arXiv:2602.05948v1 [cs.DC] 5 Feb 2026

匿名图上的位置感知分散

Himani[1], Supantha Pandit[1], Gokarna Sharma[2]

[1]Dhirubhai Ambani University，印度古吉拉特邦甘地讷格尔，202221003@dau.ac.in，pantha.pandit@gmail.com[2]肯特州立大学，美国俄亥俄州肯特，gsharma2@kent.edu

摘要

研究充分的 Dispersion 问题是分布式机器人学中的一个基本协调问题：一组移动机器人必须重新定位，使每个机器人占据网络中一个不同的节点。 Dis-persion 假设机器人可以在任意节点定居，只要没有其他机器人在该节点定居。在本研究中，我们提出了 Location-Aware Dispersion，这是对 Dispersion 的一种新颖推广，纳入了位置感知能力：设 $G = (V, E)$为一个匿名、连通的无向图，包含$n = |V|$ 个节点，每个节点都标记有一种颜色 col$(v) \in \mathcal{C} = \{c_1, \ldots, c_t\}, t \leq n$。给定一个由$k \leq n$ 个移动机器人组成的集合 $\mathcal{R} = \{r_1, \ldots, r_k\}$，其中每个机器人 r_i 都关联一种颜色 col$(r_i) \in \mathcal{C}$。机器人最初被任意放置在图上，目标是重新定位机器人，使每个机器人都占据一个颜色相同且彼此不同的节点。当 $|\mathcal{C}| = 1$, Location-Aware Dispersion 时，问题退化为 Dispersion。对于任意 $k \leq n$ 的图，无需知道 k, n，均存在 Dispersion 的解。

与 Dispersion一样，目标是解决 Location-Aware Dispersion ，同时尽可能降低每个智能体的时间和内存需求。我们提出了若干确定性算法，并为其时间和内存需求给出了保证的界限。我们还给出了针对 Location-Aware Dispersion的任意确定性算法的不可能性结果和下界。据我们所知，这些结果共同确立了 Location-Aware Dispersion在匿名网络中的算法可行性，同时也凸显了与Dispersion的解决方案相比，获得高效解法所面临的挑战。

1 引言

这一经过深入研究的问题最初由 Augustine 和 Moses Jr. 提出。 [1], 要求 $k \leq n$ 移动机器人从初始位置出发（位于一个 n节点图 $G =$ （V， E）的节点上），自主重新定位，使每个节点至多包含一个机器人。本文提出并研究 Location-AwareDispersion 问题，这是 Dispersion 的推广，将位置感知纳入其中。

Location-Aware Dispersion。设 $G = (V,\ E)$ 是一个连通、无向的匿名图，具有 n个节点。每个节点 $v \in V$都有一个颜色col$(v) \in \mathcal{C} = \{c_1, \ldots, c_t\}$，多个节点可以具有相同的颜色。给定一组 $\mathcal{R} = \{r_1, \ldots, r_k\}$ 个 $k \leq n$移动机器人，每个机器人 $r_i \in \mathcal{R}$ 都有一个颜色col$(r_i) \in \mathcal{C}$。初始时，机器人任意分布在 G上。目标是重新安置这些机器人，使每个机器人都占据一个颜色与其自身颜色相匹配的不同节点，且任意两个机器人不占据同一节点。

该变体对机器人必须分配到类别特定任务或资源的场景进行建模。例如，在多公司充电基础设施中，节点代表属于不同服务提供商的充电站，机器人（或电动汽车）必须被路由到相应服务提供商的站点。更一般地说，Location-Aware Dispersion 捕捉了物流、交通运输和多机器人系统中的协调问题，其中机器人受到类别特定要求的约束。当 $|\mathcal{C}| = 1$、 Location-AwareDispersion 简化为 Dispersion时，机器人可以在任意节点定居，只要没有其他机器人在该节点定居即可。

Dispersion 的目标是让机器人分布在不同节点上；一旦实现无碰撞（分散）配置，问题便视为解决。相比之下，仅有分散配置

并不足以解决 Location-Aware Dispersion，也就是说，即使所有机器人都占据不同节点，如果某个机器人所在节点的颜色与其自身颜色不匹配，该配置仍可能无效。因此，对于 Location-Aware Dispersion，分散配置虽然是 Dispersion的解，却只是必要条件而非充分条件。此外，已有研究表明，对于任意数量的机器人 $k \leq n$，即使不了解 k或 n， Dispersion 在任意 n节点图上始终可解。相比之下，我们将证明 Location-Aware Dispersion 并非总是可解。这意味着Location-Aware Dispersion 比 Dispersion 更难。因此，与 Dispersion相比， Location-Aware Dispersion 带来了额外挑战，因为机器人不仅必须避免碰撞，还必须遵守颜色约束。与 Dispersion一样，这里有两个核心复杂度度量：时间复杂度，即轮次数；以及空间复杂度，即每个机器人的持久内存量。图中的节点是匿名且无记忆的。我们的目标是为 Location-Aware Dispersion 设计兼顾时间与内存效率的确定性算法，并确立不可能性结果和下界，以刻画解决该问题所需的最低条件。

模型。我们考虑一个无向、无权图 $G = (V, E)$，包含 $n = |V|$ 个节点和 $m = |E|$ 条边。节点 $v \in V$被分配一种颜色 $c_i, \mathcal{C} = \{c_1, c_2, ..., c_t\}, t \leq n$，是所有用于节点的颜色集合。每个节点 $v \in V$ 都是无内存的，也就是说，它不存储任何信息。与 v 相连的每条边都使用端口号 $[1, \delta_v]$进行标记，其中 δ_v 是节点 v的度。因此，连接两个不同顶点的边在两端各有一个端口号。边两端的端口标签之间没有关联。

我们考虑放置在 G 的节点上的 $k \leq n$ 个机器人，该节点记为 $\mathcal{R} = \{r_1, r_2.....r_k\}$。机器人在 G 上的初始排列对应于以下情况：(i) 有根配置，即所有 $k \leq n$ 个机器人共置于一个源节点；(ii) 分散配置，即每个节点初始时至多有一个机器人；以及 (iii) 一般配置，即初始放置既不是有根配置，也不是分散配置。每个机器人都有一个范围为 $[1, k^{O(1)}]$的唯一标识。如果机器人知道其初始配置的类型（有根、分散或一般），则称该配置为已知配置；否则称为未知配置。机器人 $r \in \mathcal{R}$ 被分配一种颜色 c_i，其中 $\mathcal{C} = \{c_1, c_2, ..., c_t\}$ 是所有用于机器人的颜色集合，且 $t \leq k$。机器人可以通过边在顶点之间移动；但是，它们只能占据顶点，不能停留在边上。我们考虑局部通信模型（又称面对面模型），即位于节点 $v \in V$ 的机器人可以通信，并且只能与位于 v的其他共处机器人共享数据和标识符。

机器人按照通信-计算-移动（CCM）周期运行；每个 CCM 周期构成一个轮次。因此，在每一轮中，位于节点 v 的机器人 $r \in \mathcal{R}$ 将执行以下步骤：

-通信： r 与节点 v 上共处的机器人进行通信。

-计算： r 根据通信阶段共享的信息进行局部计算，并决定是否离开 v、通过哪个端口离开，以及向留在 v 的机器人写入什么信息。

-移动：在此阶段， r 通过计算得出的端口离开 v。

我们的贡献。我们关于 Location-Aware Dispersion 的主要结果总结于表 1，而与 Dispersion 相关的最新研究成果的比较见表 2。值得强调的是， Location-Aware Dispersion 的下界严格高于 Dispersion 的下界；此外，当 $k = 1$ 未知时， Location-Aware Dispersion 不可能得到解决。这与 Dispersion 形成对比：在这些条件下， Dispersion 仍然可以得到解决（见第 2 节）。

我们首先介绍一种确定性算法，用于在预先知道 n 和 k 的情况下处理有根配置。随后，我们将结果扩展到预先已知的配置，即有根配置、分散配置和一般配置，并同时考虑已知 n 和 k 的情形，以及无法获得此类先验知识的情形（第 3 至 6 节）。最后，在第 7 节中，我们研究初始配置未知的情形。综上，这些结果对匿名端口标记无记忆网络中 Location-Aware Dispersion 的复杂度给出了近乎紧确的刻画，阐明了其算法可行性及其根本局限。

相关工作。Dispersion 是与 Location-Aware Dispersion最密切相关的问题。 Dis-persion 最初由 Augustine 和 Moses Jr. 于 [1]提出。后来，该问题得到了更深入的研究

情况	Kwn	有根	一般	分散
特殊情况				
$k \leq n$: 树, 路径, Ring	Nil	T: $O(n)$, M: $O(\log(k+\Delta))$ 推论1	T: $O(n)$, M: $O(\log(k+\Delta))$ 推论1	T: $O(n)$, M: $O(\log(k+\Delta))$ 推论1
$k=1$	k,n	T: $\tilde{O}(n^5)$, M: $O(M^*)$ 定理: 1	-	T: $\tilde{O}(n^5)$, M: $O(M^*)$ 定理: 1
	k	不可解 , 定理: 1	-	不可解 , 定理: 1
$k=n$	k,n	T: $O(m)$, M: $O(\log k)$ 定理: 3	T: $O(n/k \cdot m)$, M: $O(n/k \cdot$ 对数 $k)$ 定理: 6	T: $O(\log k + n/k \cdot m)$, M: $O(n/k \cdot \log k)$, 定理: 8
对于已知配置				
$2 \leq k \leq n$	n	T: $O(n/k \cdot m)$ M: $O(n/k \cdot \log(k+\Delta))$ 定理 4	T: $O(\log(n/k) \cdot n/k \cdot m)$ M: 如果有 $\exists$ 个源点和 >2 个机器人: $O(n/k \cdot \log(k+\Delta))$, 否则: $O(n \cdot \log(k+\Delta))$ 定理 7	T: (i) $O(n^3 + n/k \cdot m)$, $i=1$, 2 (ii) $O(n^i \cdot \log n + n/k \cdot m)$, $i=3$, 4, 5 (iii) 否则: $\tilde{O}(n^5)$ M: $O(M^* + n/k \cdot \log(k+\Delta))$ 定理 9
	Nil	T: $O(\log(n/k) \cdot n/k \cdot m)$, M: $O(n/k \cdot \log(k+\Delta))$, 定理 5	定理 7	开放问题
对于未知配置				
$2 \leq k \leq n$	n	若已知 k : 定理 4 如果 k 未知 T: $\tilde{O}(n^5)$, M: $O(M^* + n/k \cdot \log(k+\Delta))$ 定理: 10	T: (i) $k \geq \lfloor n/2 \rfloor + 1$: $O(n^3 + n/k \cdot m)$, (ii) $k \geq \lfloor n/3 \rfloor + 1$: $O(n^4 \log n + n/k \cdot m)$, (iii) 否则: $\tilde{O}(n^5)$ M: $O(M^* + n/k \cdot \log(k+\Delta))$ 定理: 10	T: (i) $k \geq \lfloor n/2 \rfloor + 1$: $O(n^3 + n/k \cdot m)$ (ii) $k \geq \lfloor n/3 \rfloor + 1$: $O(n^4 \log n + n/k \cdot m)$ (iii) 否则: $\tilde{O}(n^5)$ M: $O(M^* + n/k \cdot \log(k+\Delta))$ 定理: 10
	Nil	若 k 已知: 定理 4 若 k 未知: 开放问题	开放问题	开放问题
下界				
下界	T: $\Omega(\min\{n^2/k, m\})$, M: $\Omega(n/k \cdot \log k)$, 定理: 2			

表 1: 关于 Location-Aware Dispersion的研究结果。 M^* 表示通用探索序列（UXS） [4]所需的内存。 i 是机器人之间的最小距离， n （k）是节点（机器人）的数量， Δ 是最大度数。"-"表示不可能的配置，"Kwn"列表示机器人是否拥有关于 k 或 n的先验知识；未注明的条目表示此类知识可能可用，也可能不可用，而"Nil"表示没有先验知识。在界限中，"T:"和"M:"~分别表示每个智能体所需的时间和内存，其中 O 省略了多对数因子。

配置	论文	时间复杂度	空间复杂度
有根	[9]	$O(k)$	$O(\log k + \Delta)$
	[9]	$O(k$ 对数 $k)$	$O(\log(k+\Delta))$
	[8]	$O(k)$	$O(\log(k+\Delta))$
一般	[9]	$O(k$对数$^2\ k)$	$O(\log(k+\Delta))$
	[8]	$O(k)$	$O(\log(k+\Delta))$
下界	[1]	$\Omega(k)$	$\Omega(\log\ k)$

表 2: Dispersion上的最先进研究成果。

关于 $k \leq n$ 在 [5–9], 中的情况，研究者在机器人拥有某些参数的先验知识时，针对一般图进行了详细研究；而 [10] 则研究了机器人没有此类先验知识时的情况，采用的模型是局部通信模型，在该模型中，位于同一节点的机器人可以相互通信。该问题还在其他设置中得到研究，例如动态图 [11, 12], 异步模型 [5, 12], 以及全局通信模型 [13, 14]。此外，研究者还研究了正方形网格 [14, 15] 和三角网格 [16]上的分散问题。

与 Dispersion 相比， Location-Aware Dispersion 更具挑战性，因为某些情况将无法求解。对于可求解的情况，其下界高于 Dispersion 下界。此外，参数知识和初始配置知识对于能够实现何种解决方案及保证起着重要作用。最近，Sudo 等人 [9] 在同步环境中取得了一项重要成果，为一般初始配置提出了两种算法：(i) 一种时间最优的 $O(k)$

算法，其内存为 $O(\Delta + \log k)$；(ii) 一种运行时间为 $O(k \log^2 k)$、内存为 $O(\log(k + \Delta))$ 的算法。虽然第一项成果实现了时间最优，但其内存需求高得难以接受；第二项成果虽然实现了近乎最优的内存需求，却牺牲了时间最优性。Kshemkalyani 等人 [8] 消除了这些局限，提出了首个能够同时实现两个目标的同步算法：在最优的 $O(k)$ 轮次内运行，且每个机器人仅需 $O(\log(k+\Delta))$ 比特的内存。与 [9], 的已知最佳时间最优成果相比，这使时间减少了 $O(\log^2 k)$ 倍，从而弥合了时间最优与内存最优之间的差距。

本工作的结果总结于表 2 中。我们研究了有根、分散和一般配置下的 Location-Aware Dispersion 问题，并考虑机器人对 n 和 k的知识所作的不同假设。当初始配置未知时，我们进一步分析了 Location-Aware Dispersion 。

2 不可能性、下界和特殊情况

本节研究确定性机器人模型下 Location-AwareDispersion 问题的基本局限性和性能保证。我们首先确立一个不可能性结果，表明在某些设置下，无论机器人拥有多少本地内存，都无法解决该问题。随后，我们推导出时间和内存需求的一个下界，任何确定性解决方案在最坏情况下都必须满足该下界。最后，我们从图结构和初始配置两方面确定并分析若干特殊情况，在这些情况下可以规避上述否定性结果，并能够设计出高效的确定性算法。具体而言，我们针对树、路径和环，以及任意图中的有根配置，给出了最优或近似最优的解决方案。

不可能性 我们将证明，在某些假设下， Location-Aware Dispersion 问题不存在确定性解。

定理 1（不可能性）。设 $G = (V, E)$ 是一个有 n 个节点的匿名、端口标记无向图。对于单个机器人（$k = 1$），如果未知 n，则 Location-Aware Dispersion 问题无解，即使本地内存不受限制也是如此。但是，如果已知 n，则 Location-Aware Dispersion问题可在 $\tilde{O}(n^5)$ 时间内解决。

证明。如果机器人知道 n [4], 就可以计算并执行长度为$\tilde{O}(n^5)$ 的通用探索序列，从而保证完全探索任何有 n 个节点的匿名、端口标记图，进而解决 Location-Aware Dispersion。如果未知 n，那么 [2], 在匿名图中的确定性单个机器人无法区分某些图配置，因而可能陷入无限循环，反复访问相同的节点，而其他地方仍有未探索节点。因此，即使本地内存不受限制，它也无法保证完全探索；若没有关于n的先验知识，该问题便无解。

□

时间与记忆　　位置感知分散的下界 现在我们给出下界　　.

