iii) 若$(\mathcal{M}\ u)$ 不能分数打包 k 个基，则$(\mathcal{M}\ \tilde{u})$ 不能打包$(1+\epsilon)k/\tau$个基。

(iv) 若$(\ \mathcal{M}\ u)$ 可以由 k 个基覆盖，则$(\mathcal{M}\ \tilde{u})$ 可以由$(1+\epsilon)k/\tau$个基覆盖 .

(v) 如果 $(\mathcal{M}\ u)$ 不能由 k 个基覆盖，则$(\mathcal{M}\ \tilde{u})$ 不能由$(1-\epsilon)k/\tau$个基覆盖 *s.*

证明。 回忆一下，如果 $S \subseteq$N 是闭合的 , 则 $S =$ 张成(S)。我们断言，以高概率 有

$$|u(S) - \tau\tilde{u}(S)| \le \frac{\epsilon}{2}(u(S) + k\,\mathrm{rank}(S)) \text{ and } |u(\bar{S}) - \tau\tilde{u}(\bar{S})| \le \frac{\epsilon}{2}(u(\bar{S}) + k\,\mathrm{rank}(S)) \qquad (6)$$

对所有闭集 S （同时成立）。

¯为简化记号，若 $\tilde{u}(S)$ 或 $\tilde{u}(S)$ 违反上述不等式，就称闭集 S 不良。我们希望证明，高概率下不存在不良闭集。

首先，固定一个秩为 $=q$ 的闭集 S。考虑我们要证明的不等式中第一个（最左边的）不等式。根据标准的切尔诺夫不等式，有

$$\mathbf{P}\left[|\tilde{u}(S)-u(S)/\tau|>\frac{\epsilon}{2}(u(S)/\tau+kq/\tau)\right]\leq 2e^{-\epsilon^2 kq/4\tau}=2n^{-q/4c}.$$

同理，第二个不等式（针对 $\tilde{u}(\bar{S})$）的出错概率至多为 $2n^{-q/4c}$。取并集界，

$$\mathbf{P}[S \text{ is bad}]\leq 4n^{-q/4c}\leq n^{-c_1 q}$$

对于充分大的常数 c_1，有[3]

现在，固定 q。每个闭集 S 都由 S 的某个基定义，而该基包含 q 个元素。因此，秩为 q 的闭集至多有 n^q 个。对所有秩为 q 的集合 S 取并集界，

$$\mathbf{P}[\text{any closed set of rank } q \text{ is bad}]\leq 2n^q n^{-c_1 q}\leq n^{-c_2 q}$$

对于充分大的常数 c_2，

最后，对所有秩 $q\in[r]$ 取并集界，得到

$$\mathbf{P}[\text{any closed set is bad}]\leq\sum_{q=1}^{r}n^{-c_2 q}\leq n^{-c_3 q}$$

对一个足够大的常数 c_3。这就证明了该断言。

在证明的其余部分，我们假设不等式 (6对所有闭合的 S都成立的高概率事件。我们将使用这些不等式来证明结果 (i)–(v)。

¯先考虑结果 (i)。回忆一下， M等于所有集合 $u(S)+k\operatorname{rank}(S)$ 中的最小值；类似地， M'也是如此，只不过是相对于 $\tilde{u}$和 k/τ而言。由于将 S替换为其闭包 $\operatorname{span}(S)$只会使该量减小，因此只需考虑闭集。

对于所有闭集 S

$$\begin{aligned}\tilde{u}(\bar{S})+\frac{k}{\tau}\operatorname{rank}(S)&\geq\frac{1-\epsilon}{\tau}u(\bar{S})-\frac{\epsilon k}{\tau}\operatorname{rank}(S)+\frac{k}{\tau}\operatorname{rank}(S)\\&=\frac{1-\epsilon}{\tau}(u(\bar{S})+k\operatorname{rank}(S))\geq\frac{(1-\epsilon)M}{\tau}\end{aligned}$$

对于所有闭集 S。因此 $M'\geq(1-\epsilon)M/\tau$ 以高概率成立。

¯接下来，我们给 M' 一个上界。存在一个集合 S，它是一个闭集，使得 $M=u(S+k\operatorname{rank}(S))$。我们有

$$\begin{aligned}M'&\leq\tilde{u}(\bar{S})+\frac{k}{\tau}\operatorname{rank}(S)\leq\frac{1+\epsilon}{\tau}u(\bar{S})+\frac{\epsilon k}{\tau}\operatorname{rank}(S)+\frac{k}{\tau}\operatorname{rank}(S)\\&=\frac{1+\epsilon}{\tau}(u(\bar{S})+k\operatorname{rank}(S))\leq\frac{(1+\epsilon)M}{\tau},\end{aligned}$$

如所需。这证明了结果(i)。

现在考虑结果(ii) 和(iii)。根据拟阵基打包定理，$(\mathcal{M}\ u)$ 的打包数至少为 k 当且仅当对所有集合 $S\subseteq\mathcal{N}$,

$$u(\bar{S})\geq k(r-\operatorname{rank}(S)).$$

[3]根据 w 这意味着，通过使 c 足够小，可以将 c_1 设为任意大的常数 l.

因此，如果该不等式对所有 S 都成立，并且对于某个秩为 $< r$ 的集合 S（其中 $\mathrm{rank}(S)$）取等号，则打包数恰好为 k。

现在考虑 $(\mathcal{M}\ \tilde{u})$。根据定理 7.1，以高概率有

$$\begin{aligned}\tilde{u}(\bar{S}) &\geq \frac{1-\epsilon/2}{\tau}u(S) - \frac{\epsilon k}{2\tau}\operatorname{rank}(S) \geq \frac{(1-\epsilon/2)k}{\tau}(r-\operatorname{rank}(S)) - \frac{\epsilon k}{2\tau}\operatorname{rank}(S)\\ &\geq \frac{(1-\epsilon)k}{\tau}(r-\operatorname{rank}(S)),\end{aligned}$$

因此，打包数至少为 $(1-\epsilon)k/\tau$。这就证明了结果 (ii)。

对于结果 (iii)的反向论证，我们知道存在一个闭集 S，满足 秩$(S) < r$ 且 $u(S) = k(r - $秩$(S))$。注意到 $u(S) \geq k$。根据切尔诺夫不等式，有

$$\mathbf{P}[\tilde{u}(\bar{S}) \geq (1+\epsilon)u(\bar{S})/\tau] \leq e^{-\epsilon^2 u(\bar{S})/3\tau} \leq e^{-\epsilon^2 k/3\tau} = n^{-1/3c}.$$

（我们指出，$1/3c$ 表示一个任意大的常数。）因此，以高概率有

$$\tilde{u}(\bar{S}) \leq (1+\epsilon)u(\bar{S})/\tau = \frac{(1+\epsilon)k}{\tau}(r-\operatorname{rank}(S))$$

因此，$(\mathcal{M}\ \tilde{u})$ 的打包数至多为 $(1+\epsilon)k/\tau$。这就证明了结果 (iii)。

最后，我们证明结果 (iv) 和 (v)。回顾一下，$(\mathcal{M}\ u)$ 可以由 k 个基分数覆盖，当且仅当对所有集合 $S \subseteq \mathcal{N}$，$u(S) \leq k$ 秩(S)。由于容量非负，只需验证对所有闭集都成立的不等式 $u(S) \leq k$秩(S)。

假设 $(\mathbf{M}\ u)$ 可由 k 个基分数覆盖。我们有

$$\tilde{u}(S) \leq \left(1+\frac{\epsilon}{2}\right)\frac{u(S)}{\tau} + \frac{\epsilon k \operatorname{rank}(S)}{2\tau} \leq \frac{(1+\epsilon)k}{\tau}\operatorname{rank}(S)$$

