arXiv:2301.04938v1 [cs.DC] 12 Jan 2023

距离-2-分散：具有更多约束的分散

Tanvir Kaur * Kaushik Mondal*

摘要

分散问题旨在将一组 $k(\leq n)$ 个可移动机器人放置在由 n 个节点组成的未知图的节点上，使得从机器人在图上的任意初始配置出发，最终配置中每个节点至多包含一个机器人。本文提出了分散问题的一种变体：初始时机器人数量不限，并增加一项约束，即最终配置中没有两个相邻节点包含机器人。也就是说，任意两个包含机器人的节点之间的距离至少为 2。我们将此问题称为距离-2-分散，简称 D-2-D。然而，即使机器人数量 k 小于n，也可能无法让每个机器人都找到一个不同的节点停留，同时满足我们新增的约束。具体而言，如果包含机器人的节点已经构成一个极大独立集，那么其他正在寻找节点停留的机器人（如果有的话）将找不到可停留的节点。因此，只有在无处可停时，我们才允许多个机器人停留在某些节点上。如果 $k \geq n$，则可以保证包含机器人的节点构成底层网络的一个极大独立集。

图 $G = (V, E)$ 有 n 个节点和 m 条边，其中节点是匿名的。它是一个端口标记图，即每个节点 u 都会为其每条关联边分配一个不同的端口号，端口号取自范围 $[0, \delta - 1]$，其中 δ 是节点 u 的度。机器人具有范围为 $[1, L]$ 的唯一标识符，其中 $L \geq k$。共址的机器人可以相互通信。我们给出一种算法，从有根配置（即初始时所有机器人都共址）开始求解 D-2-D，并在 $2\Delta(8m - 3n + 3)$ 个同步轮次后终止，每个机器人使用 $O(log\Delta)$ 的内存，且无需任何关于图参数 m、n 和图的最大度 Δ 的全局知识。我们还

*印度理工学院罗帕尔分校数学系，印度 tanvir.20maz0001@iitrpr.ac.in, kaushik.mondal@iitropar.ac.in

为 D-2-D 问题的轮次数给出了 Ω($m\Delta$) 的下界。

关键词——移动机器人、匿名图、分散、确定性算法。

1 引言

分散问题旨在将一组 k（$\leq n$）个移动机器人放置在一个包含 n 个节点的未知图的节点上，使得从机器人在图上的任意初始配置出发，最终配置中每个节点至多包含一个机器人。该问题由 Augustine 等人于 2018年 提出[2]。此后，文献在各种模型和不同假设下研究了这一问题[4, 16, 14, 17, 20, 12, 8, 15, 10, 1]。解决分散问题的主要工具是深度优先搜索遍历；由于机器人配备了内存，它们可以存储完成分散所需的重要信息，从而避免陷入环中。一个自然的问题随之产生：如果对分散问题施加一些额外约束，会发生什么？例如，任何机器人都不能在已定居机器人的一跳邻域内定居。这促成了距离-2-分散问题的提出。在这个问题中，任意放置在图上的 k 个机器人需要达到一种配置，使得没有两个相邻节点被已定居机器人占据。此外，未定居机器人只有在不存在其他可供其定居且满足新增约束的节点时，才能在已包含已定居机器人的节点上定居。因此，可能存在 $many$ 个没有已定居机器人的节点，即这些节点上没有机器人来存储任何信息；这样的图是零存储图。因此，若要以每个机器人仅需 $less$ 内存的方式解决该问题，它就变得颇具研究意义。

1.1 模型与问题

设 G 是任意一个连通无向图，包含 n 个节点、 m 条边，最大度为 Δ。节点是匿名的，即没有标识符。该图是端口标记图，也就是说，每个节点 u 都会为其每条边分配一个不同的端口号，端口号取自范围 $[0, \delta(u) - 1]$，其中 $\delta(u)$ 是节点 u 的度。每条边两端分配的端口号彼此独立。节点没有内存，因此 G 是一个零存储图。

系统中共有 k 个可移动实体，我们称之为机器人。每个机器人都拥有一个来自范围 $[1, L]$ 的唯一标识符，并且知道自己的标识符。在某一轮次中，如果两个或更多机器人位于同一个节点上，我们就称它们共址，此类机器人可以通过消息传递共享信息。 *任何位于