定理2（下界）。存在一个包含 $k \leq n$ 个机器人的初始配置，位于一个有 n 个节点的图上，使得任何用于 Location-Aware Dispersion 的确定性算法都需要 $\Omega(\min\{n^2/k, m\})$ 轮次，并且每个机器人需要 $\Omega(n/k \cdot \log k)$ 比特。

证明。考虑一个有 n 个节点的图 G，其构造如下。将节点划分为 n/k 组（假设 n/k 是整数），每组包含 k 个节点。将组 ℓ 中的节点记为 $v_{\ell k+1}, v_{\ell k+2}, \ldots, v_{\ell k+k}$，其中 $0 \leq \ell \leq n/k - 1$。每组构成一个有 k 个节点的完全图。在每组内，移除边 $(\ell k+1, \ell k+k)$ 和 $(\ell k+2, \ell k+k-1)$。然后，使用桥接边 $(\ell k+k, \ell k+k+1)$ 和 $(\ell k + k - 1, \ell k + k + 2)$ 连接连续组。

假设有一个红色机器人位于 v_1，除 v_n 为红色外，所有节点均为蓝色。要实现正确的彩色分散，机器人必须区分不同的组，并追踪哪些节点已经访问过。

由于有 n/k 个组，每个组都必须能够被唯一标识，因此至少需要 $\log(n/k)$ 比特来编码组标识符。在每个组内，机器人必须确定性地探索 k 个节点

以确保正确放置，并避免不必要地重复访问节点。在每一步中，机器人都必须选择正确的端口，以到达未访问节点，并最终通过桥接边前往下一个组。为了编码其当前位置，并在包含 k 个节点的组内做出确定性决策，机器人还需要 $\log k$ 比特的内存。

综合组间和组内要求，为了正确维护组信息并确保任意图上的确定性彩色分散，每个机器人至少需要 $\Omega\big((n/k)\cdot\log\ k\big)$ 比特的内存。

□

特殊情况的高效算法 我们现在讨论两个特殊情况B1和B2。B1处理特殊图：树、路径和环，而B2处理任意图中已知 k、n 和 $k=n$的有根初始配置。

B1：树、路径、环：有根、一般或分散配置，$k \le n$ （算法1）：我们采用基于深度优先搜索的策略。每个机器人维护两种状态：探索和定居。在每个节点 v处，颜色与节点颜色匹配的最小ID机器人转换到定居状态，而处于探索状态的所有其他机器人则通过端口-(pent + 1) mod δ_v退出该节点，其中pent是机器人进入 v 所经过的端口，δ_v是 v的度。这确保了系统遍历且不会重复访问，从而在 $O(n)$轮次内得到解决方案。最初，所有机器人从一个或多个源节点出发。一般而言，在分散配置中，每组机器人或单个机器人都会独立执行其深度优先搜索探索。已经在某个节点定居的机器人会留在那里，以防止冲突，而其他机器人会忽略被占据节点并继续探索。这保证每个节点至多有一个机器人定居，从而实现 Location-Aware Dispersion。每个机器人需要 $O(\log(k+\Delta))$比特来存储其状态、颜色和端口。

推论1。对于树、路径和环中的任意初始配置，Location-Aware Disper-sion 可在 $O(n)$ 轮次内、使用每个机器人 $O(\log(k+\Delta))$ 比特求解。

证明。该证明直接由标准深度优先搜索遍历得出。在每个节点 v 处，机器人 r 当且仅当 $\text{col}(r) = \text{col}(v)$ 且 r 在符合条件的机器人中具有最小 ID 时定居。对于多源配置，每个源点发起各自的深度优先搜索遍历。当一次遍历遇到已访问节点时，它继续其深度优先搜索而不定居。若该节点未被访问，则颜色与该节点匹配且具有最小 ID 的机器人在此定居，其余机器人继续遍历。

因此，分散过程会在 $O(m)$ 轮次内完成。对于树或路径，$m = n-1 = O(n)$；对于环，$m = n = O(n)$。因此，在所有这些情况下，该问题都可以在 $O(n)$ 轮次内解决。机器人用于存储变量的内存为 $O(\log(k+\Delta))$ 比特。

□

算法1：Location-Aware Dispersion_Tree_Path_Ring(r) // 对于 $k \le n$ 机器人

```
// R[v] = 位于节点 v 的机器人
// 初始时 r.p = −1，其中 p=pent
如果 r.state = explore 则
    如果 v is empty and col(v) = c_i 则
        如果 r ∈ R[v] is the minimum ID robot with col(r) = c_i 则
            r.state = settle
    else
        通过 (p + 1) mod δ_v 退出
```

B2：任意图、有根配置，已知 $n, k,\ k=n$ （算法 2）：对于从有根配置出发的 $k=n$ 个机器人，基于深度优先搜索的遍历可确保每个节点恰好分配一个机器人。每个机器人都维护一个状态（探索或定居）和一个阶段（前向或回溯）。在前向阶段，处于探索状态的机器人移动到未访问节点。在每个节点，颜色与节点颜色匹配的最小 ID 机器人转为定居状态，而其他机器人则通过 (pent + 1) mod δ_v 退出。当某个节点的所有子节点都已探索后，机器人进入回溯阶段，并沿入口端口返回，从而实现对图的系统性覆盖。已定居机器人存储父节点和子节点指针，以支持前向和回溯移动。这保证了使用 $O(\log(k+\Delta))$ 比特数的每个机器人空间，可以在 $O(m)$ 中求得 Location-Aware Dispersion 的解。

定理 3. 对于任意图中任意有根初始配置下的 $k = n$ 机器人，若机器人已知 k, n ，则 Location-Aware Dispersion 可在 $O(m)$ 轮次内解决，每个机器人使用 $O(\log k)$ 比特。

证明。根据算法，机器人从源节点 S 开始共同执行深度优先搜索（DFS）遍历。在每个已访问节点 v 处，如果存在颜色与 v 匹配的机器人，则该机器人永久驻留在 v，其余机器人则按照规则继续进行深度优先搜索遍历。由于深度优先搜索会访问 G 的每个节点，因此每个节点最终都会被到达。此外， $k = n$ ，并且对于每种颜色 c_i ，颜色为 c_i 的机器人数量等于颜色为 c_i 的节点数量。因此，每个节点最终恰好由一个机器人占据，并且 Location-Aware Dispersion 问题得以解决。
深度优先搜索对每条树边的探索次数最多为两次（一次前进，一次回溯），对每条非树边的探索次数最多为四次。因此，轮次总数为 O（m）。每个机器人只需 O（$\log k$）比特即可存储变量及其自身的标识符。□

算法 2： CD_Arbitrary_Rooted_Graph // 对于 $k = n$ 个机器人

```
   //位于节点 v 的 R[v] = 个机器人
   // 初始时 r.p = −1，其中 p = pent
1  如果 r.state = explore ，则
2  |  如果 r.phase = forward ，则
3  |  |  如果 v is empty and col(v) = c_i ，则
4  |  |  |  如果 r ∈ R[v] is the minimum ID robot with col(r) = c_i ，则
5  |  |  |  |  r.state=settle
6  |  |  |  |  存储 parent(v) 和 child(v) 的值
7  |  |  else
8  |  |  |  r.phase = backtrack 并通过 pent退出
9  |  如果 r.phase = backtrack 则
10 |  |  如果(p + 1) mod δ_v ≠ parent(v)则
11 |  |  |  r.phase = forward
12 |  通过端口-(p + 1) mod δ_v退出
```

3 任意图：有根， $k \leq n$，已知 n

在本节中，我们考虑 $k < n$的情形，所有 k 机器人最初位于单个源节点S。第 2 节（案例B1和B2）中深度优先搜索策略的传统实现可能导致机器人困在环中。机器人受限于有限的内存，无法存储从源点到其预定目标节点的完整路径。在遍历过程中，在每个未访问节点处，颜色与该节点匹配且标识符最小的未定居机器人在此定居，而其余机器人继续探索。如果没有机器人与该节点的颜色匹配，则全部继续前进。在包含颜色不对应任何机器人的节点环的图中，机器人可能无限循环，反复重新访问相同节点而无法到达其目标节点。因此，在此类内存限制下，标准深度优先搜索策略不能保证分散，并可能导致死锁。为克服这一局限，我们提出一种新颖的基于深度优先搜索的过程，将其组织为三个系统化阶段，确保每个机器人最终到达具有匹配颜色的节点并终止求解 Location-Aware Dispersion。

我们假设 $k = \lceil n/c \rceil$ ，其中 $c \leq n$ 为常数。由于在有根设置中，所有机器人都共置于一个单源节点，因此它们可以确定机器人的总数 k。因此，在有根配置中， k 初始时已知还是未知，不会影响模型。

阶段 1：带组的图探索。图节点按指定大小划分为“组”，每组指定一名机器人作为该组的“守卫”（在算法中称为领导者或振荡者）。组大小并非固定，但有上下界，并根据 k and n 确定。深度优先搜索遍历完成后，源节点 S 处的守卫机器人会检测到遍历已完成。算法还会识别一组连接各组的边，称为“组间边”。每个组都会根据这些边记录其父端口（通向父组的端口）和子端口（通向

其子组的端口）。这些组间边保证不构成环。将每个组视为一个“超节点”后，各组及组间边共同构成一棵有向根树$\mathcal{T}$，其根是包含 S 的超节点（“根超节点”）。当根超节点中至少有两名机器人时，深度优先搜索遍历即告完成。

阶段 2：完成连通性信息收集。阶段 1 结束后，目标是将机器人汇集到 S，以获取图中每个节点的完整连通性信息。根超节点处的守卫机器人在 $\mathcal{T}$ 上发起深度优先搜索。沿着这次遍历，它带上其他机器人，最终所有机器人都返回 S。每个机器人存储一部分图结构，而它们共同掌握了源点 S 处图中各节点的完整连通性信息。

阶段 3：内存高效分散。最后的任务是分散。由于机器人掌握了每个节点的连通性信息，因此可以据此确定各自的目标超节点（以及其中的具体节点）。每个机器人最终都必须从 S 前往其已分配的超节点。然而，受内存限制，我们无法向每个机器人提供完整的路径信息。为此，我们提出一种两阶段机制：(i) 在第一阶段，每个超节点保留一个机器人，以建立从各目标超节点到根超节点 S 的路径；(ii) 在第二阶段，所需数量的机器人沿这些路径前往指定的超节点，最后抵达各自确切的目标节点。

3.1 概述、关键挑战与技术

每个阶段都会出现若干关键问题，也各自带来挑战。下面我们逐一探讨：

3.1.1 阶段 1：分组图探索

问题1：在过程中形成组时，深度优先搜索遍历是如何进行的？（参见图 1）

在 Location-Aware Dispersion 中，该策略遵循深度优先搜索，同时要求在遍历过程中形成指定大小的节点组（见问题2）。随着深度优先搜索的推进，各组逐步形成。在前向阶段，机器人集体沿未探索边移动，并指定最低编号机器人在每个新建组中定居（见问题3）；随着遍历推进，还会将未访问节点纳入组中（见问题2）。探索过程中，领导者记录父子关系，并扩展该组，直至其大小达到有效阈值（见问题3）。组形成后，对应于该组的机器人留在组内，其余机器人则继续遍历，以创建后续组。

在遍历过程中，如果机器人遇到已访问节点（即属于现有组），它要么沿着已探索边回溯以继续进行深度优先搜索，要么现有组和当前组会根据探索过程动态缩小或扩大（参见问题5）。每当一个组达到所需规模时，就会启动一个新组（参见问题4）。当所有节点都已探索，且至少有一个机器人返回源节点时，遍历即告完成（参见问题7）。

问题2：群组结构是什么？

每个组由两类节点组成：必需与辅助节点：必需节点构成该组的核心，因为在最终分散中，它们最终必须由匹配颜色的机器人占据，并且辅助节点是仅为保持该组连通性而纳入的附加节点；它们通常位于连接成对必需节点的简单路径上。每个组指定一个必需节点作为其源节点。由一个组诱导的子图始终是连通的，并可表示为一棵以该组源节点为根的树。与一个组关联的守卫机器人维护必需节点与辅助节点的列表。一个关键约束是，每个必需节点唯一地仅属于一个组，而辅助节点则充当其他某个组的必需节点，从而确保各组对必需节点的覆盖互不重叠。此外，每个组维护其组间边的信息，这些边定义了组之间的父子关系（见 Q6）。在阶段 2 和阶段 3 期间，每个组可被抽象为单个超节点[1]。因此，[1]在术语略有滥用的情况下，我们互换使用术语 *supernode* 与 *group*，因为它们以不同解释表示同一对象。术语“组间边”保持不变。

将组（或超节点）视为顶点、将组间边视为它们之间的连接所形成的结构，构成一棵有向根树 $\mathcal{T}$。

问题3：一个组的大小是多少?

形式上，令 $c = \Theta(n/k)$，并假设 $k = \lceil n/c \rceil$ 个内存受限的机器人最初被放置在源点S。如果形成的不同组至多为 $\lceil n/c \rceil$ 个，且每组包含的必需节点数量介于 $2c$ 和 $4c$ 之间，并且恰好有一个机器人被分配为每组的守卫，那么可以保证完全探索该图： G 的每个节点都恰好属于一个组，且是其中的必需节点；机器人的集体内存足以存储有关所有 n 个节点的连通性信息。

如果将组大小限制为 $[c, 2c]$，那么最多可以形成 k 个不同的组；但是，没有机器人可以返回源点，因此无法完成 DFS（见问题8）。为避免这种情况，我们考虑大小介于 $2c$ 和 $4c$ 之间的组。当一个组达到 $2c$ 个必需节点时，该组即变为有效，从而确保有足够的机器人守卫所有节点并维持探索不变量；超过 $4c$ 则会触发拆分或重组。由于每组包含 $2c$ 至 $4c$ 个必需节点，最多只需 $2c - 1$ 个辅助节点即可维持其树结构。因此，每组总共至多包含 $6c$ 个节点。这保证守卫机器人在组内遍历只需 $O(c)$ 轮（见问题4）。

问题4：机器人到达一个节点后，如何确定该节点是否已经被探索过？（参见图1d）

每个组都是一棵有根树，最多可包含 $6c$ 个节点（最多包含 $4c$ 个必需节点和 $2c - 1$个辅助节点）。守卫机器人在其组内的节点之间振荡，因此最多需要$12c$ 轮次即可至少访问该组中的每个节点一次。因此，在某个节点处不是守卫机器人的任何机器人 r 都会等待 $12c$ 轮次。经过 $12c$ 轮次后，如果该机器人 r 遇到与节点 v 对应的守卫机器人（对于该守卫机器人而言的必需节点），或者 v 是 r的必需节点，那么它会得知 v 已经被访问过；否则， r 会确定自己是首次访问 v 。

问题5：组如何 在遍历过程中，组大小是缩小还是扩大？（参见图1d $\longrightarrow$ 图1 e)

一名守卫机器人 r 在深度优先搜索遍历期间扩展其组，直到该组包含 $4c$ 个必需节点。除特殊情况外，一旦组大小至少达到 $2c$ 并且 r 遇到一个已访问节点，它便停止扩展，因为该组被视为已足够。机器人 r 在（深度优先搜索的）回溯阶段遇到另一名守卫机器人 r'，且相遇于其组的一个必需节点 v 、同时其组大小小于 $2c$时，会收缩其组。如果 v 的所有子节点都已被探索，机器人 r 会减小其组大小，并传递以 v为根的子树中必需节点的信息。机器人 r' 根据自身的组大小，存储距离 v更远的节点的信息。

Q6：如何识别组间边？（参见图1b $\rightarrow$ 图1c）

组间边定义为一条边（u， v）$\in E$，其中两个必需节点 u 和 v 来自两个不同的组。这些边在深度优先搜索过程中自然地被识别：当机器人试图沿一条边移动到一个已被另一组占用的节点，而占用该节点的是最近成为该组守卫机器人的机器人（如Q1所述）时，它们将该边标记为组间边。每个组将这些边的端口编号记录为其本地数据的一部分。总体而言，这些组间边构成组之间的父子关系。

Q7：机器人如何检测深度优先搜索的结束？（参见图1g $\rightarrow$ 图1h）

遍历是否完成由源组（包含源节点的组）检测。源点处的守卫机器人维护一个计数器，记录通过每个关联端口完成探索的次数；每当探索一个关联端口时，计数器就加 1。由于每个端口在前进和返回时都恰好经过一次，因此当计数器达到 δ_S 时，守卫便判定深度优先搜索遍历已完成，其中 δ_S 是源节点的度。此时，所有组均已创建，所有组间边均已识别，且图中的所有节点均已访问，从而确保遍历完成。

问题8：机器人如何在深度优先搜索遍历期间保持同步?