对所有闭集 S。因此$(\mathcal{M}\ \tilde{u})$ 可由$(1+\epsilon)k/\tau$ 个基分数覆盖。这证明了结果(iv)。

对于结果(v)，假设$(\mathcal{M}\ u)$ 不能由少于 k 个基分数覆盖。那么存在一个闭集 S 使得 $u(S) \geq k$的秩为S。我们有

$$\tilde{u}(S) \geq \frac{1-\epsilon/2}{\tau}u(S) - \frac{\epsilon k}{2\tau}\operatorname{rank}(S) \geq \frac{(1-\epsilon)k}{\tau}u(S).$$

因此$(\mathcal{M}\ \tilde{u})$ 不能由$(1-\epsilon)k/\tau$ 个基分数覆盖。这证明了结果(v) 并完成了证明。

□

备注 7.1。如上所述，针对基打包问题的近似，[Kar98]已经给出了一个引理，使我们能够将实值容量化简为较小的整数容量。[Kar98] 中的构造略有不同；在 [Kar98], 中，每个随机容量 $\tilde{u}(e)$都独立地从均值为 $p \cdot u(e)$的泊松分布中采样，其中参数 $p > 0$ 满足 $p \geq c\epsilon^2/k\ln(n)$，且常数 c足够大。总体而言，其净效果与我们在此分析的 $\tilde{u}$ 舍入构造相同。尽管与 [Kar98] 有所重叠，我们仍纳入结果（ii）和（iii）的证明，因为我们认为它们有以下几点值得关注。首先，此处的证明技巧与结果（i）、（iv）和（v）中其他拟阵问题的证明技巧是统一的。（反过来说，[Kar98] 中的证明技巧似乎不太适用于这些其他问题。）其次，此处的证明不同于 [Kar98] 中的证明，而且可以说更简单，因为它不依赖随机收缩算法。

7.2 独立集和森林的最大容量打包

定理 7.2。对于实值容量，可以高概率地在 $O(n+$(最优值 $/k)\log(n)\log\left(r\log(n)/\epsilon^2\right)/\epsilon^3)$ 随机化时间内，计算实值容量下最大（分数） k重拟阵并的值的$(1-\epsilon)$-近似，以及一个$(1+\epsilon)$-近似最小对偶解。（注意，最优值 $/k \le r$。）

证明。 我们应用定理 7.1 将问题规约为整数容量和最优值(最优值 $/k)\log(n)/\epsilon^2$，且以高概率成立。然后，我们应用定理 $(1-\epsilon)$中针对整数容量的近似算法3.7。运行时间由定理 3.7得出。

□

对图拟阵进行相同的规约，可得如下结论。

定理 7.3。 在边容量为实数的无向图中，可以在 $O\left(m\alpha(n)+n\log^2(n)(\log(n)+\log(k)\alpha(n))/\epsilon^3\right)$ 随机化时间内，计算出由 k 个森林的分数打包所能覆盖的最大容量 最优值 的$(1\pm\epsilon)$-近似值。（注意，最优值 $/k \le n-1$。）

7.3 拟阵基打包、拟阵成员判定与网络强度 h

定理 7.4. 对于实值容量，以高概率可在随机化时间 $O\left(n+r\log^2(n)(\log\log(n)+\log(1/\epsilon))/\epsilon^3\right)$ 次独立性查询内，计算出一个$(1\pm\epsilon)$近似，用于判定拟阵强度是大于还是小于 k 。

证明. 根据定理 7.1 或采用 [Kar98], 中的技术，我们可以将问题约化为以高概率将k/τ个基打包进整数容量。然后，我们应用定理 4.2给出的、以整数容量打包 $\ln(n)/\epsilon^2)$ 个基的$(1\pm\epsilon)$近似算法。运行时间由定理 4.2得出。

□

用于判定拟阵强度的近似算法可通过二分搜索扩展为近似拟阵强度的近似算法。这里我们介绍一种改进算法，通过谨慎调整误差参数来减少标准的对数开销。

定理 7.5。 对于介于1 和 U 之间的实值容量，可以在时间上界为 $O((n+r\log(n)\log(r))\log(nU/r)+r\ln(n)\ln(r/\epsilon)/\epsilon^3)$ 次独立性查询内，计算出拟阵强度的 $(1\pm\epsilon)$- 近似值。

证明。 一开始，我们知道拟阵强度介于 1 和 nU/r 之间。

现在，对于 $i\in\mathbb{Z}_{\ge 0}$，令 $\epsilon_i=2^{-i}$。结合深度为 $O(\log(nU/r))$ 的二分搜索和定理 7.4 中的近似判定算法，并令误差参数为比 ϵ_0 小一个常数因子的值，即可高概率地获得强度的 $(1+\epsilon_0)$-近似值。对于 $i\ge 1$，给定强度的 $(1+\epsilon_{i-1})$-近似值后，我们可以通过常数深度的二分搜索计算出 $(1+\epsilon_i)$-近似值；每次探测都会调用定理 7.4，并使用 $\Omega(\epsilon_i)$ 作为误差参数。最终，我们得到 $(1+\epsilon_i)$-近似值，其中 $\epsilon_i=2^{-i}$ 不超过输入误差参数 ϵ，如所需。

现在我们来界定运行时间。第一组 $\log(n)$ 次调用定理 7.4 所需的时间，其中误差参数为常数

$$O(n\log(nU/r)+r\log(n)\log(r)\log(nU/r))\text{-query} \tag{7}$$

时间。此后，对于每个 ϵ_i，我们都要调用定理 7.4 一次， ϵ_i 范围介于 1 和$\epsilon/2$之间。对于首项 $O(n)$，这些调用的总开销为 $O(n\log(1/\epsilon))$，其量级被以下项所支配

上文中的 $O(n\log(n))$。对于形式为 $r\ln(n)\ln(r/\epsilon)/\epsilon^3$ 的第二项，对所有 ϵ_i 求和后，其量级由最小的 ϵ_i 所支配，而该值为 $\Omega(\epsilon)$，给定

$$O\Big(r\ln(n)\ln(r/\epsilon)/\epsilon^3\Big) \tag{8}$$

wo总计 rk。将等式 (7) 和 (8) 相加，即得所述运行时间。 □

7.4 拟阵基覆盖、拟阵成员判定与拟阵的拟树性 y

分数基覆盖可以表述为一个判定问题：给定一个容量拟阵和一个附加参数 k，目标是判定这些容量能否由 k 个基分数覆盖。$k=1$ 的重要特例等价于检验一个分数点 $x\in \mathrm{R}^{\mathcal{N}}_{\geq 0}$ 是否位于独立集多面体中。这个问题称为拟阵成员判定。

在判定分数基覆盖时，不失一般性，我们可以假设 $k=1$。对于固定的误差参数 $\epsilon\in(0,1)$，拟阵成员资格问题的 $(1\pm\epsilon)$-近似是指给出正确输出，判定 (a) 该拟阵能否由 $1+\epsilon$ 个基覆盖，或 (b) 该拟阵能否由 $1-\epsilon$ 个基覆盖。请注意，当分数覆盖数介于 $1-\epsilon$ 和 $1+\epsilon$ 之间时，任一选项都可以接受。我们得到近似拟阵成员判定的如下运行时间。

定理 7.6。 以高概率，在 $O(n+r\ln(n)\ln(r/\epsilon)/\epsilon^3)$ 随机化时间内，可以计算拟阵成员资格问题的 $(1\pm\epsilon)$近似。