*这称为面对面通信模型，在研究与移动机器人相关的问题（包括探索 [5, 6] 和

某个节点上的机器人都知道该节点的度，以及与该节点对应的每条边的端口编号。因此，如果某个机器人需要通过特定端口号离开当前节点，它就可以这样做。此外，每当机器人从节点 u 移动到另一个节点 v 时，它都会获知进入节点 v 所经过的端口号。

我们的算法以同步轮次推进，每一轮中，机器人按顺序执行以下步骤：(i) 共址机器人可以交换消息；(ii) 机器人可以根据可用信息进行计算；(iii) 机器人可以根据步骤 (ii) 中的计算结果，从当前节点沿一条边移动到某个相邻节点。我们还假设所有机器人同时开始执行算法，即从同一轮次开始。算法的时间复杂度以机器人完成任务所需的同步轮次数量衡量。我们还研究运行该算法所需的每个机器人的内存量。

问题：距离-2-分散（D-2-D）： 给定一组任意放置在具有 n 个节点和 m 条边的端口标记图 G 中的 $k \geq 1$ 个机器人，机器人需要在算法结束时达到一种配置，使每个机器人都定居在某个节点，并满足以下两个条件：(i) 任意两个相邻节点不能被已定居机器人占用；(ii) 只有当不存在其他满足条件 (i) 且可供机器人定居的未占用节点时，机器人才能定居在已有已定居机器人的节点上

这些条件确保，任意一对已定居机器人的距离至少为 2，除非它们定居在同一个节点。因此，存在已定居机器人的节点构成图的独立集。并且，使用 $enough$ 个机器人，我们可以得到一个极大独立集。

我们的贡献： 我们在第3节中解决了任意图上以根节点† 为初始配置的 D-2-D 问题，所需轮次为 $2\Delta(8m-3n+3)$，每个机器人使用 $O(log\Delta)$ 内存。所有已定居机器人都会终止，即使它们对图参数 m、n 或 Δ 都没有任何全局知识。在第4节中，我们给出了 D-2-D 问题在机器人内存不超过 $O(log\Delta)$ 时所需轮次数的 $\Omega(m\Delta)$ 下界。此外，如果 $k \geq n$，则可以保证存在已定居机器人的节点构成一个极大独立集；这本身也可能是分布式计算与移动机器人领域中一个值得研究的课题。

1.2 相关工作

分散问题与我们的问题最为相关，因为我们考虑的模型与用于解决分散问题的模型相似。Augustine 等人针对有根配置在 [2] 中提出了分散问题。他们证明，对于任何确定性算法，要在图上实现分散，机器人所需的内存为 $\Omega(\log n)$。在任意图上执行分散的任何算法，其下界为

分散 [2, 13]

†所有机器人最初都放置在图的同一个节点上的配置

Ω(D)，其中 D 是图的直径。对于有根图，在已知 m、n 的情况下，他们提出了一种算法，机器人需要 $O(\log n)$ 内存，并可在 $O(m)$ 轮次内完成分散[2]。Kshemkalyani 等人[11]将运行时间下界改进为 Ω(k)，其中 $k \leq n$。他们为同步系统设计了一种算法，使用每个机器人 $O(k \log \Delta)$ 比特，可在 $O(m)$ 轮次内解决分散问题。然而，对于异步系统，他们设计的算法需要 $O(max(\log\ k, \log\ \Delta))$ 比特内存，时间复杂度为 $O((m-n)k)$。之后，Kshemkalyani 等人[9] 显著改进了这一结果，并为同步设置下的任意图中的分散问题提供了一种确定性算法，其运行时间为 $O(min(m, k\Delta)\ \cdot \log\ l)$ 轮次，其中 $l \leq \frac{k}{2}$，且每个机器人使用 $O(\log n)$ 比特内存。他们的直观思路是并行执行 DFS 遍历，以缩短时间。机器人需要知道 m、n、k 和 Δ。随后，Shintaku 等人提出了一种不需要此类全局知识 [20]的分散算法。该算法使用每个机器人 $\Theta(\log(k+\Delta))$ 比特内存，可在 $O(min(m, k\Delta)\ \cdot \log\ l)$ 轮次内解决任意图上的分散问题。最近，Kshemkalyani 等人 [13] 为常数度任意匿名图提出了一种在时间和内存方面均达到最优的算法。他们提出的算法使用每个机器人 $\Theta(\log(k+\Delta))$ 比特，可在 $O(min(m, k\Delta))$ 时间内解决分散问题；与此前已知的最佳算法相比，该算法将时间界改进了 $O(\log l)$，其中 $l \leq \frac{k}{2}$，并在渐近意义上匹配单源 DFS 遍历界[9]。他们扩展了 [9]的思路，让规模较大的 DFS 遍历包含规模较小的 DFS 遍历，从而避免反复访问已被包含的遍历所经过的节点。