在深度优先搜索遍历期间，机器人分为三类：（i）继续沿 DFS 路径移动的机器人；（ii）守卫机器人，其在所属组内振荡；（iii）最近的守卫机器人，其在所属组内振荡，并负责形成新的组。最近的守卫机器人还会与执行深度优先搜索遍历的机器人一起移动，因为它通过访问未探索节点来扩展自己的组（同时保持振荡）。为保持同步，时间被划分为中间轮次和移动轮次。在中间轮次中，机器人在当前节点等待，以确认该节点是否已被访问（参见问题4）。在此期间，守卫机器人（包括之前的和最近的守卫机器人）会振荡；如果它遇到另一台机器人，就会共享信息，并根据通信类型返回其组的源节点，或在当前节点等待。其他机器人则继续等待；如果遇到正在振荡的守卫机器人，它们还会执行计算。在移动轮次中，机器人继续进行深度优先搜索遍历，前往下一节点。这种结构化的划分确保所有机器人在每次访问节点期间都保持同步。

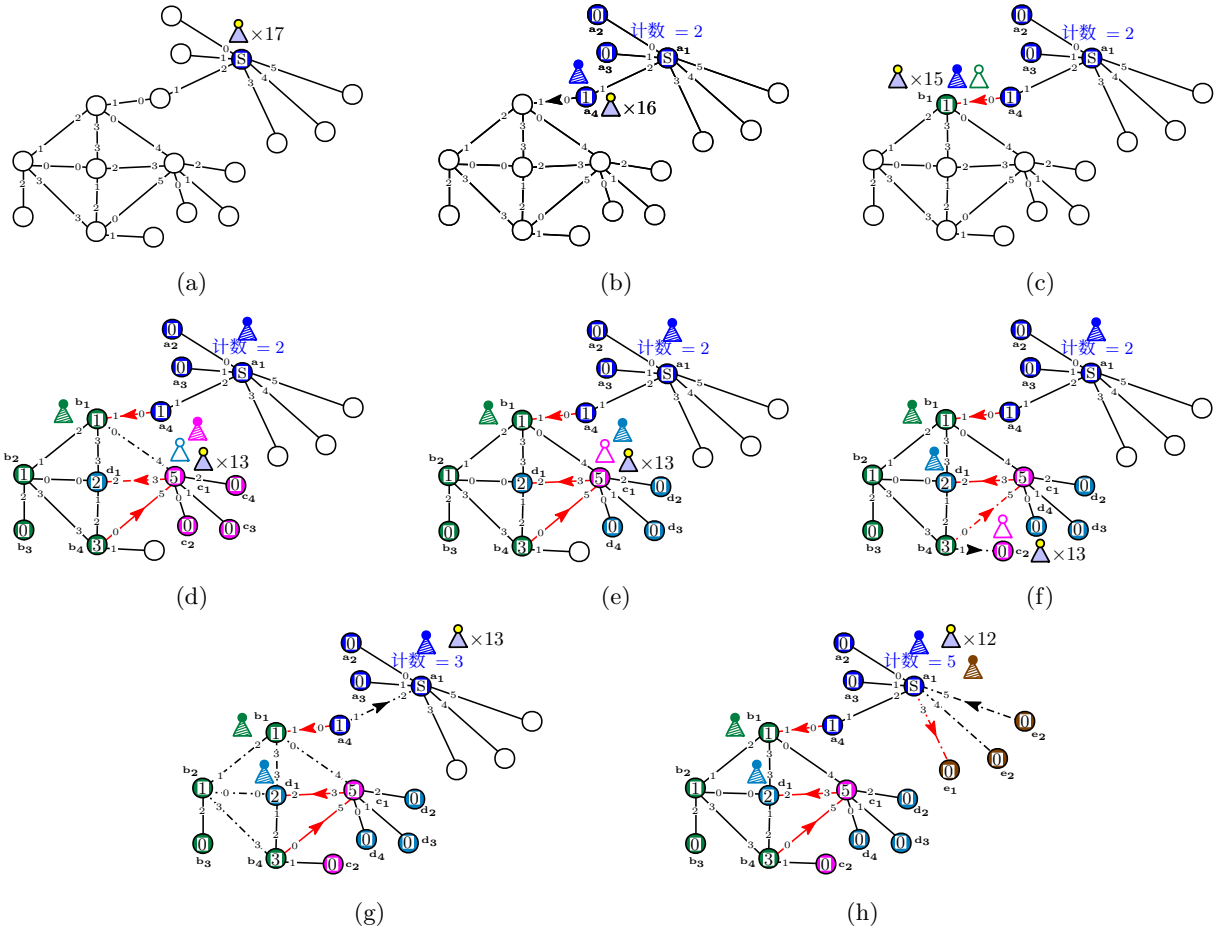

图 1：图 G中，$hollow$ 节点表示未访问节点，带有标签（$parent$ 端口）的 $colored$ 节点表示已访问节点。带有 $yellow$ 箭头的机器人表示活跃机器人，带有 $solid$ 箭头的机器人表示振荡机器人，带有$hollow$ 箭头的机器人表示领导者。(a) 初始配置，开始深度优先搜索遍历，组大小在 2 到 4 之间。(b) 遍历过程中，r_a 成为领导者并扩展其组（a_i 蓝色填充，$a_i \in F_{r_a}$，$i \leq 4$），黑色箭头表示出口端口（包括 r_a，它退出其组以存储组间边）。(c) 在 b_1处，选出领导者 r_b，红色实线表示组间边，方向从父组指向子组。(d) 组大小为 < 2的领导者 r_d 访问一个已访问节点 c_1，虚线表示遍历路径。(e)r_d 扩展其组，将 c_1处子树中的节点纳入其中；与此同时，r_c 缩小并在组大小为 < 2时转变为领导者。(f)r_c 纳入一个辅助节点 b_4 并扩展其组。(g) 活动机器人返回并遍历 S。(h) 深度优先搜索遍历结束。

3.1.2 阶段 2：完整的连通性信息收集

请注意，阶段 1 结束后，超节点集合与组间边共同构成一棵有向根树 $\mathcal{T}$，其根是包含源节点 S 的超节点。由于源超节点之外的机器人不知道阶段 1 何时结束，阶段 2 的执行方式有所不同。该阶段在 $\mathcal{T}$ 上采用经典深度优先搜索遍历，面临两个主要挑战：

(i) 将机器人收集到源点：位于 S 的守卫机器人 r 发起遍历。在前进阶段，r 在各超节点守卫机器人的引导下探索新的超节点。当深度优先搜索从超节点 W 回溯时，位于 W 的守卫机器人加入遍历，并沿回溯路径向其父节点移动。遍历完成以 W 为根的子树时，该子树中的所有机器人都已被收集，正前往 S。

(ii) 保持同步：同步方式与阶段 1 相同。具体而言，当 r 到达一个超节点时，它会在那里停留足够长的时间，以确保遇到位于该超节点的守卫机器人，从而使整棵树上的遍历保持同步。

3.1.3 阶段 3：内存高效分散

阶段 2 结束后，所有机器人在 S 集合，并掌握所有超节点及其中节点的完整连通性信息。利用这些信息，它们共同执行一次局部计算，随后执行一个两阶段过程，以确保每个机器人都到达其目标位置。

局部计算：每个机器人 r 确定其目标超节点，记为 dsg_gr （初始为$\textsc{Null}$），并识别该超节点中颜色与自身相匹配的节点。超节点W_1、W_2、.....根据其对应的组号进行唯一编号，其中 W_i 表示组 i。最低编号机器人 r 被分配到满足以下条件的最小索引 i： W_i 包含一个颜色相同且尚未分配的必需节点；随后，它设置 $r.dsg\_gr = W_i$。接着，按照机器人标识符递增的顺序，对所有机器人重复这一分配过程。每个机器人 r 还设置一个变量 $r.d = |R_i|$，其中 R_i 表示目标超节点为 W_i的机器人集合。机器人到超节点的分配完成后，机器人执行后续的局部计算。对于每个超节点，暂时最多分配一个机器人，以引导其他机器人前往各自的目标超节点。

请参阅图 2，以直观了解该计算。机器人的放置以递归方式进行，从叶超节点开始，逐步向 $\mathcal{T}$的源超节点 W_1 推进；其中，已分配给超节点 W_i 的机器人 r 存储其对应的 r.level，定义为距W_1的距离（边的数量）。每个机器人 r 还维护 3 个变量：$stmp\_gr$ （初始为 $\textsc{Null}$），用于存储位于从目标超节点 W_i 到 W_1 的路径上、有助于满足 W_i需求的超节点； tmp_gr（初始为 $\textsc{Null}$），用于存储同一路径上仅供遍历使用的超节点；以及 $ttmp\_gr$（初始为 $\textsc{Null}$），用于存储沿从 W_i 到 W_1 的路径上、有助于维持位于该路径上的节点需求的超节点。阶段 1 之后，任何具有 $r.tmp\_gr \neq \textsc{Null}$ 的机器人 r 都会返回 S。

设 W_η为一个具有子超节点 W_{η_1}、W_{η_2}、.....、W_{η_δ}的超节点。将 R_η定义为满足$r.ttmp_gr = W_{\eta_i}$ （对于某个 $1 \leq i \leq \delta$）的机器人集合；若不存在这样的机器人，则 $R_\eta = \emptyset$。此时会出现两种情形：

1. W_η 是目标超节点：至少有一个机器人 r 具有 $r.dsg\_gr = W_\eta$。其中最小 ID 的机器人 r_m设置 $r_m.stmp\_gr = W_\eta$ 和 $r_m.ctr = |R_\eta| + r_m.d$，其余机器人均设置 $r.ttmp\_gr = W_\eta$。 2. W_η 不是目标超节点：如果 $R_\eta \neq \emptyset$，具有$r.ttmp_gr = W_\eta$ 的机器人中，最小 ID 的机器人 r_m 设置 $r_m.ctr = |R_\eta| - 1$，其余机器人均设置 $r.ttmp\_gr = W_\eta$。否则，没有机器人会停留在 W_η。

初始放置后，超节点 W_η 可能没有具有 $r.stmp_gr = W_\eta$的机器人 r，但 W_η 位于一条或多条从目标超节点 W_i 通往 W_1的路径上。由于机器人数量多于超节点数量，因此 W_1有足够的机器人。对于每个这样的 W_η （按索引递增顺序），具有 $r_m.ttmp\_gr = W_1$ 的最低编号机器人 r_m 设置 $r_m.tmp\_gr = W_\eta$ 并重置 $r_m.ttmp\_gr = \textsc{Null}$。

完成这项计算后，每个机器人 r 都会掌握其当前超节点 W_i的信息，即 $r.stmp\_gr$ 或 $r.tmp_gr$中的非空值。这是通过与最初存储 W_i 数据的机器人交换信息实现的。此外，每个位于超节点 W_i 且其 $stmp\_gr$ 非空的机器人 r，都会维护一个数据结构 D，记录 W_i 及其子超节点所需的机器人数量。

具体来说， D 会跟踪每个子超节点所需的机器人数量和颜色，以及请求额外机器人的相应组编号（gr num）。

阶段：机器人遵循两阶段协作策略。在阶段 1 中，它们根据存储在 $stmp\_gr$ 和 tmp_gr 中的值执行深度优先搜索遍历。机器人停留在与其 $stmp\_gr$ 对应的节点上，而具有非空 tmp_gr 的机器人则返回 W_1。在阶段 2 中，机器人从根超节点出发，根据由每个节点上的机器人维护的数据结构 D 移动；该数据结构规定了每个子超节点所需的机器人数量和颜色。

3.2 算法

我们首先定义一些将在算法描述和算法分析中使用的关键术语和变量。

机器人类型：根据状态，机器人分为 3 种类型：活跃、领导者和振荡。最初，所有机器人都在源点 S 处处于活跃状态，并通过深度优先搜索探索图。活跃机器人会切换为领导者，以协助组建。在此过程中，领导者在其组内振荡。组建完成后，领导者转变为振荡状态，并继续在组内振荡。此外，振荡机器人可能在组压缩期间恢复为领导者。

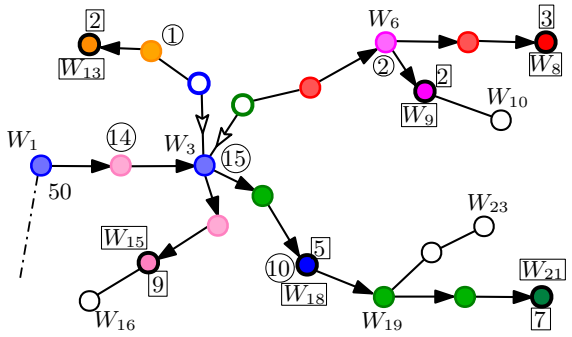

图2： W_1 是源超节点。标有 W 的方框表示目标超节点；其中的整数表示需求 d，圆圈则显示 ctr 值。空心箭头表示机器人在阶段 1 后返回父节点，实心箭头表示阶段 2 中从父节点移动到子节点。

群组结构：设 r 是一台机器人， F_r 是它构成的组。该组包含两类节点：必需节点是由 r 显式存储在集合 Q_r 中的关键节点；辅助节点位于连接必需节点的路径上，并保存在集合 H_r 中。由 F_r 诱导的子图构成一棵有根树，其根为该组的源节点， r 最初位于此处。

每个机器人 r 维护一个数据结构 F，用于存储有关 F_r 中节点的信息。函数 $\mathrm{SAVE}(r, v)$（算法3）维护节点 $v \in F_r$ 的信息。如果 v 尚不存在，则创建一条属性值为默认值的新记录；否则，更新现有记录。每条记录存储表3中提到的变量。

算法3： $\mathrm{SAVE}(r, v)$

// 设机器人 r 位于 v，且 $v \in F_r$

如果 $v \notin F[r]$，则

  在 $F[r]$ 中为 v 创建一条新记录

否则

  更新 $F[r]$ 中 v 的属性

变量	描述	初始值
pent(v)	机器人 r 进入节点 v 时经过的端口	-1
pext(v)	机器人 r 退出节点 v 所经过的端口	NULL
$parent(v)$	机器人 r 首次访问节点 v 所经过的端口,	NULL
$child[v]$	存储节点 v 的子端口列表	$\emptyset$
$required\_node(v)$	如果 v 是必需节点，则为是；否则为否。	NO
$helper\_node(v)$	如果 v 是辅助节点，则为是；否则为否。	NO
$node\_num(v)$	存储节点 v 在组中已分配的编号。每当添加一个必需节点时，该编号就会递增在 F_r 中添加必需节点时增加 1。	0
$Source(v)$	如果 v 是给定图 G 的源节点，则为是；否则为否。	NO
$gr\_source(v)$	如果 v 是其组的源节点，则为是；否则为否。	NO
$dist(v)$	存储 v 到组源节点的距离；递增在每个前向步骤中加 1，并在每个回退步骤中减 1。	0
$gr\_Parent(v)$	存储节点 v 上机器人 r 可通过其到达遍历所确定的父组的端口	NULL
$gr\_Child[v]$	存储节点 v 上机器人 r 可通过其到达的端口列表根据遍历与 F_r 相连的子组	NULL
下一_节点	存储要访问的下一节点 v （特别是由领导机器人存储），以及从当前节点 v 出发的相应出口端口，记录在 pext(v) 中	NULL

表3：机器人 r 在 F 中存储的、对应于节点 v 的变量

3.2.1 阶段 1：组队进行图探索

此阶段的目标是通过系统地组建组来探索整个图，使得 k 机器人能够访问图中的所有节点，并让部分机器人在探索完图后返回源点。我们将根据机器人的状态简要介绍该算法：活跃、领导者、振荡。

➤ 活动机器人的算法（算法4）：过程 $Active(r)$（算法4）控制活动机器人 r 在探索阶段的移动。每个机器人都维护一个局部变量 gr num（初始值为 0），用于记录机器人组建的组数；每当机器人发起组建新组时，该值便加 1。

A. 初始化（第 1–4 行）：第一轮中，每个活动机器人都将其 gr num 加 1（第 2 行）。在所有位于源节点 S 的机器人中，标识符最小的机器人留在 S 并将其状态更改为领导者（第 3 行）。该机器人充当第一个组的“守卫”。其余机器人都通过端口-$(p+1)$ mod δ_S 离开源节点（第 4 行），其中 p 是 pent[2]（初始值为 -1），而 δ_S 是源节点的度。因此，探索以深度优先遍历开始。