证明概要。 根据定理 7.1，我们可以将 $(1\pm\epsilon)$ 拟阵成员判定（容量为实数）归约为 $(1+\epsilon)$-近似整数基覆盖，使用 $k=O(\ln(n)/\epsilon^2)$ 个基。运行时间由定理 4.4得出。

□

对于图拟阵，请回想分数覆盖数称为森林度。以下结果与上面的拟阵成员判定定理相对应，只是针对图拟阵。归约过程相同，不过现在我们应用图拟阵对应的算法。

定理 7.7。 以高概率，在 $O(m\alpha(n)+n\ln(n)(\log(n)+\log(\log(n)/\epsilon)\alpha(n))/\epsilon^3)$ 随机化时间内，可以计算一个$(1\pm\epsilon)$近似，用于判定一个点是否属于森林多面体。

还可能希望求出最大值 k，使得一个容量拟阵。以下将定理 7.6 作为黑盒使用，并且比直接套用简单的二分搜索所得结果略好。

定理 7.8。 对于实值容量，可以以高概率在运行时间不超过 $O((n+r\ln(r))\ln(n)+r\ln(n)\ln(r/\epsilon)/\epsilon^3)$ 次独立性查询的条件下，计算出拟阵可被 k 个分数基覆盖所需最小 k 的 $(1\pm\epsilon)$-近似。

证明。 一开始，我们知道森林度介于任一元素的最大容量与所有元素容量之和之间，二者相差至多 n 倍。

与定理 7.5类似，对于 $i\in\mathbb{Z}_{\geq 0}$，设 $\epsilon_i=2^{-i}$。通过结合深度为 $O(\log(n))$ 的二分搜索与定理 7.6 中的近似判定算法（误差参数为常数），可以高概率获得强度的 $(1+\epsilon_0)$- 近似。对于 $i\geq 1$，给定强度的 $(1+\epsilon_{i-1})$-近似，我们可以通过常数深度的二分搜索计算出一个 $(1+\epsilon_i)$-近似。每次探测都会调用定理 7.6 ，其误差参数为 $\Omega(\epsilon_i)$。

最终，我们得到一个（$1+\epsilon_i$）-近似，其中 $\epsilon_i = 2^{-i}$ 不超过输入误差参数ϵ，正如所需。

现在我们来界定运行时间。第一组 $\log(n)$ 次定理 7.6 调用，误差参数为常数，耗时

$$O(n\log(n) + r\log(n)\log(r)) \tag{9}$$

时间。此后，对于每个介于 ϵ_i 和 $\epsilon/2$ 之间的 ϵ_i，我们会调用定理 7.6 常数次。对于首项 $O(n)$，所有这些调用的总工作量为 $O(n\log(1/\epsilon))$，被上面的 $O(n\log(n))$ 所支配。对于形如 $r\ln(n)\ln(r/\epsilon)/\epsilon^3$ 的第二项，对所有 ϵ_i 求和后，结果由最小的 ϵ_i 所支配，而该值为 $\Omega(\epsilon)$，因此

$$O\Big(r\ln(n)\ln(r/\epsilon)/\epsilon^3\Big) \tag{10}$$

总工作量。将公式 (9) 和 (10) 相加，即得所述运行时间。 □

将同样的修改版二分搜索应用于图拟阵，可得以下用于估计图的森林度的随机化算法。

定理 7.9。 以高概率，可以在 $O(m\log(n)\alpha(n) + n\log(n)(\log(n) + (\log\log(n) + \log(1/\epsilon))\alpha(n))/\epsilon^3)$ 随机化时间内，计算出图强度的一个 $(1\pm\epsilon)$ 近似。

8 通过增广实现更快的精确算法

在本节最后一部分中，我们描述了一个用于 k 重拟阵并的增广路子程序，并在 $r \geq k^{1+o(1)}$ 时利用它获得更快的精确算法。

引理 8.1。 由 $(1-\epsilon)$-近似 k 重拟阵并数据结构生成的 k 个独立集的一个打包，可以在每增加一个元素时至多进行$O(\min\{n + \text{最优值}\log(r)\ rk\log(kr)\})$ 次查询的时间内扩展为一个最优解。

将引理 8.1 与 $(1-\epsilon)$-近似算法结合起来，并适当选择 ϵ，可得到如下运行时间；当 $r \geq k^{1+o(1)}$ 时，该运行时间更快。特别地，在 最优值 $\leq kr \leq n$ 的无权情形下，我们得到 $\tilde{O}\Big(n^{3/2}\Big)$次独立性查询的次二次上界。

定理 8.2。 k 个独立集的最大容量打包可在至多进行 $O\Big(n + \text{最优值}\ \sqrt{n'\log(kr)}\Big)$ 次独立性查询的时间内计算得到，其中 $n' = \min\{n + \text{最优值}\log(r)\ rk\log(kr)\}$。

证明。 令 $\epsilon > 0$ 为待确定的参数。可以通过 $O(n + \text{最优值}\ \log(kr)/\epsilon)$ 次独立性查询，计算出一个 $(1-\epsilon)$ 最优值容量打包。每次增广可在 $O(n' + \text{最优值}\ \log(r))$ 时间内将其增广为最优解。因此，总运行时间为

$$O(n + \mathrm{OPT}\log(kr)/\epsilon + \epsilon\,\mathrm{OPT}(n' + \mathrm{OPT}\log(r))).$$

取 $\epsilon = \sqrt{(n' + \text{最优值}\ \log(r))/\log(kr)}$，即可平衡最后两项，从而得到所述运行时间。这里需要指出，通过使用常数 ϵ 运行近似拟阵并算法，可以得到最优值的常数因子近似；这足以在常数因子的范围内平衡各项。

□

还需证明引理8.1。

8.1 从拟阵推重标算法初始化

我们需要用以下 k 独立集 $I_1,\dots,I_k$来初始化算法，而第k节中的拟阵并算法会直接生成3 k 基 $B_1,\dots,B_k$。我们可以通过从某些基中移除超额元素，直到不再有超额元素，从而轻松获得总容量相同的打包 $I_1,\dots,I_k$。然而，为了保留 $B_1,\dots,B_k$的某些有用结构，我们按照以下贪心方式，谨慎地从基中移除超额元素 $B_1,\dots,B_k$。

初始时，我们设置 $I_1 = B_1,\dots,I_k = B_k$。只要存在一个超额元素 e（相对于 $I_1,\dots,I_k$），我们就从索引最大的独立集 I_j中移除 e。容易看出，算法终止时不存在超额元素，同时目标值保持不变。以这种方式移除超额元素，还能获得以下关键性质，现定义如下。

定义 8.3。设 $I_1,\dots,I_k$ 是 k 个独立集的一个打包。若对于所有未覆盖元素 e 及所有独立集 I_i，都有 $e \in I_i$，则称 $I_1,\dots,I_k$ 是极大的。若 $I_1,\dots,I_k$ 的张成按递减顺序排列，则称它们按递减顺序排列；也就是说，对于 $i = 1,\dots,k-1$，有 $I_{i+1} \subseteq$ 张成(I_i)。

引理 8.4. 设 $I_1,\dots,I_k$ 是按上述贪心方式，由拟阵并推重标算法输出的基 $B_1,\dots,B_k$ 得到的 k 独立集的一个打包。则：

(a) $I_1,\dots,I_k$ 是一个极大打包。

(b) $I_1,\dots,I_k$ 按递减顺序排列。

证明。 固定拟阵推送-重标记算法结束时的配置 $B_1,\dots,B_k$ 和 $\ell\colon \mathbf{N} \to \mathbf{Z}_{\geq 0}$。