从某种意义上说，D-2-D 也与散布或均匀分布问题有关。散布研究主要针对网格[3] 和环[7, 19] ，不过使用的是匿名机器人。最后，由于在某些情况下，我们的算法会形成极大独立集，因此我们引用了以下关于使用可移动实体形成极大独立集的研究，尽管该研究采用了更强的模型假设。Vamshi 等人提出了这样一个问题：如何在任意连通图中，使用 my-opic 发光机器人[18] 求出极大独立集（MIS）；这些机器人预先掌握 Δ、O（$\log \Delta$）比特的持久内存，并且至少具有 3 跳的可视范围。作者还使用颜色表示不同状态，并在半同步和异步调度器下开展了研究。

1.3 D-2-D 与分散：挑战

在此前关于分散问题的研究中，算法使用深度优先搜索（DFS）遍历，而机器人 [2, 4, 13] 的内存有限。从任意有根配置实现分散的关键思路如下：在起始节点，也就是根节点，最小编号的机器人定居下来，其余未定居机器人离开根节点，前往某个邻居节点。这个未定居机器人组中编号最小的机器人在此定居。每一轮中，当机器人组首次访问某个未占用节点时，编号最小的机器人便在该节点定居。节点上的已定居机器人表示该

节点已被该机器人组访问过。借助这一关键思路，机器人能够对匿名图进行深度优先搜索遍历，因为已驻留节点有助于跟踪环和已访问节点，从而确保机器人安全地分散到匿名图中。例如，如果机器人组在探索过程中从节点 u 访问节点 v，并发现 v 处已有已定居机器人，那么按照深度优先搜索，该机器人组必须回溯到 u，并探索 u 的其他未探索邻居（如果有）。每当找到一个空节点，机器人组中编号最小的机器人便在此定居。其余未定居机器人继续进行深度优先搜索遍历。然而，要成功回溯，机器人需要记住已经探索过的路径，这会带来较高的内存需求。为避免这种情况，该算法要求每个已定居机器人存储从其定居节点回溯所需的信息。请注意，机器人组从不会从没有机器人定居的节点回溯。具体而言，每个已定居机器人都会记住父指针，即它进入定居节点时所使用的端口号。回溯中的机器人组可以利用这个父指针，成功从该节点回溯。因此，每个机器人的内存需求为 $O(\log \Delta)$，即可实现分散。

在 D-2-D 中，主要目标是实现分散，使任意两个相邻节点上都不会有已定居机器人。换句话说，任意两个已定居机器人之间的距离必须至少为 2。为此，我们可能会面临以下挑战。

- 由于任意已定居机器人的一跳邻域内都没有已定居机器人，因此很难存储这些邻居节点的父指针信息，并在回溯时使用。这可能导致较高的内存需求。

- 当且仅当节点 u 的任何邻居上都没有已定居机器人时，机器人才能在该节点定居。由于最大度 Δ 可能很大，这可能导致较高的时间复杂度。

2 预热：每个机器人使用 O（Δ对数 Δ）内存的 D-2-D

本节将非正式地讨论一种有根 D-2-D 的直接解法，不考虑每个机器人的内存需求或时间复杂度。我们的算法基于深度优先搜索遍历（但做了些许修改），用于解决我们在第1节中讨论过的分散问题。随后，我们将在第3节中对这一解法进行改进。