B. 中间轮次（第5-32行）这持续从第$(dT+2)$轮到第$((d+1)T)$轮，对于 $d=0,\ 1,\ldots,T=24c+2$。

设活动机器人 r_a 在第$(dT+1)$轮次以前进或回溯阶段到达节点 v 。到达某节点（记为 v）后，机器人 r_a 在 v 停留 $24c+2$ 轮次。在此期间，机器人根据在 v遇到的机器人类型评估其状态。为确定 v 是否先前已被访问， r_a 等待 $12c$ 轮次，以便以 v 为必需节点的机器人（振荡或领导者）到达 v （如问题5所述）。值得注意的是， r_a 在到达决策点之前已经等待了 $12c$ 轮次，即到第$((dT+2)\ +12c-1)$轮时它可以检测到任何相遇。以下三种情况是可能的。

(1.) 机器人 r_a 遇到一个振荡机器人 r_o ，其具有 $v \in Q_{r_o}$ （第 6–14 行）：在最初的 $12c$轮等待期间（第 $dT+2$ 轮至第 $dT+2+12c-1$ 轮），如果 r_a遇到一个具有 $v \in Q_{r_o}$的振荡机器人 r_o，这意味着 v 已经被访问，其信息由 r_o存储。存在两种可能的情况。

(i) r_o.源点(v) $=$ 是 （第 7–8 行和第 11–14 行）：当 r_a 回溯或到达 S 时，会出现这种情况。分两种情形：（i）当 r_a 在探索完整个图后（第 7–8 行）返回到 S ，以便[2]为简洁起见， $r.pext$（v）、 $r.pent$（v）分别记作 $pext$ 和 $pent$。

完成任务；它会记录深度优先搜索已完成。（ii）当 r_a 由于算法或图结构的原因，在探索完成之前返回（第 11–14 行）。随后，活动机器人继续探索图中剩余的部分。
(ii) r_o.源点(v) $=$ 否（第 9–14 行）：活动机器人继续探索。如果 r_a 处于前向阶段，则切换到回溯阶段，并设置 pext $=$ pent （第 9–10 行）。如果 r_a 处于回溯阶段，则更新 pext （第 11–12 行）。如果 pext 的值与 $parent$ (v) 不同，表示尚未探索 v 的所有邻居， r_a便切换回前向阶段（第 13–14 行）；否则，它会继续处于回溯阶段。

(2.) 机器人 r_a 没有遇到任何满足 $v \notin Q_{r_0}$ 的振荡机器人 r_o （第 16–32 行）：此时，根据领导者 r_l 是否位于 v ，分为两种情况：
(i) 领导者 r_l 位于 v （第 16–25 行）：根据领导者是否存储了关于 v 的信息以及活动机器人的阶段，会出现两种情况。如果活动机器人 r_a 处于前向阶段，则不执行任何操作，并从 v 回溯（第 16–18 行）。如果 r_a 处于前向阶段，且领导者 r_l 存储了关于v的信息，其中 v 是具有 $r_l.count = \delta_S - 1$的源节点，那么在探索完图并返回 S 后，r_a 设置 $r_a.DFS = \textsc{Done}$ （第 20–21 行）。否则，如果遍历尚未完成或 v 不是源点， r_a将保持空闲并存储 pext（第 23 行）。如果 v仍有未访问的端口，即 $(p+1) \bmod \delta_v \neq parent(v)$，$r_a$ 则切换到前向阶段（第 24–25 行）；否则，它会回溯。
(ii) v 处没有领导者（第 27–32 行）：当最后一名领导机器人完成组建并切换到振荡状态时，就会出现这种情况。具有最小 ID 的活动机器人 r_{a_m} 成为领导者，记录 v，并发起一个新组（第 28–30 行）。其他机器人更新 pext 并递增 gr num。如果所有端口 v 均已探索，机器人就切换到回溯状态（第 31–32 行）。

C. 等待与退出（第 33–37 行）：活动机器人在第 $((dT+1)+12c+1)$ 轮的 v 处进行计算。其余时间，即从第 $(dT+2)$ 轮到第 $((dT+2)+12c-1)$ 轮，以及从第 $((dT+2)+12c)$ 轮到第 $((d+1)T)$ 轮，机器人都在 v 处等待（第 37 行）。否则，在节点 v处停留结束时，具体来说，在第 $((d+1)T+1)$ 轮，机器人通过 pext 中记录的出口端口离开（第 36 行）。D. 遍历结束（第 7–8 行、第 20–21 行和第 34 行）：负责源节点 S 的机器人会记录与活动机器人的相遇次数。当计数器达到 δ_S，即 S的度时， S 的所有端口都已遍历，所有返回的机器人也都已重新访问 S。此时，深度优先搜索遍历完成，探索结束（$r.DFS = \textsc{Done}$）。这一结束条件在第 7–8 行和第 20–21 行中有明确体现。机器人在该节点等待，并在第 $((d+1)T+1)$ 轮将状态更改为 $passive$ ，然后运行下一阶段（第 34 行）。

➤ 领导机器人算法：当机器人 r 成为领导者后，它执行 $Leader(r)$（算法5）。它会增加组编号（r.组编号 $= r$.组编号 $+1$），维护 $r.F$、 Q_r、 H_r和计数器 $count$。对于添加到 Q_r的每个节点 v，它都会增加组大小。领导者 r 通过与活动机器人相同的端口（pext）扩展该组，而活动机器人则继续探索未访问节点，以支持组建。

A. 初始化（第1–3行）：在第一轮中，具有最小 ID 的机器人 r_l 成为领导者，并将源节点 S 存储在 $r_l.F$ 中（第2行）。它将组大小增加1（第3行），并通过端口-(pent $+1) \bmod \delta_S$退出 S，存储下一待访问节点的信息；其中，pent 是活动机器人 r_a 进入 S 时经过的端口（第3行）。

B. 中间轮次（第4–48行）：这些轮次从第$(dT+2)$轮持续到第$((d+1)T)$轮，其中 d =取0、1、.....、 $T = 24c+2$。它分为三个部分。从第$(dT+2)$轮到第$(dT+2)+12c$轮，领导机器人 r_l 在其组内振荡，并处理存储在变量“下一节点”中的即将访问的节点。在第$(dT+1)+12c+1$轮，领导机器人执行计算并扩展其组。从第$(dT+1)+12c+2$轮到第$((d+1)T+1)$轮，领导机器人使用存储在 $r.F$中的信息，振荡返回其组源节点。

如果遇到活动机器人，它就会停下来并在那里等待（第 4–6 行）。
设领导机器人 r_l 在振荡期间的第（$dT+2$）轮至第（（$d+1$）T）轮于节点 v 遇到一名活动机器人（第 4–6 行）。机器人 r_l在 v处等待。在第（（$dT+1$）$+12c+1$）轮，于 v， v处出现以下情况：

(1.) r_l 不会遇到带有 $v \in Q_{r_o}$ 的振荡机器人 r_o （第 7-26 行）：根据活动机器人的阶段，有两种情况。如果活动机器人处于回溯阶段，则有三种可能的情况：(i) 如果 v 是源节点，且具有 $v \in Q_{r_l}$ 和 $r_l.count = \delta_v - 1$，即 r_l 已完成深度优先搜索遍历

算法4: *Active*(*r*)

```
   // R[v] = 个机器人位于节点 v
   // 初始时 r.p = −1，其中 p = pent
 1 若 round == 1 则
 2 |  r.gr_num = r.gr_num + 1
 3 |  若 r is the minimum ID robot 则 r.state = leader ;
 4 |  否则通过 pext = (p + 1) mod δ_S 退出;
 5 若 round == (dT + 1) + 12c + 1 则
 6 |  若 ∃ oscillating robot r_o ∈ R[v] with v ∈ Q_{r_o} then
 7 |  |  若 r_o.Source(v) = YES in r_o.F and r_o.count = δ_v − 1 则
 8 |  |  |  r.DFS = Done
 9 |  |  否则若 r.phase = forward 则
10 |  |  |  r.phase = backtrack 且 pext = p
11 |  |  否则若 r.phase = backtrack 则
12 |  |  |  pext = (p + 1) 模 δ_v
13 |  |  |  如果 (p + 1) 模 δ_v ≠ parent(v) 则
14 |  |  |  |  r.phase = forward
15 |  else
16 |  |  如果 ∃ leader robot r_l ∈ R[v] with v ∈ Q_{r_l} then
17 |  |  |  如果 r.phase = forward 则
18 |  |  |  |  r.phase = backtrack
19 |  |  |  else
20 |  |  |  |  如果 r_l.Source(v) = YES in r_l.F and r_l.count = δ_v − 1 则
21 |  |  |  |  |  r.DFS = Done
22 |  |  |  |  else
23 |  |  |  |  |  pext = (p + 1) 模 δ_v
24 |  |  |  |  |  如果 (p + 1) 模 δ_v ≠ parent(v)，则
25 |  |  |  |  |  |  r.phase = forward
26 |  |  else
27 |  |  |  r.gr_num = r.gr_num + 1
28 |  |  |  如果 r ∈ R[v] is the minimum ID robot，则
29 |  |  |  |  Save(r, v) 并存储 gr_Parent(v) = p′，其中 p′ 是 pent 的 oscillating 机器人，带有
   |  |  |  |    recent = true 和 r.state = leader
30 |  |  |  pext = (p + 1) 模 δ_v
31 |  |  |  如果 (p + 1) 模 δ_v = parent(v)，则
32 |  |  |  |  r.phase = backtrack
33 如果 round == (d + 1)T + 1 则
34 |  如果 r.DFS = Done 则
35 |  |  r.state = passive 并运行阶段 2
36 |  否则通过 pext 退出;
37 否则在当前节点等待;
```

并返回 S，它会设置 $r_l.DFS = \text{Done}$，并将其状态更改为振荡（第 10-11 行）。(ii) 否则，如果 v 是源节点且 $r_l.count \neq \delta_v - 1$（第 12-13 行），$r_l$ 会递增 $count$，并记录待访问的下一节点信息，以及从当前节点离开的出口端口 $v_\circ$。(iii) 如果 v 不是源节点且 $v \in Q_r$（第 14-15 行），r_l 会记录待访问的下一节点信息，以及从当前节点离开的出口端口 $v_\circ$

如果活动机器人处于前向阶段，则有两种情况。如果 $v \in Q_{r_l}$（第 17 行），那么 r_l 不执行任何操作，因为活动机器人正在从 v回溯。如果 $v \notin Q_{r_l}$（第 19–26 行），r_l 会将新访问的节点 v 添加到 F_r中。现在，如果 gr 大小为 $< 4c$，r_l 会继续进行组扩展；否则，当 gr 大小为 $= 4c$时，它会切换到振荡状态，并根据活动机器人处于前向阶段还是回溯阶段，更新 pext（以存储父子关系）。

(2.) r_l 遇到一个振荡机器人 r_o，其带有 $v \in Q_{r_o}$（第 27–45 行）：当 v 已被访问时会出现这种情况。令 g 表示 r_l当前的组大小，g' 表示扩展后的更新大小。如果 v是 S，且具有 $r_o.count = \delta_v - 1$（第 28–29 行），r_l 满足终止条件，切换为振荡状态，并设置

$recent = false$，并在第 $((d+1)T + 12c + 1)$ 轮返回其组。如果 $2c \le g < 4c$（第 28–29 行），r_l 拥有足够的必需节点，则切换为振荡状态，设置 $recent = false$，记录出口端口，并设置 $r.move = 1$。如果是 $g < 2c$（第 31–48 行），则根据活动机器人的阶段采取相应操作：在前向阶段（第 31–32 行），r_l 将下一个待访问节点及出口端口的信息存储为 p；而在回溯阶段，如果存在未访问子节点（第 34–35 行），则探索该子节点（将信息存储在下一节点中，即下一个要访问的节点），并将 v 记录为辅助节点。否则（第 37–45 行），使用来自 r_o的后代节点信息扩展组，并维护 $g' \le 4c$。当 $2c \le g' \le 4c$时，它切换为振荡状态；否则，它将 v 添加到 $r_l.F$ 中，并继续扩展，直到 $g' \ge 2c$。

从第$((dT + 2) + 12c + 1)$轮到第$((d+1)T)$轮（第 46–47 行），r_l 在其组源节点之间来回振荡。

C. 等待与退出（第 48–49 行）：在第$((d+1)T + 1)$轮，如果深度优先搜索已完成，则开始下一阶段。

➤ 振荡机器人的算法：处于振荡状态的机器人 r 遵循 $Oscillating(r)$（算法 6）。其行为取决于变量 $recent$。如果 $recent = true$，机器人 r 与活动机器人一起移动，以收集有关组间边的信息；如果 $recent = false$，则在其当前组内振荡。

A. 初始化（第 1 行）：在第 $(dT + 1)$ 轮，机器人不执行任何操作。

B. 中间轮次（第 2–35 行）：分为三个部分。从第 $(dT + 2)$ 轮到第 $((dT + 2) + 12c - 1)$ 轮，如果 $recent = false$，振荡机器人 r_o 在其组内振荡，遇到活动机器人或领导机器人时暂停；如果 $recent = true$，则停留在其当前节点。在第 $((dT + 2) + 12c)$ 轮，它通过与共处机器人通信来进行计算。最后，从第 $((dT + 2) + 12c + 1)$ 轮到第 $((d+1)T + 1)$ 轮，振荡机器人继续振荡并返回其组源节点。

设振荡机器人 r_o（具有 $recent = false$）在第 $dT + 2$ 轮至第（$dT + 2 + 12c - 1$）轮期间，于节点 v 遇到一台 fl 机器人（第 2–6 行）。机器人 r_o 在当前节点 v 等待，而在第（$dT + 2$）轮 $+12c$（第 8–32 行）中，会出现以下情况：

- $recent = false, move = 0$：如果 v 是源节点且 $r_o.count = \delta_v - 1$（第 10–11 行），即机器人探索完整个图后返回，则深度优先搜索终止，且 r_o 设置 $DFS = \text{DONE}$。如果 $r_o.count \neq \delta_v - 1$，即由于图的结构，机器人在探索端口 p（进入的端口）后返回源节点，则 r_o 增加 $count$（第 13 行）。

  如果一台领导机器人 r_l 位于 v，则 r_o 分享其组的信息。根据 r_l 的计算，如果它利用 r_o 存储的节点扩展其组，则 r_o 更新其组大小 g，并相应地更新其状态（第 17–23 行）。如果 $g < 2c$，则 r_o 成为领导机器人，并存储下一待访问节点的信息（第 17–18 行）。如果组大小满足 $2c \le g \le 4c$，则 r_o 保持振荡状态，并与一台活动机器人一起移动到其组以外的节点，以记录组之间的父子关系（第 20–22 行）。如果一台领导机器人位于 v，则 r_o 在前向阶段与一台活动机器人一起移动；若该活动机器人处于回溯阶段，则原地停留（第 25–26 行）。
- $recent = true$（第 27-30 行）：如果节点 v 处没有振荡机器人，且 v 被标记为必需节点，则 r_o 记录新组（组间边）的入口端口和出口端口，并设置 $recent = false$。
- $recent = false, \ move = 1$（第 31-32 行）：机器人 r_o 经由端口 pext 退出。

从第（$dT + 2$）$+12c + 1$轮到第（$d+1$）T轮（第 33-35 行），如果满足 $recent = false, \ r_o$就振荡返回其组源节点；否则，它留在当前节点。

C. 等待与退出（第 36-38 行）：机器人 r_o 与 $recent = true$ 退出当前节点以记录组间边（第 37 行）。一旦 $r_o.DFS = \text{DONE}$，机器人进入算法的下一阶段（第 38 行）。

3.2.2 阶段 2：完成连通性信息收集

算法从位于源节点 S 的振荡机器人 r 开始（或等价地，位于源超节点）。机器人 r 转换到 $osci\_head$ 状态，并在组上启动深度优先搜索遍历。由于每个振荡机器人都知道其父节点和子组，跨组遍历通过组间边实现。