我们首先断言，任意超额装填的元素 e 都有 $\ell(e) = 0$。事实上，最初所有元素都有 $\ell(e) = 0$，而只有在元素未被覆盖时才会对其重新标记。同时，元素只能因初始配置而超额装填，因此，终止时仍处于超额装填状态的元素从一开始就处于超额装填状态，且始终未曾被覆盖。

现在，每个 I_i 都将包含来自 B_i 且未超额装填的所有元素。由于超额装填的元素层级为 0，因此有 $B_{i,\geq 1} \subseteq I_i \subseteq B_i$，对所有 i 均成立。同时，任何未覆盖元素的层级都是 > 1。根据不变量 (II)，对于所有未覆盖元素 e，我们有

$$e \in \mathcal{N}_{>1} \subseteq \operatorname{span}(B_{i,\geq 1}) \subseteq \operatorname{span}(I_i)$$

这就证明了性质 (a)。

接下来我们证明性质 (b)，即独立集按递减顺序排列。对于每个独立集 I_i，我们可以将 I_i 表示为 $B_{i,\geq 1}$ 和 $I_{i,0} \overset{\text{定义}}{=} I_i \cap B_{i,0}$的不交并。对于每个索引 $i \in [k-1]$，根据基的递减顺序（见定义 3.4），我们有 $B_{i+1,\geq 1} \subseteq$ 张成$(B_{i,\geq 1})$，并且 $I_{i+1,0} \subseteq I_{i,0}$，因为超额元素会从索引最大的独立集中移除。因此

$$I_{i+1} = B_{i+1,\geq 1} \cup I_{i+1,0} \subseteq \operatorname{span}(B_{i,\geq 1}) \cup I_{i,0} \subseteq \operatorname{span}(I_i),$$

如所需。 □

8.2 贪心稀疏化

在继续介绍增广路径算法之前，我们指出，通过[Kar98], 的技术，可以假设 $n \leq O(kr\ln(r))$。

引理 8.5. 设 $I_1, \ldots, I_\ell$ 是 $\ell \geq k(1+\ln(r))$ 集合的一个极大打包。那么，在由 $I_1, \ldots, I_\ell$ 诱导的（较小的）容量拟阵中，最大 k 重的拟阵并的大小与输入拟阵中的相同。

证明概要. 证明本质上与 [Kar98] 相同，不过后者关注的是基打包。证明中，我们将基打包的对偶刻画 [Edm65a] 替换为拟阵并的对偶刻画 [Nas67]。

□

[Kar98] 介绍了如何通过贪心方式，使用 $O(n + kr\ln(r))$ 次独立性查询构造这样的打包。这里也可以沿用相同的构造方法，但我们从推送-重标记算法给出的极大打包 $I_1, \ldots, I_k$开始，然后贪心地扩展。因此得到以下结果。

引理 8.6. 增加 $O(\min\{n, kr\ln(r)\})$ 次独立性查询的运行时间开销后，我们可以假设 $n \leq O(kr\ln(r))$。

8.3 构建辅助图

从整体上看，我们有一组独立集的打包 I_1，……，I_k，当前的直接目标是增加它们的总大小。然而，事情并不像找到一个未覆盖元素 e 并将其加入集合 I_k那么简单——所有未覆盖元素都被所有独立集张成，因为 I_1，……，I_k是极大的。要增加 I_1，……，I_k的总大小，我们可能必须在 I_1，……，I_k中移动许多元素，为另一个元素腾出空间。

众所周知，可以通过在有向辅助图中搜索路径来找到这样的增广，其中每条弧都编码了一种局部交换，例如在独立集中用一个元素替换另一个元素 I_i，或将一个元素从一个 I_i移到另一个集合。这里我们介绍标准的辅助图 [GM75；Knu74]。[4] 我们有两个辅助顶点 s 和 t，它们将分别作为搜索的起点和终点。对于每个元素 e，我们有 k 个辅助顶点 $e^{(1)}$，……，$e^{(k)}$。每个 $e^{(i)}$ 表示 e 相对于 B_i 的情形，具体含义将由我们下面描述的弧加以说明。我们有四种类型的弧。

1. $(s, e^{(i)})$，其中 e 未覆盖且 $e^{(i)} \notin B_i$。该弧表示尝试将 $e^{(i)}$ 交换进B_i。2. $(e^{(i)}, d^{(i)})$，其中 $d \in B_i$、$e \notin B_i$，且 $B_i - d + e$是独立的。该弧表示在 B_i中用e 交换 d 。3. $(d^{(i)}, d^{(j)})$，其中 $d \in B_i$、$d \in$ 张成$(B_j) \setminus B_j$，且 d已覆盖。该弧表示从 B_i 中移除 d，并利用释放出的容量启动将 d 交换进 B_j。4. $(e^{(i)}, t)$，其中 e 不被 B_i张成。该弧表示将 e 插入 B_i。

总计而言，该辅助图有 $O(nk)$ 个顶点，至多有 $O(nrk)$ 条弧。

从 s 到 t 的路径在此图中具有非常特定的结构。在 s 和 t之间，路径由形如 $e^{(i)}$ 的辅助顶点组成，这些顶点交替满足 $e \notin B_i$ 和 $e \in B_i$。这类内部顶点的数量始终为奇数，且前一种类型比后一种多一个。就弧而言，第一条弧属于类型1，最后一条弧属于类型4，中间的弧则在以下类型之间交替：

[4]该辅助图　　也可以通过拟阵交的视角来理解，但需要进行一些修改。

类型2 和类型3，从类型2 开始，到类型2结束。综合起来，辅助图中的一条$(s,\ t)$路径对应于一个交换序列，外加一次最终插入，其净效果是使 I_1、……、 I_k的总大小增加1。如果这条路径保持可行性，我们就称之为增广路径。也就是说，集合 I_1、……、 I_k保持独立，并且不会过度装入元素。

$(s,\ t)$-路径不一定是增广路径。一方面，很容易看出这些操作不会违反任何容量约束，因为每当将一个已覆盖元素插入某个集合时，都会先从另一个集合中移除同一元素。对于 I_1，……， I_k 的可行性而言，尽管每次交换（由类型为 2 的弧编码）单独来看都能保持各集合的独立性，但所有交换合在一起却未必能保持独立性。不过，以下准则给出了条件，在这些条件下，对同一独立集进行交换序列仍能保持独立性。该准则是标准做法，长期以来一直用于证明拟阵交和划分算法的正确性。（此处措辞引自 [Cun84]。）

事实 8.7。 给定一个拟阵 $\mathcal{M}\ =(\mathcal{N},\ \mathcal{I})$，令 $I\ \in\mathcal{I}$，并令 e_1、……、$\notin\ I$、 d_1、……、$d_p\ \in\ I$，以及可选的 $e_{p+1}\notin I$ 为互不相同的元素，使得：

(a) 对于 $i=0,\ldots,p,\ I-d_i+e_i\in\mathcal{I}$。
(b) 对于 $0\le j<i\le p,\ I-d_i+e_j\notin$
$\mathcal{I}$。*(c)* $I+e_{p+1}\in\mathcal{I}$ （包含 e_{p+1} 时）。

Then $I'=I-d_1-\cdots-d_p+e_1+\cdots+e_{p+1}\in\mathcal{I}.$

辅助图中一条 (s,t)-路径所对应的交换，只可能违反事实 (b) 中的三条性质之一，即性质 8.7。下面，我们设计一个子程序：给定辅助图中的一条(s,t)-路径，通过高效地识别并消除对性质(b)的违反，提取一条增广子路径，具体如下。