在解决分散问题时，向前探索过程中遇到已定居机器人，意味着移动群组中的未定居机器人存在环，因此会借助已定居机器人存储的父指针进行回溯。为解决我们的问题，我们作如下修改。由于在我们的 D-2-D 问题中，群组可能需要从未占用节点回溯，因此也需要存储未占用节点的父指针。请注意，任何已占用节点的所有相邻节点都必须是未占用的。我们指示

每个已定居机器人记住其所有邻居（包括其自身）的父端口。基本上，每个已定居机器人在其邻居处都充当 $virtuallysettled$ 机器人。虚拟已定居的概念是指，尽管该节点上没有已定居机器人，但访问群组中的机器人仍无法占据该节点。不过，为了存储并提供已存储的父指针，每个已定居机器人都必须在移动群组到达其某个相邻节点时与该机器人群组会合。

为实现这一点，每个定居机器人都会从所在位置出发，在相邻节点之间往返移动。具体来说，设机器人 r 在第 T 轮定居于度为 δ 的节点 u。它会在第 $T+1$ 轮访问 $u(0)$，在第 $T+2$ 轮返回 u，在第 $T+3$ 轮访问 $u(1)$，依此类推。在访问完 $u(\delta-1)$ 后，它会再次访问 $u(0)$。已定居机器人 r 只有在某个节点（例如 $v=u(p)$）遇到未定居机器人组时，才会停止；假设相遇发生在第 T' 轮。它会在 $u(p)$ 与该群组待在一起，直到群组离开 $u(p)$，例如在第 T'' 轮。随后，r 会在第 $T''+1$ 轮返回 u，并再次开始如前所述逐一访问其邻居。

请注意，仅靠这一点无法解决问题，因为移动机器人群组可能会到达某个已占用节点的相邻节点，但此时相应的已定居机器人可能正在访问另一个邻居。为解决这一问题，算法指示移动群组在每个节点 v 等待 2Δ 轮，以确保移动中的已定居机器人必定能在这段时间内与该群组相遇。为简单起见，假设所有机器人都知道 Δ.[‡] 如果 v 已被占用，那么已定居机器人必定能在 2 轮内与该群组相遇；否则，如果 v 是某个已占用节点的邻居，那么作为虚拟已定居机器人的已定居机器人必定能在 2Δ 等待时间内与该群组相遇。

现在我们给出该算法。如果群组处于前向探索阶段，则可能出现以下情况。

- 群组遇到至少一个虚拟已定居机器人，并发现遇到该群组的虚拟已定居机器人中没有任何一个持有指向该节点的父指针。于是，群组明白自己是第一次访问该节点，并在为每个虚拟已定居机器人提供父指针后，继续在前向探索阶段进行深度优先搜索遍历。由于所有遇到该群组的已定居机器人都会与该群组一起等待，直到群组离开，因此这是可行的。

- 群组遇到至少一个虚拟已定居机器人，并发现至少有一个虚拟已定居机器人带有指向该节点的父指针。于是，群组明白该节点此前已经被探索过，随后便回溯。

- 群组发现该节点已被占据，便进入 $backtrack$ 阶段，并借助已定居机器人存储的父指针回溯。
- 群组看不到任何已定居或虚拟已定居机器人，随后群组中的最小 ID 机器人在那里定居。

[‡]在下一节中，我们将在主算法中去除这一假设。具体思路是：每当未定居机器人组访问一个新节点时，每个未定居机器人都会更新自身迄今为止见到的最大度，其值为 Δ。

如果群组处于回溯阶段，则它必须在该节点遇到至少一个虚拟已定居机器人或已定居机器人。群组通过查看父指针以及它刚刚经由其回溯到该节点的端口，检查与该节点关联的所有端口是否已经被探索。具体决策如下。