算法5: $Leader(r)$

```
// p' = (p_a + 1) mod δ_v, 其中 p_a = r_a.pent
如果 round == 1 and r ∈ R[v] is the minimum ID robot 则
    r.state = leader, 且 Save(r, S)
    r.gr_size = gr_size + 1, r.next_node = v, pext(S) = p'
如果 round = ((dT + 1), (dT + 1) + 12c] 则
    在其组节点和节点 v之间振荡
    如果 encounters an active robot r_a 则等待;
如果 round == (dT + 1) + 12c + 1 则
    如果 ∄ oscillating robot r_o ∈ R[v] with v ∈ Q_{r_o}  then
        如果 active robot r_a ∈ R[v] is in backtrack phase 则
            如果 r.Source(v) = YES in r.F and r.count = δ_v − 1 with v ∈ Q_r 则
                r.count = r.count + 1, r.state = oscillating, r.DFS = Done
            否则, 如果 r.Source(v) = YES in r.F and r.count ≠ δ_v − 1 with v ∈ Q_r 则
                r.count = r.count + 1, r.next_node = v' 并且 pext(v) = p'
            否则, 如果 v ∈ Q_r 则
                r.next_node = v' 并且 pext(v) = p'
        否则, 如果 active robot r_a ∈ R[v] in forward phase 则
            如果 v ∈ Q_r 则不执行任何操作;
            else
                Save(r, v), 以及 r.gr_size = r.gr_size + 1
                如果 r.gr_size < 2c 则
                    r.next_node = v' 以及 pent(v') = p'
                else
                    r.state = oscillating
                    如果 p' ≠ parent(v), 则
                        recent = true 且 pext = p'
                    否则 recent = false ;
    如果 ∃ oscillating robot r_o ∈ R[v] with v ∈ Q_{r_o}  then
        如果 r_o.Source = YES and r_o.count = δ_v − 1 or r.gr_size ∈ [2c, 4c) 则
            r.move = 1, r.recent = false, r.state = oscillating 且 pext = p_a
        else
            如果 active robot r_a ∈ R[v] in forward phase 则
                r.next_node = v' 和 pent(v) = p_a
            else
                如果 p' ≠ parent(v), 则
                    r.next_node = v' 和 pext(v) = p'
                else
                    r 扩展其组
                    如果 after expand r.gr_size = [2c, 4c] , 则
                        r.state = oscillating, r.recent = false
                    else
                        Save(r, v), r.gr_size = r.gr_size + 1
                        如果 r.gr_size < 2c 那么
                            r.next_node = v', pext（v） = p'
                        else
                            r.state = oscillating 和 r.recent = false
如果 round = （（dT + 1） + 12c + 1, （d + 1） T + 1）那么
    振荡并返回到 gr_source 节点
若 round == (d + 1)T + 1 则
    若 r.DFS = Done 则运行阶段 2;
```

该过程以同步轮次进行。在探索期间，振荡机器人在其组内振荡：它在（$dT+2$）至（$dT+2$）+（$12c-1$）期间从其源节点移动到另一个指定节点，暂停一轮，然后在（$dT+2$）+（$12c$）至（$d+1$）T期间返回源节点。

类似地，$osci\_head\ r$ 机器人在 F_r 期间沿组间边从其源节点移动到 $F_{r'}$ 中的一个子组，并在（$dT+2$）至（$dT+2$）+（$12c-1$）期间到达节点$v \in F_{r'}$。到达后，r 停留在 v，直到轮次（$d+1$）T，以确保 r' 的振荡机器人在返回其源节点时遇到 $F_{r'}$ 的 r。在轮次（$d+1$）$T+1$，r 将 $F_{r'}$ 的信息记录在临时数据结构 F'_r中，随后继续进行深度优先搜索。

在下一个时间区间内，从 $(d'T+2)$ 到 $(d'T+2)+(12c-1)$，在 $d'=d+1$, r 遍历$F_{r'}$中的节点，通过一条组间边退出，并前往子组 $F_{r''}$。等待至 $(d'+1)T$后，它会根据 $F_{r''}$ 的结构信息更新记录，并继续遍历。

算法 6: $Oscillating(r)$

```
1 如果 round == dT + 1 ，则在当前节点等待;
2 如果 round = ((dT + 1), (dT + 1) + 12c] 则
3 |  如果 recent = false ，则
4 |  |  在其组内往返移动
5 |  |  如果 ∃ active r_a or leader r_l ∈ R[v] 则
6 |  |  |  在当前节点等待
7 |  否则在当前节点等待;
8 如果 round == (dT + 1) + 12c + 1 则
9 |  如果 r.recent = false, r.move = 0 则
10 |  |  如果 r.Source(v) = YES in r.F, r.count = δ_v − 1 with v ∈ Q_r 则
11 |  |  |  r.count = r.count + 1, r.DFS = DONE
12 |  |  else
13 |  |  |  如果 r.Source = YES in r.F, r.count ≠ δ_v − 1 with v ∈ Q_r 则
14 |  |  |  |  r.count = r.count + 1
15 |  |  |  如果 ∃ leader robot r_l ∈ R[v] 则
16 |  |  |  |  共享必要信息
17 |  |  |  |  如果 r.gr_size < 2c 则
18 |  |  |  |  |  r.state = leader 且 pext = p'
19 |  |  |  |  else
20 |  |  |  |  |  如果 v ∈ F_r then
21 |  |  |  |  |  |  如果 p' ≠ parent(v)，则
22 |  |  |  |  |  |  |  pext = p' 且 r.recent = true
23 |  |  |  |  |  否则 r.move = 1, r.pext = parent(v);
24 |  |  |  else
25 |  |  |  |  如果 p' ≠ parent(v)，则
26 |  |  |  |  |  pext = p' 且 r.recent = true
27 |  否则，如果 r.recent = true ，则
28 |  |  如果 ∄ oscillating robot r_o ∈ R[v] with v ∈ Q_r 0 then
29 |  |  |  r.gr_Child[v'] = r.portext(v') 在 r.F 中，其中 v' 是先前经过的节点
30 |  |  r.move = 1、 r.recent = false 和 pext = p
31 |  否则，如果 r.recent = false and r.move = 1 ，则
32 |  |  r.move = 0 并通过 pext 退出
33 如果 round = ((dT + 1) + 12c + 1,(d + 1)T + 1) 则
34 |  如果 recent = false ，则往返振荡并返回到 gr_source 节点;
35 |  否则停留在当前节点;
36 如果 round == (d + 1)T + 1 则
37 |  如果 recent = true ，则通过 pext 退出;
38 |  否则，如果 r.DFS = DONE ，则运行阶段 2;
```

如果 r 到达叶组 F_{r^l}，它会与守卫 r^l一起回溯到父组，并将$r^l.gather$ = YES 更新为是（初始设置为否）。回溯过程中，r 会系统地探索所有尚未探索的子组。当一个节点的所有子组都已访问后，遍历便继续向上进行，同时带上标记 $gather$ 设为是的振荡机器人。这一基于深度优先搜索的过程确保收集每个组的信息

并将其传播。最终， $osci\_head$ 会汇集G的完整结构信息。

3.2.3 阶段 3：内存高效分散

阶段 2 之后，机器人 r 在 S处进行局部计算，设置变量 dsg_gr、 d、 $stmp\_gr$、 tmp_gr、ctr、 $level$以及数据结构 D （第 3.1.3 节）。机器人现在采用两步协作策略。

阶段 1：机器人根据已分配的组进行分散，这些组存储在变量 tmp_gr 和 $stmp_gr$中；机器人按照深度优先搜索进行操作，具体如第 3.2.2 节中针对 $osci\_head$ 机器人的描述。 tmp_gr 或 $stmp\_gr$ 中任一变量的值非NULL的机器人会启动深度优先搜索，并在协同引导下遍历各组。每个机器人维护一个计数器变量 $round$ （初始值为 0），每经过一轮就将其加 1。每个机器人在每个组停留 $12c$ 轮。

从超节点 F^i进行反向遍历时，其中 F^i 表示组编号为 i的组；一旦所有子组都已访问，具有 $r.stmp\_gr = F^i$ 的机器人 r 就留在该节点，而其余机器人则回溯并继续遍历。

阶段 2：当 $round = 12c \cdot k$ 时，机器人启动阶段 2，随后重置 $round = 0$，标志着阶段 1 完成。机器人 r，其 $r.level$，会在 $round = 12c \cdot l_{i-1}$时执行计算；它 l_i。组 F^j 中的机器人 r（其中 $r'.stmp\_gr = F^j$）根据存储在 $r'.D$ 中的数据执行计算。具体而言， r 会移动到子组 $F^{j'}$ 以满足待处理需求，并由机器人 r' 引导；或者留在当前组 F^i。

经过 $12c \cdot l_d$ 轮次后，其中 l_d 是 $\mathcal{T}$ 的直径，每个节点至多容纳一个同色机器人，并进入空闲状态。

3.3 算法分析

引理1. 设 F_r 为机器人 r 形成的组。设 Q_r 和 E_{Q_r} 分别为 F_r 的必需节点集和需求边集，设 H_r 和 E_{H_r} 分别为 F_r 的辅助节点集和辅助边集。定义 $U_r = Q_r \cup H_r$ 为 F_r 中所有节点构成的集合，并设 $E_r = E_{Q_r} \cup E_{H_r}$ 为 F_r 中所有边构成的集合，其中，对于任意两个顶点 v_i、 $v_j \in U_r$， $E_{Q_r} = \{(v_i, v_j) \mid v_i \in Q_r$ 或 $v_j \in Q_r\}$，且 $E_{H_r} = \{(v_i, v_j) \mid v_i$、 $v_j \in H_r\}$。则组 F_r 满足以下性质：
1. (U_r, E_r) 构成一棵以组源节点 $v \in Q_r$ 为根的树，即 r.gr source(v) = *YES*。2. F_r 中的节点数满足 $|U_r| \leq 6c$。3. 设 v 是 U_r 的组源节点（即 r.gr source(v) = *YES*），设 $u \in Q_r$ 是距离 v 最远的必需节点。从 v 出发，机器人 r 至少访问 U_r 中的每个节点一次，并在不超过 $12c$ 轮次内到达 u。

证明。

1. 我们通过证明 (U_r, E_r) 无环且连通，来说明它构成一棵树。令机器人 r 在节点 v 成为领导者。根据算法，r 通过将 v 纳入为源节点来初始化其组 F_r。
   无环性。一个顶点只会被添加到 U_r 一次：要么是在前向遍历中首次访问该顶点时（作为必需节点），要么是在回溯阶段，当机器人 r 扩展其组并将另一个组中的顶点纳入为辅助节点或必需节点时。在执行过程中，机器人 r 可能会扩展或缩小其组。

   在前向遍历期间，如果机器人 r 遇到一个已经访问过的顶点 $u \in U_r$（即 $u \in Q_r$ 或 $u \in H_r$），活跃机器人会检测到这一情况，并在规定的等待时间后回溯，而不添加任何边。因此，在前向阶段，不会在两个已访问过的顶点之间添加边。

   在回溯阶段，机器人 r 到达某个机器人 r'的顶点 $v \in Q_{r'}$ 时，可以扩展其组。扩展沿着一条简单路径进行，并且对于每个新加入的顶点，只会添加一条边，将其连接到路径上的前驱，并将该边加入 E_r。假设这种扩展会形成一个环，这就产生了矛盾。那么，某个新加入的顶点 v' 必定与一个在扩展前就已存在的顶点 $u \in U_r$ 相连，

这意味着机器人 r 遍历了从 v'到 u的边，并与 v' 建立了父子关系，其中 v' 是父节点。然而，在回溯阶段，只允许从子节点向父节点遍历；因此，机器人 r 必定是从 u 回溯到 v'，这就产生了矛盾。因此，扩展过程中不会形成环。

最后，当机器人 r 缩小其组时，会移除顶点及其关联边。由于删除边不会产生环，因此无环性得以保持。

因此，在所有情况下，机器人 r 维护的子图 (U_r, E_r) 都是无环的。
连通性。设 F_r 中有两个顶点 $u, v \in U_r$ 。如果 u 和 v 之间的路径完全由属于 U_r 的顶点构成，那么存在一条仅使用 E_r 中边的 u 和 v 之间的路径，因为每当r 从一个已访问的顶点 $u \in U_r$发现一个新顶点 $v \in U_r$ 时，恰好会向 E_r 中添加一条边 (u, v)。因此， u 和 v 在 F_r 中连通。

反设存在一个位于 u 和 v 之间路径上的顶点 $w \notin F_r$ ，从而导出矛盾。根据算法，如果机器人 r 在前向阶段从 u 扩展其组时，遇到一个已访问的顶点 $w \notin U_r$ ，那么 r 会立即回溯。这意味着机器人无法继续朝 v 前进，这与 $v \in U_r$的假设矛盾。

或者，如果 r 在回溯阶段扩展其组并遇到这样的顶点 w，那么算法会确保将从 w 到 v 的路径上的顶点添加为辅助节点或必需节点。因此，存在一条完全位于 F_r内、从 u 到 v 的路径。

最后，当机器人 r 收缩其组时，连通性得以保持：收缩操作之后，顶点 v 要么仍属于 F_r，要么与其依赖子树一并移除，从而确保 F_r 中剩余的顶点保持连通。

此外，由于探索过程保持无环性，这条路径是唯一的。因此，由 F_r 诱导的子图是连通的。

由于 (U_r, E_r) 既无环又连通，因此它构成一棵以源节点v为根的树。

2. 在组 F_r中，必需节点的数量满足 $2c \le |Q_r| \le 4c$。因此， $|Q_r| \le 4c$，接下来只需证明辅助节点的数量满足 $|H_r| \le 2c$。
根据算法，当机器人 r 遇到一个已访问且属于另一机器人的组的节点时，会在 F_r 中（如有需要）引入辅助节点。假设机器人 r的组大小为 $g < 2c$，到达已访问节点 v。如果 $g > 2c$，那么 r 的规模足够大，它会直接返回自己的组，而不会将 v 添加为辅助节点或必需节点。假设机器人 r 正在回溯至节点 v。如果 r 在前向阶段访问 v （其中 $v \notin Q_r$ 或 $v \in Q_r$），那么无论其规模如何，它都不会执行任何操作，而是按算法规定从进入时经过的端口退出v 。设 r_o 为振荡机器人，它存储与 v 对应的信息，并位于 v。

当 r 到达 v时，有两种可能：

(i) 如果 v 至少有一个未访问子节点 v_1，那么 r 会在前向阶段继续前往 v_1。在这种情况下，v 会作为辅助节点纳入 H_r 。在 v_1处， r 会存储 v_1的信息，组大小增加到 $g + 1$。如果 v_1 没有未访问子节点，那么根据图探索算法， r 最终会回溯至 v （因为 r 会在前向阶段访问）。当 v的所有子节点都探索完毕后， r 会将以 v为根的子树信息补充到 F_r 中。

(ii) 如果 r 到达时， v 的所有子节点都已被访问，那么 r 会立即从 r_o 接收子树信息，并将其整合到 F_r 中。

在这两种情况下，关键在于， r 从 v 的子树中（经由 r_o）接收到的信息量上限为 $4c$ 个节点。对机器人 r 而言，最坏的情况出现在必需节点距离 v 更远时（即最多相隔 $4c$ 跳），此时 r 必须遍历更多的中间节点（辅助节点），才能到达相应的节点，而这些节点的信息由 r_o共享。设与 r_o对应的 v 的子树（包括 v）的大小为 $4c$ （最大规模）。因此，与 r 相关的节点距离 v最多为 $4c$ ，在这一极端情况下， r

被迫包含比必需节点更多的辅助节点，因为通往每个必需节点的路径可能会经过额外的中间节点。

现在，如果组大小为 $g < 2c$，组形成过程将继续。具体而言，当$g = 2c - 1$时，机器人 r 尝试从 v 的子树中存储尽可能少的额外必需节点，使必需节点远离 v。在这种情况下，机器人 r_o 贡献了 $2c+1$ 个节点的信息。因此， r 恰好存储了 $2c+1$ 个节点，其余 $2c-1$ 个节点则充当辅助节点，引导 r 到达必需节点。因此， F_r 至多包含 $2c-1$ 个辅助节点。

因此可证明 $|U_r| = |Q_r| + |H_r| \leq 6c$。

3. 令 v 为 U_r 的组源节点，并令 $u \in Q_r$ 为距离 v最远的必需节点。根据第 (ii) 部分，组 F_r 至多包含 $4c$ 个必需节点和 $2c-1$ 个辅助节点。因此， F_r 中的节点总数严格小于 $6c$，且 F_r 构成一棵树。

在深度优先搜索遍历期间，机器人 r 可能因前向阶段和回溯阶段而多次访问每个节点。具体而言，每个必需节点至少被访问一次（前向阶段），至多被访问两次（前向阶段以及可能的回溯阶段），每个辅助节点至少被访问一次（作为连接节点时），至多被访问两次（前向阶段和回溯阶段）。