引理 8.8。 给定辅助图中一条长度为 ℓ 的(s,t)-路径，可以在 $O(\ell\log(r))$ 次独立性查询的运行时间上界内，计算出一条增广子路径。

证明。固定一个独立集 I_i。假设该路径表示依次插入 $e_1,\ldots,e_p\notin I_i$，分别交换出 $d_1,\ldots,$ $d_p\in I_i$。这意味着，针对 $j=1,\ldots,p$ 的辅助弧 $(e_j^{(i)},d_j^{(i)})$ 按该顺序出现在辅助路径中。若 $I_i-d_1-\cdots-d_p+e_1+\cdots+e_p\notin\mathrm{I}$，则根据事实 8.7，必定存在索引 $1\le j_1<j_2\le p$，使得 $I_i-d_{j_2}+e_{j_1}\in\mathrm{I}$。此交换意味着辅助图中存在一条弧 $\left(e_{j_1}^{(i)},d_{j_2}^{(i)}\right)$，因此，我们可以用 $\left(e_{j_1}^{(i)},d_{j_2}^{(i)}\right)$ 替换 $(e_{j_1}^{(i)},d_{j_1}^{(i)})$ 和 $(e_{j_2}^{(i)},d_{j_2}^{(i)})$ 之间（包括这两条弧在内）的所有弧，从而缩短我们的 (s,t)-路径。我们将这样的弦对 (e_{j_1},d_{j_2}) 称为弦，因为它相对于我们的 (s,t)-路径构成一条弦。我们的目标是反复识别弦对 (j_1,j_2) 并缩短路径，直到得到一个更短的、针对 I_i 可行的交换序列。

如上所述，一对索引（$j_1,\ j_2$）可以按以下方式高效识别。对于从 $j_1=p$ 到 1 的每个值，我们对第一个满足 j_2 使得 $I-d_1\cdots-d_{j_1}-d_{j_2}+e_{j_1}\in\mathrm{I}$ 的索引进行二分搜索。然后，j_2 是满足 d_{j_2} 属于 $I+e_{j_1}$的回路的第一个索引。现在，如果 $j_1=j_2$，那么这就验证了对于所有 $j<j_2,\ I-d_j+e_{j_2}\notin\mathrm{I}$，正如所需。否则，（$j_1,\ j_2$）表示一条弦，可用于缩短（$s,\ t$）-路径。我们在（$j_1,\ j_2$）处对路径进行捷径处理。随后减小 j_1，并继续搜索，对每个 e_{j_1} 进行捷径处理或验证，直到确认不再存在弦对。对于 I_i，剪除所有弦对的总运行时间上界为 $O(p\log(r))$，其中 p 指的是路径在剪除之前最初编码的交换数量。

上面的描述针对的是路径编码交换序列而非额外插入的情况。然而，该讨论可以直接扩展到路径编码一系列 p 次交换加一次额外插入的情况。同样，对于固定的独立集 I_i，我们可以剪除所有弦对，并保证剩余的交换与插入序列保持 I_i的独立性，其运行时间上界为 $O(p\log(r))$ 次独立性查询。

总的来说，该算法依次处理每个独立集 I_i。对于每个 I_i，我们剪除弦对并缩短 $(s,\ t)$-路径，使剩余的交换保持 I_i 的独立性。对所有集合 I_i 的总运行时间上界为 $O(\ell\log(r))$ 次独立性查询，因为 $O(\ell)$ 统计了所有独立集 I_i 中原始路径上的交换总数。

□

8.4 在辅助图中高效搜索 $(s,\ t)$-路径

我们现在已经构建了一个辅助图，并证明任意 $(s,\ t)$-路径都能高效地转换为增广路径。接下来需要设计一种高效方法来寻找 $(s,\ t)$-路径。当然，我们可以通过逐一检查上述每条弧是否存在来显式构建该图，但正如前面所述，整个图的规模超过了所需的运行时间。

回想一下，我们的目标是找到一条 $(s,\ t)$-路径，或者得出不存在此类路径的结论。特别是，我们并不一定要检查并遍历图中的所有弧。我们的目标是设计一种搜索算法，其运行时间所需的独立性查询次数与图中顶点总数成正比（至多相差对数因子）。

我们将说明如何为此目的修改广度优先搜索。[5] 回想一下，广度优先搜索会在首次访问顶点时将其标记，然后再将其加入待搜索元素队列。当然，这些标记记录了顶点已被访问，并可防止无限循环。在我们的隐式辅助图中，如果一条弧的头顶点已经标记，我们也可以跳过对该弧是否存在的检查。

为提高效率，我们将在已标记顶点集合上施加以下不变量。对于打包中的每个独立集 I_i，令 M_i 表示已标记的辅助顶点集合 $e^{(i)}$，其中$e \in I_i$。我们将严格遵守这一不变量：已标记集合M_1，……，M_k按递减顺序排列（其含义与 I_1，……，I_k相同，参见定义8.3）。为维持这一不变量，我们引入以下子程序。

前驱搜索。每当我们即将标记一个元素 $e^{(i)}$，其中 $e \in B_i$，就需要确保对于所有 $j < i$，$e \in$ 张成(M_j)。在这种情况下，我们会从 e_i 启动前驱搜索，简称预搜索，并执行以下步骤。

预搜索辅助元素 $e^{(i)}$：令 $e \in B_i$。

1. 当 $e \notin$张成(M_{i-1}) 时：
   A. 对于每个满足 $I_{i-1} - d + e \in$ I.B. 的元素 $d \in I_{i-1} \setminus$ 张成(M_j)，递归地预搜索 d_j。
   C. 标记 d_j，并记录弧 (e_i, d_j)，然后将 d_j 加入（外层的广度优先搜索）搜索队列。

上述步骤是在高层次上描述的，我们将在后文讨论具体细节和效率问题。首先，我们说明为什么上述（高层次的）步骤能够使 $M_1,\ldots,M_k$ 保持递减顺序。（之所以在分析具体实现细节之前先阐明这一点，是因为我们将利用 $M_1,\ldots,M_k$ 的递减顺序来提高实现效率。）

[5]也可以使用其他基于标记的搜索算法，例如深度优先搜索。

引理 8.9。 假设 I_1、...、I_k 和 M_1、...、M_k 均按递减顺序排列，并令 $e \in B_i$。那么，对 $e^{(i)}$ 进行预搜索会使 M_1、...、M_k 保持递减顺序。如果 $i > 1$，我们还得到 $M_i + e \subseteq$ 张成 (M_{i-1})。

证明。 我们对 i 使用归纳法来证明该断言。若 $i = 1$，则无需处理，该断言显然成立。现在，令 $i > 1$，并假设该断言对 $i - 1$ 成立。考虑对辅助顶点 $e^{(i)}$ 进行预搜索，其中 $e \in I_i$。回想一下，I_{i-1} 张成 e，因为 $I_1, \ldots, I_k$ 按递减顺序排列。因此，如果 M_{i-1} 不张成 e，则 $I_{i-1} + e$ 的回路中仍有其他元素 d 不属于 M_{i-1}。只要 M_{i-1} 不张成 e，预搜索子程序就会反复选取这样的元素 d，并在标记 $d^{(i-1)}$ 之前对 $d^{(i-1)}$ 调用预搜索。根据归纳假设，对 $d^{(i-1)}$ 的预搜索确保我们可以安全地标记 $d^{(i-1)}$，同时保持 $M_1, \ldots, M_k$ 按递减顺序排列。对 e 的预搜索结束时，$M_1, \ldots, M_k$ 仍按递减顺序排列，且 M_{i-1} 张成 e。 □

隐式搜索算法。 现在，我们将介绍一种将预搜索作为黑盒的搜索算法。这里的挑战在于，在不知道所有弧的显式信息的情况下搜索辅助图。从高层次来看，通过利用 I_1、I_k 和M_1、.....、M_k 的递减顺序，除对数因子外，我们能够将查询限制在那些会生成指向尚未访问顶点的弧的查询上。

搜索算法的行为取决于我们搜索的辅助顶点类型。这里有三种类型。第一种是源点 s。第二种是某个辅助顶点 $e^{(i)}$，其中 $e \in I_i$ 或 $i = k$。特别地，出于下文将进一步说明的效率原因，我们不会直接对满足 $e \notin I_i$ 和 $i < k$ 的辅助顶点 $e^{(i)}$ 调用搜索。

我们从 针对 s 的搜索例程开始，它会简单地搜索每个未覆盖元素 e .