- 如果所有端口都已探索，它就继续回溯。
- 如果还有端口尚未探索，它就切换到前向探索阶段。

图1展示了一个示例中算法的执行过程。

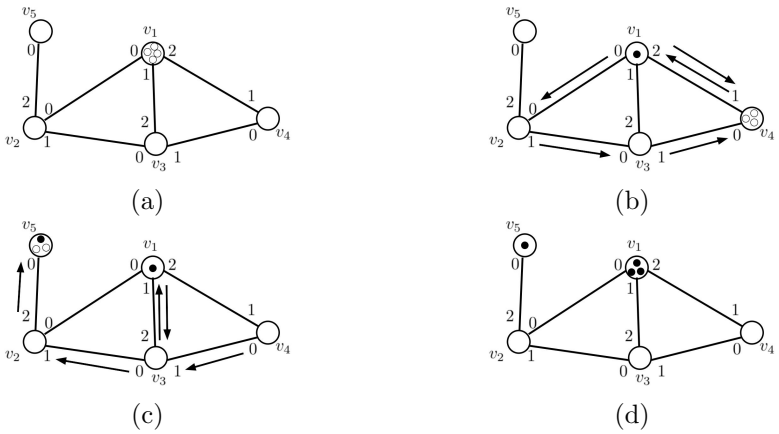


图1：（a）四个机器人位于 v_1 的初始配置。（b）一个机器人在 v_1 定居，其余未定居机器人则通过深度优先搜索遍历 v_2、v_3、v_4。它们都不会在这些节点中的任何一个定居，因为在这些节点上，已在 v_1 定居的机器人充当虚拟已定居机器人。在 v_1 定居的机器人会在所有相邻节点处为群组维护父指针。由于 v_1 上有一个已定居机器人，机器人便从该节点回溯。（c）未定居机器人继续进行深度优先搜索遍历；群组到达 v_5 时，一个机器人在此定居。（d）其余机器人继续遍历，在 v_1 完成遍历后再次遍历，最终达到目标配置。

为了存储每个邻居的父指针，每个已定居机器人需要$O(\Delta\log\ \Delta)$内存，即每个端口号需要 $\log\ \Delta$ 内存。这导致内存需求很高，而移动群组在每个新探索的节点处等待 2Δ 轮次，也使算法的运行时间很长。我们在讨论第1节中的挑战时提到了这两个问题。该算法的运行时间为 $2\Delta(4m-2n+2)$。这是因为我们只需运行用于分散的深度优先搜索遍历，该遍历需要 $4m-2n+2$ 时间，做法与 [2, 9] 中相同：假设每个已占用节点的所有未占用邻居节点都被虚拟占用。此外，我们将所有必要信息保存在已定居机器人中。完成

一次深度优先搜索遍历后，如果仍有一些机器人尚未定居，它们就会在根节点定居。请注意，由于已定居机器人会持续移动，因此机器人能够终止是理想的。不过，这里我们不讨论算法的终止，但可以像主算法在第3节中那样实现。

3 D-2-D 与终止： $O(\log \Delta)$ 每个机器人所需内存

本节介绍一种算法，它改进了第2节所讨论算法的内存需求，并且无需全局知识即可使机器人终止。首先，我们给出一个高层次的思路。该思路是不采用通常的深度优先搜索遍历，因此已定居机器人无需为其所有未占用邻居节点维护父指针。每个已定居机器人只存储其一个相邻节点对应的父指针，以及其定居节点对应的父指针。这限制了从最后一个定居机器人到未定居机器人组当前所在节点的路径长度。如果这条路径很长，那么在最坏情况下，路径上的所有节点都可能保持未占用，因为它们都可能是某个特定已占用节点 v 的邻居；在这种情况下，定居在 v 的机器人需要记住该路径上未占用节点的所有父指针。我们将向前探索期间的遍历深度限制为从机器人最后定居的节点起 2。现在，问题是，仅对应一个邻居的信息为何足够？与前一种算法类似，每个已定居机器人在定居后的后续轮次中逐一访问其每个邻居；与此同时，未定居机器人组等待 2ϕ 轮，其中 ϕ 是未定居机器人组到达当前节点之前观察到的最大度，以检查其一跳邻域内是否存在已定居机器人。与第2节所述算法类似，未定居机器人组到达新节点 u 时会通过 $max\{\phi, \delta_u\}$ 更新 ϕ 的值，因为它们事先并不知道图的最大度。未定居机器人组维护一个计数器 $r_i.dist$ ，从上一个遇到的已定居机器人开始计数，用于记录与该已定居机器人的距离。变量 $r_i.dist$ 可以取值 0、1 或 2。未定居机器人组根据 $r_i.dist$ 的值、等待期内访问该群组的已定居机器人数量，以及这些访问机器人的 ids，决定向前探索还是回溯。因此，任何机器人的状态可以是$explore$或 $backtrack$。机器人还维护另一个变量 $r_i.stage$ ，当达到 D-2-D 配置且终止阶段开始时，该变量取值 2。为了终止算法，已定居机器人维护变量$r_i.count$、 $r_i.count'$和 $r_i.act\ settled$ ，其说明见表1。