因此，在最坏情况下，遍历所有必需节点至多需要 $2 \cdot 4c = 8c$ 轮次，遍历所有辅助节点至多需要 $2 \cdot (2c-1) < 4c$ 轮次。将这些上界相加，访问 F_r 中所有节点所需的总轮次数至多为 $8c + 4c = 12c$。

因此，从 v出发，机器人 r 至少访问 U_r 的每个节点一次，并在至多 $12c$ 轮次内到达最远的必需节点 u 。

□

引理 2. 设 F_r 为机器人 r （处于领导者状态或振荡状态）所形成的组，并设 $S \in F_r$ 为给定图 G的源节点。当由 r 维护的变量 $count$ 满足 $r.count = \delta_S$时，图探索过程结束，其中 δ_S 是节点 S的度。

证明。令 F_r 表示由机器人 r组成的组，并令 $S \in F_r$ 为图 G的源节点。机器人维护变量 $count$ ，用于记录 S 中已完全探索的关联边数量。

根据探索算法，每当活动机器人在回溯阶段通过端口 p 离开源节点 S ，并随后通过同一端口返回 S 时，机器人 r 就会递增 $count$ ，这表示相应的关联边已完全探索。由于 S的度为 δ_S，因此恰好有 δ_S 条这样的关联边。

因此，当且仅当 $count = \delta_S$时，探索才会终止；这保证了 S 的所有关联边均已探索，且机器人 r 已返回源节点，从而完成对该图的探索。

□

如前所述， F_r 表示由机器人 r组成的组。为便于分析，我们根据组编号（gr num）对组进行分类。令 $W_1, W_2, \ldots, W_b$ 为满足以下条件的组的集合： W_i对应于机器人 r 的组，且 $r.gr\_num = i$。为方便起见，我们也将每个组称为一个超节点。因此， W_i 表示超节点（或等价地，组） F_r，其中 r 是位于 F_r 的源节点 v 处的机器人（即，对于某个 $v \in F_r$， r.gr source(v) $=$ 是），且 r.gr num $= i$。类似地，我们用 Q_i和 H_i 分别表示超节点 W_i 的必需节点集和辅助节点集。令 $\mathcal{W} = \{W_1, W_2, \ldots, W_b\}$ 为所有超节点的集合。显然， W_1 是遍历过程中创建的第一个组，这意味着对于某个机器人 r ， $S \in Q_1 \subseteq W_1$ ，或等价地， $S \in F_r$ ，且 r.Source(S) $=$ 是。从现在起，我们将用“超节点”代替“组”。下面我们证明有关这些组的一些性质。

引理 3. 在图探索过程结束时，以下性质对组/超节点集合成立：

1. 与处于领导者或振荡状态的机器人 r 对应的源节点 $S \in W_1$ 至少包含两个机器人。2. 对于图 G 的每个节点 $v \in V$，存在唯一的超节点 W_i ，使得 v 是 W_i 的必需节点。等价地，必需节点集合构成 V 的一个划分，即对于所有 $i \neq j$，有 $Q_i \cap Q_j = \emptyset$ ，且有 $Q_1 \cup Q_2 \cup \cdots \cup Q_b = V$，其中 Q_i 表示超节点 W_i 的必需节点集合。

3. 超节点的数量满足 $|\mathcal{W}| \leq k/2$，其中 k 是机器人的总数。（等价地， b 满足 $b \leq k/2$。）

4. 令 $\mathcal{E}$ 为我们的算法在集合 $\mathcal{W}$ 中识别出的组间边集合。那么，结构 $\mathcal{T} = (\mathcal{W}, \mathcal{E})$ 构成一棵以 W_1 为根的定向树，其中 W_1 是包含机器人 r 的超节点，且对于 $S \in W_1$，r.Source(S) = *YES*。

证明。

1. 我们将证明，在探索过程结束时，源节点至少包含两台机器人。考虑一个由 n 个节点组成的图 G。在探索过程中，每个已定居机器人负责存储至少 $2c$ 个节点、至多 $4c$ 个节点的信息。假设所有机器人都在探索过程中定居，并形成 k 个不同的组。于是，每个组包含介于 $2c$ 和 $4c$ 个之间的节点。

因此，在这种情况下，覆盖的节点总数满足 $k \cdot 2c \leq n \leq k \cdot 4c$。
由于初始时机器人的数量为 $k = \lceil n/c \rceil$，因此可得 $k \cdot 2c > n$。
该不等式导致矛盾，因为它意味着并非所有机器人都能在探索过程中定居。因此，某些机器人必须返回源节点 s。
此外，由于至少有一个机器人被指定用于存储源节点 s 本身的信息（处于振荡状态），并且至少还有另一个机器人（处于领导者状态）在完成探索后必然返回 s，因此源节点 s 在该过程结束时将包含至少两个机器人。这证明了该引理。

2. 我们通过矛盾来证明这一点。假设存在一个节点 v，使得对每个机器人 r都有 $v \notin Q_i$。根据该算法，当机器人 r 在探索过程中首次访问 v 时，节点 v 会被包含在 Q_i 中。在探索过程结束时，图中的每个节点最终都会被某个机器人访问。如果 v 不属于任何 Q_i，则意味着在整个探索过程中没有任何机器人访问过 v。然而，这与探索算法相矛盾，该算法保证遍历覆盖整个图，且没有任何节点保持未访问。

因此，我们的假设是错误的，由此可知，对于每个节点 v，至少存在一个机器人 r，使得 $v \in Q_i$ 且 $\bigcup_{i=1}^{b} Q_i = V$。
现在，我们必须证明 $v \in Q_i$ 是唯一的。根据算法，未访问的顶点 v 在首次被发现时，会作为必需节点加入组 W_i。具体而言，一旦顶点 v 被放入 Q_i，v 处的访问标记就会阻止后续任何组将 v 认领为必需节点。

考虑属于另一组 W_j 的机器人，其中 $j \neq i$，它之后到达 v。读取到 v 已访问后，机器人会遵循算法，根据遍历阶段选择将 v 用作辅助节点或回溯。如果组 W_j 在 v 扩展，那么组 W_i 就会收缩，从而确保单个组将 v 保留为必需节点。因此，对于所有 $i \neq j$，没有任何顶点能同时属于两个不同的 Q_i 和 $Q_i \cap Q_j = \varnothing$。

结合上述情况，我们得到 $\{Q_1, \ldots, Q_b\}$ 是 V的划分，正如所要求的那样。

3. 在探索过程中，每个已确定组（超节点） W_i 负责介于$2c$和 $4c$个必需节点之间的信息。根据第 (iii) 部分，集合 Q_1、.....、Q_b构成 V的一个划分，因此 G的每个顶点都恰好被分配给一个组。因此，覆盖 n 个节点所需的机器人总数至少为 $k/2$，其中每个组 W_i 覆盖 $2c \leq |Q_i| \leq 4c$ 个节点。

4. 不失一般性，设 W_1 是包含全局源节点 S 以及满足 r.Source(S) = 是 的机器人 r 的超节点。我们将每条组间边从父节点指向子组，并从子节点指向父节点；因此， $\mathcal{T}$是一个定向图。我们通过证明无环性和连通性，证明 $\mathcal{T}$是以 W_1 为根的有根树。

无环性。反设 $\mathcal{T}$包含一个有向环 $W_{i_1} \to W_{i_2} \to \cdots \to W_{i_k} \to W_{i_1}$与 $k \geq 2$。根据算法的构造，当且仅当组 W_q 由源自组 W_p的机器人新组建时，才会创建组间边 $(W_p, W_q) \in \mathcal{E}$。因此， W_p 必须先于 W_q 创建。

然而，边 $W_{i_k} \to W_{i_1}$ 的存在意味着 W_{i_1} 是在 W_{i_k}之后创建的，这与 W_{i_1} 是有向环中 W_{i_k} 的祖先这一事实矛盾。因此，这样的有向环不可能存在。
此外，在组 W_{i_ℓ} 的扩展和收缩操作过程中，算法会保留各组之间的父子关系，并且绝不会向先前创建的祖先组添加边。因此，整个执行过程中始终保持无环性。

因此，$\mathcal{T}$ 是无环的。

连通性。探索过程从源组 W_1 开始。每当形成一个新组 W_j 时，它都是由一个已完成群组形成过程的现有组 W_i 创建的。在这一转换过程中，机器人会明确记录一条连接 W_i 和 W_j 的组间边。因此，每个新形成的组都有一条定义明确的关联边，将其连接到前驱。

通过反复沿着这些已记录的组间边前进，可以从源组 W_1 到达任意组 W_j。

如果组 W_i 随后发生收缩，那么根据算法，它会更新自己的组间边，以保持与扩展中的组 W_j 的连通性。在此过程中，没有任何组会失去连接。

因此，每个组都始终与源组连通，而结构 $\mathcal{T}$ 也是连通的。

由于 $\mathcal{T}$ 的底层无向图是连通的，且有向父关系无环并具有唯一根节点 W_1，因此 $\mathcal{T}$ 是一棵以 W_1 为根的定向树。 □

引理4. 考虑树 $\mathcal{T} = (\mathcal{W}, \mathcal{E})$，所有机器人都已共同获得 $\mathcal{T}$（G）的连通性信息，并且位于源节点 $S \in W_1$。经过局部计算后，每个机器人 r 都被分配到唯一的目标超节点 W_p，即 $r.dsg\_gr = W_p$，使得存在一个节点 $v \in W_p$ 满足 col(r) =col(v）。

证明。我们分两部分证明该引理：机器人到超节点分配的存在性和唯一性。

根据构造，每个超节点 W_i 都包含一组必需节点，并且对于每种机器人颜色，某个超节点中至少存在一个颜色与该机器人匹配的必需节点。设 r 是任意一个机器人。由于所有机器人都完全了解这些超节点及其必需节点，因此至少存在一个超节点 W_p，其中包含一个必需节点 v，满足 col(v) =col(r）。因此，存在一种将 r 分配给 W_p 且满足颜色约束的方案。

反设存在矛盾：机器人 r 被分配到两个超节点 W_p 和 W_q，或者两个机器人r_1 和 r_2 被分配到同一个必需节点 v。根据分配规则，当机器人被分配到某个必需节点时，该节点会立即标记为已分配。设节点 $v \in W_p$ 被分配给了 r_1。当 r_2 尝试进行分配时，它会发现 v 已经被分配，因此无法选择它。同样，机器人不可能同时被分配到两个超节点，因为分配是基于颜色匹配的首个未分配节点原子地进行的。这与假设矛盾。因此，每个机器人恰好被分配到一个超节点，而每个必需节点至多被分配给一个机器人。

结合存在性和唯一性，我们得出结论：经过局部计算后，每个机器人 r 都有唯一的目标超节点 W_p，使得存在一个节点 $v \in W_p$ 满足 col(r) =col(v)。 □

引理5. 设 $\mathcal{W}_f = W_{i_1}, \ldots, W_{i_l} \subseteq \mathcal{W}$为目标超节点集合，其中某个机器人 r 满足$r.dsg_gr = W_{i_j}$。阶段 3 的阶段 1 结束后，以下结论成立：
1. 设 $r \in \mathcal{R}$为任意机器人，且对某个 $W_{i_j} \in \mathcal{W}_f$有 $r.dsg\_gr = W_{i_j}$。设 π(r) =($W_{q_1}, W_{q_2}, \ldots, W_{q_t}$)，其中 $W_{q_\ell} \in \mathcal{W}$ 为从 W_{q_1} 到 W_{i_j}的下行路径上的超节点序列。则，(a) 若$r.stmp_gr = W_{q_1} \in \mathcal{W} - \{W_{i_j}\}$或 $r.stmp\_gr = \mathrm{NULL}$，即 r位于源点 S($W_{q_1} = W_1$)，则存在一条从超节点 W_{q_1} 到 W_{i_j}的路径 π(r)，该路径上的每个超节点 W_{q_ℓ} 至少包含一个机器人，从而使 r 能够高效到达其目标超节点 W_{i_j}。(b) 若 $r.stmp\_gr = W_{i_j}$，则 r 位于目标超节点。2. 每个超节点 $W_{i_j} \in \mathcal{W}_f$ 恰好包含一个机器人。3. 对于每个满足 $r.stmp_gr = W_{i_j}$的机器人 $r \in W_{i_j}$，维护计数器变量 ctr 和数据结构 D，其中存储以 W_{i_j} 为根的子树所需的机器人数量，以及每个子超节点所需机器人的颜色，以及相应的请求额外机器人的 gr num。

证明。

1. 设 $r \in \mathcal{R}$ 为一个满足 $r.dsg_gr = W_{i_j} \in \mathcal{W}_f$的机器人。

   情形 (a)：根据引理4， $\mathcal{T}$是一棵树。因此，在 $\mathcal{T}$中，从超节点W_{q_1} 到 W_{i_j}存在一条唯一的路径 $\pi(r)$。我们证明路径 $\pi(r)$上的每个超节点都至少包含一个机器人。
   我们用反证法证明。假设$r.stmp_gr = W_{q_1} \neq W_{i_j}$，或者说$r.stmp_gr = \text{NULL}$ （即r 位于源超节点 W_1），并假设从 W_{q_1} 到 W_{i_j} 的路径上存在一个不包含任何机器人的超节点 W_{q_p} 。

   根据算法的构造，目标组为$r.dsg_gr = W_{i_j}$ 的机器人要么放置在 W_{i_j} 本身，要么放置在树 $\mathcal{T}$中 W_{i_j} 的最近祖先超节点。由于$r.stmp_gr = W_{q_1} \neq W_{i_j}$，机器人 r 不能放置在 W_{i_j}。

   然而，如果 W_{q_p} 为空，那么 r 应将其临时组更新为$r.stmp_gr = W_{q_p}$，这与假设 W_{q_p} 为空相矛盾。因此，路径 $\pi(r)$上的每个超节点都至少包含一个机器人。

   情况 (b)：假设 $r.stmp\_gr = W_{i_j}$。根据定义， r 已位于其目标超节点，无需进一步遍历。

   在这两种情况下， r 要么已位于其目标位置，要么存在一条路径，可从其当前位置经过包含机器人的超节点到达 W_{i_j}。因此，每个已分配目标超节点的机器人都有一条有保障的路由可以抵达该超节点，这就证明了该结论。

2. 考虑一个超节点 $W_{i_j} \in \mathcal{W}_f$。根据分配过程，恰好指定一台机器人 r 拥有 $r.stemp\_gr/ttemp\_gr = W_{i_j}$。算法确保不会将同一个目标超节点分配给其他机器人，因为每次分配都是唯一的，且冲突会通过最小ID规则解决。此外，任何可能将 W_{i_j} 作为中间节点进行遍历的机器人（即出于遍历目的）都不会成为 W_{i_j} 的驻留者，因为只有目标为 W_{i_j} 的机器人会留在那里。因此，在阶段 1 结束时，恰好有一台机器人占据 W_{i_j}。由此，每个超节点 $W_{i_j} \in \mathcal{W}_f$ 恰好包含一台机器人。

3. 设 $r \in W_{i_j}$ 是一个带有 $r.stmp_gr = W_{i_j}$的机器人。根据构造， r 是被分配来管理目标超节点 W_{i_j} 的机器人。在阶段 3 的局部计算期间， r 从 W_{i_j} 的所有子超节点 W_{η_1}、.....、W_{η_δ} 收集信息，了解满足各自子树需求所需的机器人数目和颜色。

   算法会显式更新计数器变量 $r.ctr$，以反映以 W_{i_j} 为根的子树所需的机器人总数，其中包括 W_{i_j} 自身的需求。同时， r 构建并维护数据结构 D，记录每个子超节点所需的机器人数目、所需的具体颜色，以及请求额外机器人的组号。

   因此，对于每个带有 $r.stmp_gr = W_{i_j}$的机器人 r，计数器变量 ctr 和数据结构 D 都得到了正确维护。 □

引理 6. 对于每个 $W_{i_j} \in \mathcal{W}_f$，令 d 表示目标超节点为 W_{i_j} 的机器人数量。在阶段 3 的阶段 2 结束时，恰有 d 个机器人位于 W_{i_j} 中，并且每个节点 $v \in W_{i_j}$ 至多容纳一个满足 col(v) = col(r) 的机器人 r 。