在源点 s 处搜索：

1. 对于每个未覆盖元素 e，如果 $e^{(k)}$ 未被标记，则标记 $e^{(k)}$，记录弧 $(s, e^{(k)})$，并对 $e^{(k)}$ 调用搜索。

接下来，我们描述用于搜索形式为 $e^{(i)}$ 的元素的算法。

在辅助顶点 $e^{(i)}$ 处搜索：

1. 如果 $e \notin$ 张成(I_k)：

   (a) 令 j 为满足 $e \notin$ 张成(I_j) 的第一个索引。(b) 标记 $e^{(j)}$，标记 t，并记录弧 $(e^{(i)}, e^{(j)})$ 和 $(e^{(j)}, t)$。发出信号，表示我们已找到一条 (s, t)-路径。

2. 否则，当 $e \notin$ 张成(M_k) 时：

   (a) 令 $d \in I_k \setminus M_k$ 满足 $I - d + e \in$ I。(b) 对 $d^{(k)}$ 进行预搜索。(c) 标记 $e^{(k)}$ 和 $d^{(k)}$。记录弧 $(e^{(i)}, e^{(k)})$ 和 $(e^{(k)}, d^{(k)})$。将 $d^{(k)}$ 加入搜索队列。

引理 8.10。 搜索辅助顶点 $e^{(i)}$ 后，其中 $e \in I_i$，我们得到以下结论：

1. 如果存在一个索引 j，使得 $e \notin \mathrm{span}(I_j)$，那么对于第一个这样的索引 j，记录路径 $(e^{(i)}, e^{(j)}, t)$，并标记顶点 $e^{(j)}$ 和 t。*2.* 否则，对于所有索引 j，都有 $e \in \mathrm{span}(M_j)$ 。

证明。 第一种情况可直接从代码得出。第二种情况下，对于所有 i，都有 $e \in$span(I_i)。搜索仅在 $e \in$ span(M_k) 时于 e 处退出。由于 $M_1,\ldots,M_k$ 按递减顺序排列，因此对于所有 i，都有 $e \in$span(M_i)。 □

引理 8.11。 在搜索 $e^{(k)}$ 以寻找未覆盖元素 e之后，对于所有 i，我们有 $e \in$ 张成(M_i) 。

证明。 我们知道 ein 张成(I_k)，因为 e 未覆盖，且 $I_1,\ldots,I_k$ 是极大打包。只有当 $e \in$张成(M_k)时，搜索才会退出。由于 $M_1,\ldots,M_k$ 按递减顺序排列，因此对于所有 i，我们有$e \in$张成(M_i)。 □

引理 8.12。 如果存在从 s 到 t 的路径，搜索就会找到一条。

证明。 该算法记录一些弧，这些弧共同包含通往所有已标记顶点的路径。因此，该算法会标记 t 并指示已找到一条 (s,t)-路径，此时记录的弧中包含所需的弧。（父指针或对已记录弧进行图搜索，都能找到一条 (s,t)-路径。）

请注意，如果搜索没有指示找到一条 (s,t)-路径，它仍会标记所有可达的辅助顶点，但辅助顶点 $e^{(i)}$ 除外；在该顶点处， e 未覆盖且 $e \notin I_i$。然而，前一个引理保证了 $e \in$ 张成(M_i)，因此从 $e^{(i)}$可达的任何辅助顶点仍已被探索。因此，如果 t 可达，它就会被标记，并会找到一条 (s,t)-路径，从而达到预期目的。 □

搜索与预搜索的效率：

引理 8.13。 从 s 开始搜索所需的运行时间不超过 $O(n+$最优值log(r) $+\log(k))$ 次独立性查询。

证明。 令 ℓ 表示被标记的 $e^{(i)}$ 的辅助顶点总数。注意，$\ell \le O(n+$最优值)，因为如果我们标记一个辅助顶点 $e^{(i)}$，且它具有 $e \notin I_i$ 和 $i<k$，那么也存在一条出弧指向某个 $d^{(j)}$，其中 $d \in I_j$。对一个元素 e执行的每次搜索或预搜索例程，都包含常数次查询，外加针对每个被标记的辅助顶点的若干次查询，用于搜索被标记的元素。每次搜索所需的常数次查询总计至多为 $O(n+\ell)$，因为每个辅助顶点至多被预搜索或搜索一次（在其被标记之前或之后）。接下来，我们将说明如何将后一类查询实现为每个被标记元素只需对数次查询。这里有两种搜索下一个被标记元素的方式。

首先，要在给定 e 和索引 i的情况下，找出满足 $I-d+e \in$ I的元素 $d \in I_i \setminus M_i$。这可以通过按任意顺序维护 I_i 来实现，只要确保 M_i 排在最前面，然后搜索该顺序中张成 e的最短前缀。该顺序中的最后一个元素给出一个元素 d。搜索需要 O(log(r))次预言机查询。（注意，随着元素被添加到 M_i，很容易维护这一顺序。）

第二种操作是找出尽可能小的索引 j，使得 $e \notin$张成span(I_j)。由于I_1、.....、 I_k 按递减顺序排列，我们可以通过二分搜索找到满足 O的第一个索引 j，所需探测次数为(log k)。每次探测对应一次独立性查询。此外，这次搜索也会结束搜索，因此我们只需搜索一次这样的索引 j。 □

保持 I_1、.....、 I_k 按递减顺序排列。接下来，我们来处理这样一个事实：我们对增广的特定选择会使 I_1、.....、 I_k 按递减顺序排列。

引理 8.14. 由搜索算法导出的增广会使 $I_1,\ldots,I_k$ 按递减顺序排列。

证明. 假设路径的终点表示将元素 e 插入 I_i。我们首先注意到，之前的交换不会影响张成。然后，在将 e 插入 I_i 时，根据对 i 的选择，我们知道，对于所有 $j < i$，$e \in$ 张成(I_{i-1})。因此 $I_i + e \in$ 张成(I_{i-1})，并且我们保持递减顺序。

□

综合起来。引理 8.8 和 8.13 给出了寻找一条增广路径的整体运行时间。（应用引理 8.8时，我们注意到任意一条 (s,t)-路径的长度至多为O(最优值)。）至此，引理 8.1的证明完成。