每当未定居机器人组从一个包含编号最小的已定居机器人的节点 u 移动到其某个邻居时，设该邻居为 $u\,(p)$ ，也就是说，未定居机器人组位于 $r_i.dist = 1$ （ u，p），且其状态为 $explore$ ，并且除 r_j 外，没有其他已定居机器人在 $u\,(p)$ 与该组相遇，那么就必须由 $u\,(p)$ 的父指针

由已定居机器人 r_j （$r_j.virtualparent$ ，定义见表1）存储，之后该群组继续探索。从已定居机器人的角度来看，它会存储自己的父指针（$r_j.parent$ ，定义见表1），以及仅一个邻居的父指针 u（p）。其思路是， r_j 可以存储对应于另一个邻居 u（q）的父指针，例如，当 q 是从 r_j 已定居的节点探索的最后一个端口时。否则，如果一个或多个编号小于 $r_j.id$ 的已定居机器人在 u（p）与该群组相遇，那么该群组会与已定居机器人 r_j 一同回溯到节点 u ，而在这种情况下， r_j 无需存储 u（p）的父指针。所有与未定居机器人组相遇的已定居机器人，包括 r_j ，每次都会递增其 $count$ 变量的值。请注意，为避免已定居机器人不必要地递增 $count$ 变量的值，当已定居机器人在往返移动过程中于某个节点遇到未定居机器人组时，它们会与该组一同等待，直到该组离开该节点。

当 $r_i.dist = 2$ 群组到达节点（例如 v 离开 $u(p)$ 后到达的节点）时，若 r_j 已定居，则只有在该节点没有相邻的已定居机器人时，机器人才会在此定居。在这种情况下，新定居的机器人会保留父指针，而群组继续前进。否则，如果有一个或多个已定居机器人正在访问节点 v，群组就会回溯，因为之后可以通过其他已定居机器人（其标识符可能小于 r_i）来探索该节点。例如，如果位于节点 u 、处于 $dist = 2$ 的未定居机器人群组正在经历 2ϕ 轮的等待期，而在等待期间，机器人 r_m 到访，那么群组不仅会明白没有机器人能在该节点定居，也会明白可以从有已定居机器人 r_m 的节点探索此节点。换言之，如果未定居机器人群组位于距离有已定居机器人的节点 $dist = 2$ 处，且处于 $explore$ 状态，而某个已定居机器人 r_m 在等待期内到访该群组，这就意味着从 r_m 到该节点存在直接连接。这意味着可以从 r_m 已定居的节点探索当前节点。因此，在任何情况下，这里的任何已定居机器人 r_j 都不会保留距离至少两跳的节点的父指针。所有遇到未定居机器人群组的已定居机器人 r_j ，每次与该群组相遇时都会增加 $r_j.count$ 的值。这些步骤构成了算法的阶段 1。最后一个机器人定居后，便开始下一阶段，即终止阶段。最后一个已定居机器人从根节点出发，并沿着与上一段所述类似的路径前进。在此终止阶段，最后一个机器人 r_L 扮演阶段 1 中未定居机器人群组的角色，而在距离-2-分散过程中最初定居的机器人 r_j 则按照它们在上一阶段定居的相同顺序设置 $r_j.act\ settled = 1$ 。因此，终止阶段是阶段 1 的一个副本。在遍历过程中，每当 $act\ settled$机器人 r_j 遇到最后一个已定居机器人 r_L 时， r_j 就会增加 $r_j.count'$ 的值。当 $r_j.count$ 的值达到 $r_j.count'$ 时，机器人 r_j 便终止。 如此一来，机器人即可终止，并实现带终止的距离-2-分散。