证明。设 $W_{i_j} \in \mathcal{W}_f$ 为一个目标超节点，并令 d 表示具有 $r.dsg\_gr = W_{i_j}$ 的机器人数量。根据构造，阶段 1 确保每个已分配到 W_{i_j} 的机器人都掌握有关其目标超节点的信息，以及从源点经由 $stmp\_gr$ 和 tmp_gr 变量到达 W_{i_j} 的路径。

在阶段 2 中，机器人根据各超节点上维护的数据结构 D 沿这些路径移动。基于 DFS 的放置过程确保机器人被路由至各自的目标超节点，且不会发生冲突。由于每个机器人都被唯一分配到一个超节点，并遵循其所占节点的颜色，因此不会有两个颜色相同的机器人被放置在同一个节点上。

因此，在阶段 2 结束时：

1. 恰有 d 个机器人位于 W_{i_j}，因为每个已分配给 W_{i_j} 的机器人都能抵达其目的地，且不会出现重复或丢失。

2. 每个节点 $v \in W_{i_j}$ 最多容纳一个机器人 r，使得 col(v) = col(r)，从而确保保留颜色匹配约束。

因此，机器人在 W_{i_j} 中的放置是正确的，引理得证。 □

定理 4. 在有根初始配置下，使用 $O(n/k \cdot \log(k+\Delta))$ 个比特的每个机器人，可让拥有 n 先验知识的 $k > 1$ 个机器人在 $O(n/k \cdot m)$ 轮次内确定性地解决 Location-Aware Dispersion 。

证明。该定理由引理 1-6 得出。因此，可在有根初始配置下，在 $O(n/k \cdot m)$ 轮次内确定性地解决 Location-Aware Dispersion 。每个机器人 r 只存储与其所属组 W_i相关的信息，这需要 $O(n/k \cdot \log(k+\Delta))$ 比特的内存。因此，在所述时间和内存界限内，可以确定性地解决 Location-Aware Dispersion 。

□

4 任意图：有根， $k \leq n$，未知 n

本节研究有根配置下的 Location-Aware Dispersion 问题：所有 k 个机器人最初都位于同一个源节点 S。图中的节点总数 n 未知。与有根设置中一样，所有机器人最初都位于同一个节点，因此每个机器人都能确定机器人的总数 k。因此，在有根配置下， k 最初是否已知并不影响模型。

核心思路是通过基于 DFS 的探索进行迭代猜测（如第 3 节所述），使机器人在不知道 n 的先验知识的情况下探索任意图。由于第 3 节中已知 n 的策略依赖于用 n 限定 DFS，机器人首先假设 $n = k$，并启动 DFS；其中每个节点至多分配给一个机器人，且每个组由该组中的振荡机器人协调。如果遍历完成，且所有机器人均已就位、没有留下未探索节点，则猜测正确，分散成功。否则，若无法使所有机器人定居（或发现了未探索节点），就说明猜测值太小。此时，所有机器人回溯并在 S 处重新集合，将假定的 n 值加倍，然后开始新一轮 DFS 尝试。通过逐步加倍猜测值，机器人最终会达到一个包含真实 n 的区间 $[2^i k, 2^{i+1}k]$，从而确保在该次迭代中完成分散。无论对 n 的初始估计低了多少，该策略都能保证正确性；与已知 n 的情形相比，它在 DFS 尝试次数上只增加了对数级开销 O（log(n/k)）。图探索阶段完成后，应用第 3 节中的集合和分散步骤，从而完成 Location-Aware Dispersion。

定理 5. Location-Aware Dispersion 可在有根初始配置下，以确定性方式解决： $k > 1$ 个机器人在对 n 没有先验知识的情况下，使用每个机器人 $O(n/k \cdot \log(k+\Delta))$比特，在 $O(\log(n/k) \cdot n/k \cdot m)$轮次内完成。

证明。该算 法采用迭代猜测策略，并结合基于深度优先搜索的探 n.

初始时，机器人假设 $n = k$，并从源节点 S开始进行有界深度优先搜索。在此次遍历过程中，每组分配一个机器人，以确保每组至少包含 $2c$ 个节点，且至多包含 $4c$ 个节点。如果遍历成功完成，即 $count = \delta_S - 1$ 且一些机器人返回源节点 S，那么根据引理 2，分散即可实现。

否则，假设探索过程中只剩下一台活跃机器人，且它检测到存在未探索节点。这意味着遍历尚未完成。在这种情况下，剩余的活跃机器人将成为领导者，并且没有活跃机器人返回 S，这与引理 2 矛盾。因此，当前估计低于 n的实际值。此时，所有机器人沿深度优先搜索遍历回溯，返回源节点 S，将猜测值 n加倍，并重复深度优先搜索过程。

由于猜测值 n 在每次迭代中都会翻倍，经过 $O(\log(n/k))$ 次迭代后，估计值将足够准确，深度优先搜索也会成功完成。正如已知 n 情形（定理 4）中所证明的，每次深度优先搜索尝试需要 $O(n/k \cdot m)$ 轮。因此，该算法的总轮复杂度为 $O(\log(n/k) \cdot n/k \cdot m)$。

每个机器人的内存需求与已知 n 设置下相同，因为机器人只存储局部深度优先搜索状态和协调信息。因此，每个机器人使用 $O(n/k \cdot \log(k+\Delta))$ 比特的内存。

因此，在未知 n 的有根配置中，分散问题可以在规定的时间和内存界限内确定性地求解。

□

5 任意图：一般， $1 < k \leq n$

我们研究一般初始配置下的 Location-Aware Dispersion ，也就是说，图 G 中存在一个源节点 s ，其初始时至少容纳两个机器人。在任何一般初始配置中，恰好容纳一个机器人的节点称为 $single$源点，而容纳至少两个机器人的节点称为 $multi$源点。我们的方法基于 Kshemkalyani 和 Sharma 的多源深度优先搜索框架 [7], ，该框架用于经典分散，并经过调整以保留 Location-AwareDispersion所需的群组结构。

基本思路：位于 $single$ 源点的机器人会留在当前节点，因为如第 2 节所述，单个机器人在没有节点内存的情况下无法探索图。相反，位于 $multi$-源点的机器人会启动深度优先搜索遍历，组成有界大小的群组，并由指定的振荡机器人协调（如第 2 节所述）。当两个深度优先搜索遍历在某个节点相遇，或遇到一个 $single$-源点机器人时，就会执行归并步骤。已定居机器人较多的遍历得以保留，而较小的遍历则被合并，其机器人重新分配给保留的深度优先搜索；若数量相同，则由最小树标签决定。多个同时发生的碰撞通过两两归并解决。随着时间推移，并行的深度优先搜索遍历数量逐渐减少，最终只剩下一个遍历。

5.1 特殊情况：当 $k = n$

当 $k = n$ 且 k 和 n 均已知时，每个源点都会启动一次深度优先搜索，形成大小为 $[2c, 4c]$ 的组；吸收规则确保占主导地位的遍历继续探索。

算法：每个机器人将树标签初始化为其源点处最小 ID 机器人的标识符，并设置一个组编号计数器，用于记录已定居机器人的数量。 $single$源点机器人将其状态更改为 $wait$ ，并留在当前节点，直到被发现并并入经过中的深度优先搜索遍历。每个 $multi$源点都会启动一次有界深度优先搜索遍历，其中深度优先搜索头遍历图并形成大小为 $[2c, 4c]$ 的组。当深度优先搜索头到达某个节点时，机器人会等待一段固定时间，以检测与其他深度优先搜索遍历发生的碰撞。归并规则以确定性方式应用，遍历由负责各组的机器人引导；已定居机器人较多的遍历得以保留，其他遍历则被合并。被吸收遍历中的每个机器人都会重新分配给保留的深度优先搜索，该搜索最多会重新访问被折叠遍历中的每个节点一次。当保留的深度优先搜索遇到 $single$ 源点机器人时，该机器人会立即并入该遍历，成为其已确定组的一员。

定理 6. Location-Aware Dispersion 可以在一般初始配置下，以确定性方式解决 $k = n$ ，其中 $k > 1$ 个机器人具有关于 n 和 k 的先验知识，并在 $O(n/k \cdot m)$轮内完成，每个机器人使用 $O(n/k \cdot \log k)$比特。

证明。考虑一个连通图 $G = (V, E)$，其中有 $|V| = n$ 和 $k = n > 1$ 个机器人处于一般初始配置，每个机器人都知道 n 和 k。每个机器人都会初始化一个树标签和一个组编号计数器，其值等于其源点处已定居机器人的数量。

每个多源机器人都会发起一次有界深度优先搜索遍历，形成大小为 $[2c, 4c]$的组。当两次深度优先搜索遍历相遇时，将应用确定性包含规则：已定居机器人较多的遍历得以保留，另一次则被合并。由于组大小严格递增，活跃遍历的数量会单调减少，从而确保终止。单源机器人会一直保持在 $wait$ 状态，直到被发现，随后并入一条经过中的深度优先搜索遍历。

每次被包含的遍历都可能导致节点被再次访问，但每个节点最多只会被再次访问一次。因此，每次深度优先搜索都会探索 $O(m)$ 条边；由于最多有 $O(n/k)$ 次深度优先搜索遍历依次存续，总运行时间为 $O(n/k \cdot m)$ 轮次。

每个机器人存储一个树标签、一个组计数器和深度优先搜索状态，需要 $O(n/k \cdot \log k)$ 比特的内存。因此， Location-Aware Dispersion 可在一般初始配置下确定性地解决，且满足所述时间和内存界限。

□

5.2 当 $k < n$

我们考虑机器人可能知道也可能不知道 n 和 k 的情形。

算法：每个源点 S_p 启动一次深度优先搜索，并采用初始猜测 $n = k_p$，由振荡机器人协调形成有界大小群组。主要挑战在于，由于组大小存在差异，以及不同深度优先搜索的组之间存在不同步的振荡，独立的深度优先搜索可能永远无法在任意节点处相遇。因此，合并不能依赖探索过程中的偶然相遇。

为保证收敛，每个源点保留一个 $passive$ 机器人。由于每次深度优先搜索遍历最终都保证会访问每个节点，因此，任何到达某个源点的遍历都会遇到其 $passive$ 机器人，并与从该源点发起的遍历合并。然而，当源点恰好包含两个机器人时，还会出现一个额外的挑战。

让我们考虑某个源点 s恰好存在两个机器人的情况。为保证收敛，其中一个机器人留在 s 处并处于 $passive$ 状态，而另一个机器人开始探索。然而，如前文第1节所述，单个机器人无法探索匿名图。

这使我们根据源点的初始配置，得到两种可能的情况。

情况1：如果初始时至少有一个源点上有两个以上的机器人：假设存在一个源点 S ，其初始时至少包含三个机器人。在这种情况下，从S 发起的深度优先搜索可以正常进行，并在源点留下一个 $passive$ 机器人。每当该深度优先搜索遇到另一次遍历或一个 $single$-源点机器人时，就会根据标准吸收规则进行合并：已定居机器人较多的遍历继续进行，较小的遍历则被吸收。因此，最终所有机器人都会聚集到占主导地位的遍历所对应的源点，并作为一个统一的深度优先搜索继续探索。

如果另一个源点初始时恰好包含两个机器人，这两个机器人会直接开始深度优先搜索，不会留下 $passive$ 机器人，因为单个机器人无法进行探索。它们的遍历最终会到达源点S，此处有 $passive$ 机器人留守，并入从 S发起的深度优先搜索。

因此，只要有一个源点至少包含三个机器人，就能保证全局收敛：所有深度优先搜索过程最终会合并为一个过程，该过程探索整个图，随后进行分散。

这种情况每个机器人仅需 $O(n/k \cdot \log(k+\Delta))$内存位数，与一般算法的内存上界相符。

情况2：如果不存在至少一个初始时承载超过两个机器人的源点：考虑一个源点 S ，其初始时恰好包含两个机器人。由于单个机器人无法探索，两个机器人启动相互独立的有界深度优先搜索遍历，源点处不留下 $passive$ 机器人。存在两种可能情况：

- 如果源点 S 的两个机器人中任一遇到另一次深度优先搜索遍历或一个 $single$-源点机器人，则按照标准吸收规则进行合并：包含更多已安置机器人的遍历继续进行，而较小的遍历被吸收。一旦合并产生一个拥有超过两个机器人的源点，过程即切换到情况1：一个机器人在源点处成为 $passive$ ，其余机器人启动一次新的深度优先搜索。

- 如果在探索过程中没有发生任何相遇，则这种情况表明每个源点（如果存在）初始时都恰好包含两个机器人，并且源自不同源点的DFS遍历在整个执行过程中始终完全不相交。由于没有任何遍历与其他遍历或 $single$-源点机器人发生交互或合并，每一对机器人都独立完成其有界探索，并获得图 G的完整知识。源点 S处的两个机器人随后根据节点颜色与机器人颜色之间的匹配，分散到不同的最近节点。如果某个节点已被另一个机器人占据，机器人便利用其存储的知识移动到下一个最近的可用节点。若两个或更多机器人同时到达同一节点，则使用它们的唯一标识符进行决胜。

这一过程确保了分散的成功，但要求每个机器人存储完整图知识，存储需求为 $O(n\log(k+\Delta))$ 比特。

定理 7。 Location-Aware Dispersion 可在一般初始配置下于 $O(\log(n/k) \cdot n/k \cdot m)$ 轮次内确定性地解决。若某个源点最初至少容纳三个机器人，则该算法要求每个机器人使用 (i) $O(n/k \cdot \log(k+\Delta))$ 比特的内存；否则要求每个机器人使用 (ii) $O(n \cdot \log(k+\Delta))$ 比特的内存。

证明。我们通过反证法证明该定理。为导出矛盾，假设该算法无法在所述条件下解决 Location-Aware Dispersion 问题。那么以下至少一项必须成立：

(i) 算法不终止，或 (ii) 算法终止，但未能实现正确的分散。

假设算法不终止。那么，在尚未实现收敛的情况下，会执行无限多个深度优先搜索阶段。

考虑任意一个深度优先搜索阶段。要么至少有两次深度优先搜索遍历相交，要么它们互不相交。如果两次深度优先搜索遍历相交，那么根据吸收规则，恰好有一次遍历得以保留，另一次则被吸收。因此，活跃的深度优先搜索遍历总数会严格减少。由于源点数量有限，这一过程不可能无限持续。因此，在这种情况下，无限多个阶段后仍未终止是不可能的。

否则，假设在某个阶段中没有任何深度优先搜索遍历相交。那么，每次遍历都会完成其有界探索并返回源点。根据算法，下一阶段的探索容量会呈指数级增长。由于图 G 是有限的，经过有限个阶段后，至少一次遍历的探索范围会大到足以与另一次遍历相交，或到达一个包含被动机器人的源点。这同样会触发吸收，从而减少深度优先搜索遍历的数量。

因此，在每个阶段中，要么遍历次数严格减少，要么探索容量增加。这两个量都是有界的，这意味着算法必定终止。这与不终止的假设相矛盾。

现在假设算法终止但未实现分散。如果执行到达恰好只有一个活跃深度优先搜索遍历的配置，则该情形可归结为有根情形。根据有根算法的正确性（由引理4），分散得以实现，这与假设相矛盾。

否则，假设算法在多个从未合并的DFS遍历下终止。这种情况只有在每个源点最初恰好包含两个机器人且没有任何遍历与另一遍历相交时才会发生。在这种情况下，每个深度优先搜索最终完成其探索并获得该图的完整知识。利用这些信息，每一对机器人通过确定性决胜分散到不同节点。因此，分散仍然得以实现，这与假设相矛盾。

由于不终止与错误终止都会导致矛盾，算法必须终止并正确求解 Location-Aware Dispersion 问题。

每个深度优先搜索阶段需要 $O(n/k \cdot m)$ 轮次，探索容量在每个阶段都会翻倍，因此需要$O(\log(n/k))$ 个阶段。因此，轮复杂度总计为 $O(\log(n/k) \cdot n/k \cdot m)$。内存界限直接取决于机器人存储的是部分还是完整的图信息，如定理所述。因此，该定理成立。 □

6 任意图：分散，$k \leq n$，已知 n

6.1 特殊情况：当 $k = n$

在本小节中，我们研究分散初始配置下的 Location-Aware Dispersion 问题，其中图中的每个节点最初至多有一个机器人。因此，$k = n$ 个机器人分布在任意图 G 的 n 个不同节点上，且所有机器人都知道 k 和 n。

我们的方法基于 [3], 中的“相邻两个智能体会合协议”，该协议保证两个相邻智能体在 $O(\log L)$ 轮次内会合，其中 L 是标识符最高的机器人的标识符。随着时间推移，最初的分散配置会演变为一般配置，其中至少有一个节点容纳两个或更多机器人。