8.5 图拟阵

本节最后，我们将基于预言机的运行时间转化为图拟阵的具体运行时间。这里的思路与第 5 节类似。通过在 I_i 上维护并查集数据结构，我们可以维护并实现独立性查询，其开销为 I_i，另加 $\alpha($ $n)$。此外，通过在链接-切割树中管理每个 I_i，我们可以在 $O($ $n)$ 时间内检索未标记的边 $d \in I_k \backslash$ M_k 以进行交换（省去了预言机模型中的二分搜索）。重新考察引理 8.8 和 8.13 的证明，并注意到路径的最大长度为最优值，可知每条增广路径的耗时为 $O(m'\alpha($ $n) +$ 最优值 $\log(nk))$，其中 $m' = \min\{m, n\log(nk)\}$。综合考虑 ϵ 的取值与 $(1-\epsilon)$-近似算法的运行时间 $O(m\alpha($ $n)$ 最优值 $\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$，得到如下结果。

定理 8.15。对于具有整数边容量的图，可以在

$$O\Big(m\alpha(n) + \sqrt{\mathrm{OPT}(\min\{m, n\log(nk)\}\alpha(n) + \mathrm{OPT}\log(nk))\log(n)(\log(n) + \log(k)\alpha(n))}\Big)$$

时间。

一种简化的 连通图运行时间，注意到对于 $k \geq 1$，最优值 $\geq \Omega(n)$，i s

$$O\Big(m\alpha(n) + \mathrm{OPT}^{3/2}\log^2(nk)\alpha^2(n)\Big).$$

参考文献

[AO89] Ravindra K. Ahuja 和 James B. Orlin。“最大流问题的一种快速而简单的算法”。载于：运筹学研究37.5（1989年），第748–759页。[Bar92] Francisco Barahona。“从生成树多面体的支配面中分离”。载于：运筹学研究快报12.4（1992年），第201–203页。[Bar95] Francisco Barahona。“生成树打包”。载于：数学运筹学20.1（1995年），第104–115页。[BBMN21] Joakim Blikstad、Jan van den Brand、Sagnik Mukhopadhyay 和 Danupon Nanongkai。“打破拟阵交的二次障碍”。载于：*STOC '21*：第*53*届*ACM SIGACT*计算理论年会，线上会议，意大利，*2021*年*6*月*21*日至*25*日。Samir Khuller 和 Virginia Vassilevska Williams 编。ACM，2021年，第421–432页。

[Bli21] Joakim Blikstad。“打破拟阵交中的 O(nr) 界”。载于：第48届自动机、语言与编程国际研讨会（ICALP 2021），2021年7月12日至16日，苏格兰格拉斯哥（线上会议）。Nikhil Bansal、Emanuela Merelli 和 James Worrell 编。第198卷。LIPIcs。Schloss Dagstuhl - Leibniz-Zentrum für Informatik，2021年，31:1–31:17。[CC94] Eddie Cheng 和 William H. Cunningham。“一种更快的计算网络强度的算法”。载于：信息处理快报49.4（1994年），第209–212页。[CG97] Boris V. Cherkassky 和 Andrew V. Goldberg。“最大流问题中推送-重标记方法的实现”。载于：Algorithmica 19.4（1997年），第390–410页。初步版本发表于 IPCO，1995年。[CGM98] Boris V. Cherkassky、Andrew V. Goldberg 和 Paul Martin。“增广或推送：二分匹配与单位容量流算法的计算研究”。载于：ACM实验算法学期刊3（1998年），第8页。初步版本发表于 WAE，1997年。[CLSSW19] Deeparnab Chakrabarty、Yin Tat Lee、Aaron Sidford、Sahil Singla 和 Sam Chiu-wai Wong。“更快的拟阵交算法”。载于：第60届IEEE计算机科学基础年会（FOCS 2019），美国马里兰州巴尔的摩，2019年11月9日至12日。David Zuckerman 编。IEEE计算机学会，2019年，第1146–1168页。[CM89] Joseph Cheriyan 和 S. N. Maheshwari。“最大网络流预流推送算法分析”。载于：SIAM计算杂志18.6（1989年），第1057–1086页。初步版本发表于 FSTTSC，1988年。[CQ17] Chandra Chekuri 和 Kent Quanrud。“若干隐式分数装箱问题的近线性时间近似方案”。载于：第二十八届ACM-SIAM离散算法研讨会论文集（SODA 2017），西班牙巴塞罗那，Hotel Porta Fira，1月16日至19日。SIAM，2017年，第801–820页。[Cun84] William H. Cunningham。“拟阵多面体成员资格检验”。载于：组合论杂志B辑36.2（1984年），第161–188页。[Cun85] William H. Cunningham。“网络的最优连接与加固”。载于：ACM杂志32.3（1985年），第549–561页。[Cun86] William H. Cunningham。“拟阵划分与交算法的改进界”。载于：SIAM计算杂志15.4（1986年），第948–957页。[Edm65a] Jack Edmonds。“Lehman的切换博弈与Tutte和Nash-Williams的一个定理”。载于：美国国家标准局研究杂志B辑69（1965年），第72–77页。[Edm65b] Jack Edmonds。“将拟阵划分为独立子集的最小划分”。载于：美国国家标准局研究杂志B辑69（1965年），第67–72页。[ET75] Shimon Even 和 Robert Endre Tarjan。“网络流与图连通性检验”。载于：SIAM计算杂志4.4（1975年），第507–518页。[FI00] Lisa Fleischer 和 Satoru Iwata。“次模函数最小化与次模流的改进算法”。载于：第三十二届ACM计算理论年会论文集，2000年5月21日至23日，美国俄勒冈州波特兰。F. Frances Yao 和 Eugene M. Luks 编。ACM，2000年，第107–116页。[FI03] Lisa Fleischer 和 Satoru Iwata。“用于次模函数最小化的推送-重标记框架及其在参数优化中的应用”。载于：离散应用数学131.2（2003年），第311–322页。

[FM12] András Frank 和 Zoltán Miklós。“拟阵与次模流的简单推送-重标记算法”。载于：《日本工业与应用数学杂志》29（2012），第419–439页。[FZ92] Satoru Fujishige 和 Xiaodong Zhang。“次模系统交问题的新算法”。载于：《日本工业与应用数学杂志》9（1992），第368–392页。[FZ96] Satoru Fujishige 和 Xiaodong Zhang。“次模流的推送/重标记框架及其在0-1次模流中的扩展”。载于：Optimization 38.2（1996），第133–154页。[Gab91a] Harold N. Gabow。“一种用于求边连通度和打包分支定向树的拟阵方法”。载于：第23届ACM计算理论研讨会论文集，1991年5月5日至8日，美国路易斯安那州新奥尔良。Cris Koutsougeras 和 Jeffrey Scott Vitter 编。ACM，1991，第112–122页。[Gab91b] Harold N. Gabow。“一种用于求边连通度和打包分支定向树的拟阵方法”。载于：第23届ACM计算理论研讨会论文集，1991年5月5日至8日，美国路易斯安那州新奥尔良。Cris Koutsougeras 和 Jeffrey Scott Vitter 编。ACM，1991，第112–122页。[Gab98] Harold N. Gabow。“图拟多阵与参数化 s-集的算法”。载于：J. Algorithms 26.1（1998），第48–86页。初步版本发表于SODA，1995年。[GGT89] Giorgio Gallo、Michael D. Grigoriadis 和 Robert Endre Tarjan。“快速参数化最大流算法及其应用”。载于：SIAM计算杂志 18.1（1989），第30–55页。[GM75] C. Greene 和 T.L. Magnanti。“一些抽象枢轴算法”。载于：SIAM应用数学杂志 29（1975），第530–539页。[GM98] Harold N. Gabow 和 K. S. Manu。“容量图中分支定向树（及生成树）的打包算法”。载于：Math. Program. 82（1998），第83–109页。初步版本发表于IPCO 1995年。[Gol85] Andrew Vladislav Goldberg。《一种新的最大流算法》。技术报告 MIT/LCS/TM-291。美国马萨诸塞州剑桥：麻省理工学院计算机科学实验室，1985年。[Gol87] Andrew Vladislav Goldberg。“适用于串行和并行计算机的高效图算法”。博士论文。麻省理工学院，美国马萨诸塞州剑桥，1987年。[GS85] Harold N. Gabow 和 Matthias F. M. Stallmann。“图拟阵交与奇偶性的高效算法（扩展摘要）”。载于：《自动机、语言与程序设计》第12届研讨会，希腊纳夫普利翁，1985年7月15日至19日，论文集。Wilfried Brauer 编。第194卷，《计算机科学讲义》。施普林格，1985，第210–220页。[GT88] Andrew V. Goldberg 和 Robert Endre Tarjan。“最大流问题的新方法”。载于：ACM杂志 35.4（1988），第921–940页。初步版本发表于STOC，1986年。[GT90] Andrew V. Goldberg 和 Robert E. Tarjan。“通过逐次近似求解最小费用循环流”。载于：数学运筹学 15.3（1990），第430–466页。初步版本发表于STOC，1987年。