相邻两个智能体会合协议：考虑两个机器人 r_u 和 r_v，它们最初分别位于图中相邻的节点 u 和 v，其中 u 通过 u处的出端口 p_u 与 v 连通。目标是让机器人 r_u 通过端口 p_u与其邻居 r_v 会合。每个机器人首先将自己的标识符填充至 $\log L$位二进制字符串，然后在左侧追加其按位取反的结果，从而形成一个长度为 $2 \log L$ 比特的新标识符，其中 $\log L$ 是机器人中最长标识符的长度。协议按位依次进行，从最低有效位（LSB）到最高有效位（MSB）。对于每个位位置 i，如果 r_u的新标识符的第 i位是 1，该机器人就在这一对轮次的第一轮从节点 u 移动到节点 v，并在第二轮返回 u；如果该位是 0，则机器人在两轮中都停留在 u。由于所有机器人都有不同的标识符，至少存在一个比特位置

使得一个机器人的新标识符在该位置为 1，而另一个机器人的新标识符在该位置为 0。在对应轮次中，一个机器人移动，而另一个保持不动，从而确保两个机器人会合。

因此，该协议在 $4\log L$ 轮次内完成，每个位位置分配两轮。该协议完全是局部的，无需了解 r_v 的状态或行为。机器人可以使用其打算经过的任意指定端口调用此过程。

请注意，在 [3], 中，假设机器人对 $\log L$ 具有先验知识。在我们的设定中，机器人知道 n 的值，而每个标识符属于 $[1, k^{O(1)}]$ 的机器人都假定知道该值。

算法：初始时，所有机器人都处于分散配置，并且每个机器人都知道 n 的值。每个位于节点 u 的机器人 r 都会尝试通过端口 0 访问其邻居 v，以启动相邻两个智能体会合协议。在该协议执行期间，如果机器人 r 在节点 v 遇到另一机器人r'，它就与 r' 组成一组，此后遵循 r'，而 r' 则继续执行算法。通过在所有相邻对之间反复应用此协议，可以保证位于相邻节点上的机器人相遇。经过至多 $4\log k$ 轮次后，初始的分散配置将转变为一般配置，其中至少有一个节点上有两台或更多机器人。

一旦获得这样的一般配置，我们就应用针对一般初始设置（第5节）设计的算法，该算法可保证在具有 $k = n$ 的任意初始摆放下，确定性地实现 Location-Aware Dispersion 。

定理8. 在分散初始配置下，拥有 n、k 先验知识的 $k = n$ 个机器人，可以使用每个机器人 $O(n/k \cdot \log\ k)$ 比特，在 $O(\log(k)\ + n/k \cdot m)$ 轮内确定性地解决 Location-Aware Dispersion 。

证明。我们证明该算法的正确性、终止性和复杂度。

初始时，机器人处于分散配置中，每个节点恰有一个机器人，并且都完全了解 n 和 $k = n$。每个机器人独立地与其一个邻居发起相邻两个智能体会合协议。根据该协议的性质，位于相邻节点的任意两个机器人都保证在 $O(\log L)$ 轮内会合。

由于图 G 是连通的，反复执行会合协议可确保至少一对相邻机器人会合，从而组成一个至少有两个机器人的组。一旦发生这样的会合，配置便不再是分散配置，而会变为一般配置，其中至少有一个节点上有多个机器人。

到达一般配置后，机器人执行第5节中所述的适用于一般初始设置的确定性算法。根据定理 7，该算法能够正确解决具有任意初始放置且已知 n 的 Location-AwareDispersion 问题，确保所有机器人均位于不同节点上，且每个节点至多承载一个机器人。因此，正确性得以保证。

会合阶段至多需要 $O(\log L)$ 轮次，即可将分散配置转换为一般配置。完成转换后，一般设置下的算法需要 $O(\log(n/k) \cdot n/k \cdot m)$ 轮次。由于在分散设置下 $k = n$，总轮复杂度为 $O(\log(L)\ + \log(n/k)\ \cdot n/k \cdot m)$。因此，时间和空间复杂度可由定理 7 得出。 □

6.2 当 $k < n$

我们研究 Location-Aware Dispersion 问题在 $k < n$个机器人的分散初始配置下的情况，其中图的每个节点最初至多承载一个机器人。因此， k 个机器人占据任意图 G中的 k 个不同节点，并且所有机器人都知道 k 和 n 。

总体思路是将分散配置转化为一般配置，也就是说，转化为至少有一个节点容纳两个或更多机器人的配置。一旦存在这样的源点，就可以使用为一般配置开发的技术求解 Location-AwareDispersion 问题。

这一转换与经典确定性聚集问题密切相关。具体而言，相关工作 [4] 研究了这样的场景： k 个机器人放置在一个有 n 个节点的图上，目标是让所有 k 个机器人在单个节点相遇并终止。他们提出了能够在任意图上实现带检测的聚集的确定性算法。

先前工作的结果总结 [4]: 作者首先提出了一种基于通用探索序列（UXS）的集合算法，需要 $\tilde{O}(n^5)$ 轮次。随后，他们聚焦于非分散配置（即某些节点最初有多个机器人），并提出了一种效率更高的两阶段算法。在

第一阶段，一组机器人协作构建图的地图，耗时 $O(n^3)$ 轮次；第二阶段则利用这张地图，将所有机器人路由到指定的集合节点。

他们随后处理分散配置，首先尝试将其转换为非分散配置。所得的时间复杂度取决于任意两个机器人之间的最小距离。若最近机器人对的距离为 i，这些机器人会相遇，并在$O(n^i \log n)$ 轮次内形成非分散配置。随后，所有机器人在 $\tilde{O}(n^5)$ 轮次内于指定节点集合。

因此，从分散配置将所有机器人集合到单个节点时，存在以下复杂度权衡：

- 若初始配置为非分散配置，或者该配置为分散配置且至少有两个机器人彼此相距为 2，则集合在 $O(n^3)$ 轮次内完成。
- 若初始配置为分散配置，且至少有两个机器人位于距离 i 处（对于 $i \in \{3,4,5\}$），则集合在 $O(n^i \log n)$ 轮次内完成。
- 否则，集合需要 $\tilde{O}(n^5)$ 轮次。

因此，对于任意初始配置，以下界限作为机器人数量 k 的函数成立：

1. 若 $k \geq \lfloor n/2 \rfloor + 1$，则聚集在 $O(n^3)$ 轮次内完成。
2. 若 $k \geq \lfloor n/3 \rfloor + 1$，则聚集在 $O(n^4 \log n)$ 轮次内完成。
3. 否则，聚集需要 $\tilde{O}(n^5)$ 轮次。

重要的是，该算法假设已知图的规模 n，但不要求知道机器人的数量 k。

算法：我们的方法建立在 [4] 的结果之上。由于机器人知道 k 或 n，因此可以直接应用该工作中的确定性聚集例程，在单个节点上集合。具体而言，若最近机器人对之间的距离为 $i \in \{1,2,3,4,5\}$，则聚集在 $O(n^i \log\ n)$ 轮次内完成。获得这样的一般配置后，我们调用针对一般初始设置设计的算法（第5节）。

如果不存在这样的机器人对，聚集过程会将所有机器人收到单个节点，耗时 $O(n^5)$ 轮次。形成此有根配置后，我们应用为有根初始设置开发的算法（第 3 节）。

定理 9。对于 Location-Aware Dispersion 问题，在分散初始配置中，给定 $k < n$ 个机器人以及关于 n的先验知识，可以通过确定性方法求解；每个机器人使用 $O(M^* + n/k \cdot \log(k+\Delta))$ 比特的内存，其中 M^* 表示实现通用探索序列（UXS）所需的内存。时间复杂度为：(i) 若 $i = 1, 2$，则为 $O(n^3 + n/k \cdot m)$ 轮次，其中 i 是最近机器人对之间的距离；(ii) 若 $i \in \{3,4,5\}$，则为 $O(n^i \log\ n + n/k \cdot m)$ 轮次；(iii) 其他情况为 $\tilde{O}(n^5)$ 轮次。

证明。初始时， $k < n$ 个机器人处于分散配置中，分别位于连通图 G的不同节点上，且所有机器人都知道 n 和 k。目标是将此分散配置转换为一种配置，使得 Location-Aware Dispersion 问题已可求解。

算法首先应用来自 [4] 的确定性聚集过程。该过程根据最近机器人对之间的距离，保证所有机器人聚集在单个节点上，或至少有一个节点容纳多个机器人。无论哪种情况，所得配置都不再是分散的。

如果聚集过程得到一般配置，即至少有一个节点容纳两个或更多机器人的配置，我们就调用适用于一般初始配置的算法（第5节）。根据定理 7，该算法可以正确解决 Location-Aware Dispersion 问题。

如果聚集过程得到有根配置，即所有机器人都聚集在单个节点上，我们就应用适用于有根初始配置的算法（第 3 节）。根据该算法的正确性，可以确定性地实现分散。

因此，在所有情况下，该算法都能产生正确的彩色分散。

当 [4], 最近机器人对之间的距离为 $i \in \{2,3,4,5\}$时，聚集阶段在$O(n^i \log n)$轮次内完成。若不存在这样的机器人对，则聚集阶段在 $\tilde{O}(n^5)$轮次内完成。聚集阶段终止后，所得配置要么是有根配置，要么是一般配置。在这两种情况下，后续执行都会保持渐近界，因此可得所声称的时间和内存复杂度。 □

7 任意图：未知，$k \leq n$

对于已知的 n，如果配置是有根配置且已知 k，机器人可以立即检测到这一点，并使用第 3 节的算法解决问题。对于其他所有情况，机器人首先在单个节点处集合，从而利用 Molla等人的结果创建有根配置。[4]，通过通用探索序列，只需知道 n，即可在任意图上实现带检测的确定性聚集。当初始配置已经是有根配置但未知 k 时，机器人会探索图，以便正确识别。当形成有根配置且已知 n 后，机器人应用有根算法（第 3 节）来解决 Location-Aware Dispersion。

如果 n 未知而 k 已知，机器人仍然可以检测有根配置。它们按照第 4 节所述，通过对 n进行指数猜测来执行有根算法。其余配置——即 n 和配置类型都未知的情况——仍是一个开放问题。请注意，对于 Dispersion，无论初始配置（已知或未知）如何，也无论 n、k 已知还是未知，都不存在解决方案。

定理10. 使用每个机器人 $O(M^* + n/k \cdot \log(k+\Delta))$ 比特的内存， Location-Aware Dispersion 可以在具有 $k \leq n$ 个机器人的任意图 G上确定性地解决；其中初始配置未知，但机器人具有关于 n的先验知识。时间复杂度为：(i) 若 $k \geq \lfloor n/2 \rfloor + 1$，则为 $O(n^3 + n/k \cdot m)$ 轮次；(ii) 若 $k \geq \lfloor n/3 \rfloor + 1$，则为 $O(n^4 \log n + n/k \cdot m)$ 轮次；(iii) 其他情况下为 $\tilde{O}(n^5)$ 轮次。其中 M^*表示实现 UXS 所需的内存。

证明。我们证明，无论初始配置如何，当机器人具有 n 的先验知识时，都可以确定性地解决 Location-Aware Dispersion 。

如果初始配置是有根的，并且已知 k，机器人将立即执行第 3 节中的有根算法，该算法保证实现正确的分散。

否则，机器人首先应用 Molla等人提出的确定性带检测的聚集算法 [4]。该算法仅利用 n 和通用探索序列的知识即可在任意图上运行。终止时，所有机器人都聚集在单个节点上，并且能够检测到这一事件，从而形成有根配置。

如果配置是有根的但 k 未知，机器人将探索图，以实现对有根配置的正确识别。一旦确认有根性且已知 n，它们便执行第 3 节中的有根算法。

如果已知 k 但未知 n，机器人将检测有根性，并按照第 4 节中的描述，通过对 n 进行指数猜测来执行有根算法。由于猜测值最终会超过真实的 n，因此正确性得以保证。

使用正确的 n 值获得有根配置后，有根算法即可完成分散，并确保每个被占据节点恰好驻留一台机器人。

每台机器人所需的内存为 $O(M^* + n/k \cdot \log(k+\Delta))$，其中 M^* 是实现通用探索序列所需的内存。所述时间界分别由相应情形下确定性集合和有根分散的复杂度得出。

因此，当已知 n 时，可以在初始配置未知的任意图上确定性地解决 Location-Aware Dispersion 。 □

8 结论与讨论

本文研究了有根、分散和一般配置下的 Location-Aware Dispersion 问题，并考虑了机器人对 n 和 k 的先验知识所作的不同假设。结果汇总于表 1。

我们还研究了一个更具挑战性的情形：初始配置未知。当机器人知道 k 时，它们可以判断该配置是否为有根配置。然而，如果配置不是有根配置，或者机器人不知道 k，单个机器人就无法区分分散配置和一般配置，因此未知初始配置仍有若干尚未解决的情形。此外，一旦实现分散配置，无论是有根配置还是一般配置， Location-Aware Dispersion 问题都可以得到解决。这些发现凸显了配置识别、图探索与彩色分散可解性之间的内在联系。

未来研究的一个重要方向是解决其余尚未解决的情形：找出能够实现配置识别和图探索的最小附加假设，例如有限的节点内存、对图参数的部分了解，或可使用额外的机器人。另一个重要方向是探究我们提出的算法在时间和/或内存复杂度方面能否改进，以达到相应的时间/内存下界。弥合这些差距，并将我们的结果扩展到随机化或容错情形，是未来研究的自然方向。

参考文献

[1] J. Augustine 与 W. K. Moses Jr，“移动机器人的分散：内存-时间权衡研究”，载于 ICDCN，2018，第 1–10 页。[2] Y. Sudo、D. Baba、J. Nakamura、F. Ooshita、H. Kakugawa 与 T. Masuzawa，“未知无向图上使用白板的智能体探索”，载于 ARES，2010，第 1–6 页。[3] P. K. Chand、A. Das 与 A. R. Molla，“简要公告：使用移动智能体的分布式蝴蝶分析”，载于 SPAA，2025，第 623–627 页。[4] A. R. Molla、K. Mondal 与 W. K. Moses，“任意图上带检测的快速确定性聚集：多机器人的力量”，载于 IPDPS，2023，第 47–57 页。[5] A. D. Kshemkalyani 与 F. Ali，“图上移动机器人的高效分散”，载于 ICDCN，2019，第 218–227 页。[6] A. D. Kshemkalyani、A. R. Molla 与 G. Sharma，“任意图上移动机器人的快速分散”，载于 ALGOSENSORS，2019，第 23–40 页。[7] A. D. Kshemkalyani 与 G. Sharma，“任意匿名图上的近最优分散”，《计算机与系统科学杂志》，第 152 卷，第 103656 页，2025。[8] A. D. Kshemkalyani、M. Kumar、A. R. Molla、D. Pattanayak 与 G. Sharma，“分散在（a）同步条件下（几乎）是最优的”，载于 SPAA，2025，第 367–381 页。[9] Y. Sudo、M. Shibata、J. Nakamura、Y. Kim 与 T. Masuzawa，“移动智能体的近线性时间分散”，载于 DISC，2024，第 38 页。[10] T. Shintaku、Y. Sudo、H. Kakugawa 与 T. Masuzawa，“无需全局知识的移动智能体高效分散”，载于 SSS，2020，第 280–294 页。[11] A. D. Kshemkalyani、A. R. Molla 与 G. Sharma，“动态图上移动机器人的高效分散”，载于 ICDCS，2020，第 732–742 页。[12] A. Agarwalla、J. Augustine、W. K. Moses Jr、S. K. Madhav 与 A. K. Sridhar，“动态环中移动机器人的确定性分散”，载于 ICDCN，2018，第 1–4 页。[13] A. D. Kshemkalyani、A. R. Molla 与 G. Sharma，“利用全局通信的移动机器人分散”，《并行与分布式计算杂志》，第 161 卷，第 100–117 页，2022。[14] A. D. Kshemkalyani、A. R. Molla 与 G. Sharma，“网格上移动机器人的分散”，载于 WALCOM，2020，第 183–197 页。[15] R. Banerjee、M. Kumar 与 A. R. Molla，“定向网格上的最优容错分散”，载于 ICDCN，2025，第 254–258 页。[16] Himani 与 S. Pandit，“三角网格中的最优分散：无需先验知识实现高效”，载于 ICDCIT，2024，第 75–91 页。