[Gus83] Dan Gusfield。“连通性与边不相交生成树”。载于：信息处理快报 16.2（1983），第87–89页。[Gus91] Dan Gusfield。“图强度计算”。载于：SIAM计算杂志 20.4（1991），第639–654页。[GW92] Harold N. Gabow 和 Herbert H. Westermann。“森林、框架与博弈：拟阵和的算法及其应用”。载于：Algorithmica 7.5&6（1992），第465–497页。初步版本发表于STOC, 1988年。[HK73] John E. Hopcroft 和 Richard M. Karp。“二分图最大匹配算法”。载于：SIAM计算杂志 2.4（1973），第225–231页。初步版本发表于FOCS，1971年。 $n^{5/2}$ HO[Jianxiu Hao 和 James B. Orlin。“在有向图中寻找最小割的更快算法”。载于：J. Algorithms 17.3（1994），第424–446页。初步版本发表于SODA，1992年。94] HRG[Monika Rauch Henzinger、Satish Rao 和 Harold N. Gabow。“计算顶点连通性：基于旧技术得出的新界”。载于：J. Algorithms 34.2（2000），第222–250页。初步版本发表于FOCS，1996年。00] Ima[Hiroshi Imai。“下截横拟多阵的网络流算法”。载于：《日本运筹学会杂志》26.3（1983），第186–211页。83] IMS[Satoru Iwata、S. Thomas McCormick 和 Maiko Shigeno。“次模流的快速费用缩放算法”。载于：信息处理快报 74.3-4（2000），第123–128页。00] IMS[Satoru Iwata、Kazuo Murota 和 Maiko Shigeno。“强映射序列的快速参数化次模交算法”。载于：数学运筹学 22.4（1997），第803–813页。97] IO[Satoru Iwata 和 James B. Orlin。“次模函数最小化的简单组合算法”。载于：第二十届ACM-SIAM离散算法研讨会论文集（SODA 2009），美国纽约州纽约, 2009年1月4日至6日。Claire Mathieu 编。SIAM，2009，第1230–1237页。09] Kar[David R. Karger。“拟阵优化问题的随机抽样与贪心稀疏化”。载于：Math. Program. 82（1998），第41–81页。初步版本发表于FOCS 1993年。98] Knu[Donald E. Knuth。《拟阵划分》。技术报告，报告编号STAN-CS-73-342。美国加利福尼亚州斯坦福：斯坦福大学计算机科学系，1974年。74] Nar[Harihar Narayanan。“拟阵成员判定问题的舍入技术”。载于：《线性代数及其应用》221（1995），第41–57页。95] Nas[C. St. J. A. Nash-Williams。“有限图的边不相交生成树”。载于：《伦敦数学学会杂志》36（1961），第445–450页。61] Nas[Crispin St. John Alvah Nash-Williams。“拟阵在图论中的一个应用”。载于：《图论——国际研讨会——图论国际研讨会论文集》（罗马，1966年）。R. Rosenstiehl 编。Gordan、Breach（纽约）和 Dunod（巴黎），1967，第263–265页。67] Ngu[Huy L. Nguyen。“关于拟阵交的 Cunningham 算法的一则注记”。载于：CoRR abs/1904.04129（2019）。arXiv：1904.04129。

[RT85] James Roskind 和 Robert E. Tarjan。“关于寻找最小成本边不相交生成树的注记”。载于：数学运筹学 10.4（1985年），第 701–708 页。[Sch03] Alexander Schrijver。组合优化：多面体与效率。第 24 卷。算法与组合学。施普林格，2003年。[ST83] Daniel Dominic Sleator 和 Robert Endre Tarjan。“动态树的数据结构”。载于：*J. Comput. Syst. Sci.* 26.3（1983），第 362–391 页。初版发表于 STOC 1981。[Tru93] V. A. Trubin。“图的强度以及树和分枝的打包”。载于：*Cyber-netics and Syst. Analysis* 29.3（1993），第 379–384 页。[Tun94] Levent Tunçel。“最大流问题预流-推送算法的复杂度”。载于：*Algorithmica* 11.4（1994年），第 353–359 页。[Tut61] W. T. Tutte。“关于将图分解为 n 连通分量的问题”。载于：*J. London Math. Soc.* 36（1961），第 221–230 页。

A 附加背景

本节补充一些第 1.1 节中省略且本文其他部分未提及的背景知识。

回顾一下最大化森林打包规模的问题（即图拟阵中的 k重拟阵并问题）。此前已有多项研究探讨这一问题 [GS85；GW92；Ima83；RT85]。 [GW92] 得出了若干运行时间，每种运行时间在某些参数范围内都是最优的；我们在第 1.1 节重点介绍了其中最具可比性的结果。

通过 Nash-Williams [Nas61] 和 Tutte [Tut61]的经典定理，生成树打包与网络强度之间还有重要的联系，下面对此进行说明。给定顶点集 $(V_1, \dots, V_k)$ 的一个划分 V，该划分的强度定义为以下比值

$$|\partial(V_1, \dots, V_k)|/(k-1),$$

其中 $\partial(V_1, \dots, V_k)$ 表示由划分 $V_1, \dots, V_k$切割得到的边集。Cunningham[Cun85] 在网络脆弱性的背景下提出了对划分强度的一种解释：该强度反映了每增加一个由切割产生的连通分量所需的成本。图的（网络）强度定义为所有划分强度中的最小值。Gusfield [Gus83]提出将网络强度作为衡量网络脆弱性的指标。定义可自然推广至正边容量。[Nas61；Tut61]的定理表明，网络强度等于任意（分数）树打包的最大规模；任意整数树打包的最大规模则等于网络强度的向下取整。

关于容量图和无容量图中的生成树打包，已有多项研究[Bar95; Cun84; Gab91b; GM98; GW92; Ima83; RT85; Tru93]。除第 1.1 节所述的运行时间外，[GM98] 还获得了一个强多项式运行时间 $O(n^3 m \log(n^2/m))$。除了通过生成树打包之外，还有研究通过流或参数流直接计算强度 [Bar92; CC94; Cun85; Gab98; Gus91] ，最终得到通过参数流（及相关技术）计算强度的 $O(n^2 m \log(n^2/m))$ 时间算法[CC94; Gab98]。对于树覆盖，除了第 1.1，[节提到的运行时间外，98] GM $O(n^3 m$还为容量图中的树覆盖实现了$(n^2/m))$ log运行时间，与打包相同。