变量	描述
$r_i.parent$	此变量包含节点 u 的父端口 其中 r_i 在阶段 1 已定居，或 $act\ settled$ 在阶段 2 已定居。 否则， $r_i.parent = -1$。
$r_i.portentered$	机器人 r_i 进入当前节点所经过的端口 节点。初始时，所有机器人均为 $r_i.portentered = -1$ 。
$r_i.virtualparent$	u（p）的父端口，其中 p 是 从 r_i 所定居的节点探索的最后一个端口。 初始时，所有机器人均为 $r_i.virtualparent = -1$ 。
$r_i.dist$	每个未定居机器人 r_i 都会记录与 其在最近一次探索过程中遇到的已定居机器人的距离 遍历。根据我们的算法， $r_i.dist \leq 2$。对于 每个已定居机器人， $r_i.dist = 0$ 若它位于 它定居的节点，否则 $r_i.dist = 1$。
$r_i.special$	在某个节点 u 定居的机器人 r_i ，例如，会更新 $r_i.special = 1$ ，但仅当未定居机器人组 位于 u ，且带有 r_i ，并将经过其中一个 在 u 的 $explore$ 状态下的相邻边。 对于其他已定居机器人， $r_i.special = 0$ ；对于任何 未定居机器人， $r_i.special = -1$。
$r_i.settled$	如果 r_i 是阶段 1 中的已定居机器人，则取值为 1，否则 取值为 0。
$r_i.count$	如果 r_i 是阶段 1 中的已定居机器人，则此变量 统计 r_i 与群组相遇的次数 未定居机器人组。
$r_i.stage$	此变量可取值 1 或 2，其中 $r_i.stage = 1$ 表示算法的阶段 1 $r_i.stage = 2$ 表示阶段 2。
$r_i.act_settled$	若 r_i 在阶段 2 定居，则取值 1，否则 取值为 0。
$r_i.count'$	在阶段 2 中，如果 r_i 是一个 $act_settled$机器人，则此 变量统计 r_i 与.....相遇的次数 机器人 r_L 与 $r_L.terminate = 1$。

表1：变量说明

3.1 算法

本节将介绍一种算法，机器人可以使用每个机器人具有 $O(log\Delta)$ 内存的配置来运行该算法，从而实现有根距离-2-分散，并在 $2\Delta(8m -$ 中终止

$3n + 3$) 轮次。

考虑任意一个图 G ，并设有 k 个机器人最初放置在 G的一个节点上，记为 u。每个机器人 r_i 维护变量 $r_i.parent$、 $r_i.settled$、 $r_i.dist$、$r_i.special$、 $r_i.portentered$、 $r_i.virtualparent$、 $r_i.count$、 $r_i.stage$、 $r_i.act$ $set\bar{t}led$ 和 $r_i.count'$。我们已在表1中详细定义这些变量。此外，每个未定居机器人还维护自己的状态，可以是 $explore$或 $backtrack$。处于 $explore$状态时，它通过从当前节点沿计算出的端口号移动来进行向前探索；处于 $backtrack$ 状态时，它从已定居机器人处获取父指针，并沿该端口回溯。机器人还维护变量 ϕ ，其初始值为 δ(u)，其中 u 为根节点。每个未定居机器人在发现某个节点的度大于 ϕ的当前值时，都会相应地更新 ϕ 。但是，已定居机器人定居后不会修改 ϕ 。注意，未定居机器人组中的每个机器人具有相同的 ϕ值；此外， $\phi \leq \Delta$是图的最大度。具体来说，当未定居机器人组到达一个新节点（记为 v）时，组中的每个机器人都会将 ϕ 的值更新为 $max\{\delta(v)\ \phi\}$。我们的算法分两个阶段运行：阶段 1 和阶段 2。在阶段 1，机器人实现 D-2-D；在阶段 2，机器人终止。未定居机器人组分多个阶段运行该算法，每个阶段包含 2ϕ轮次。由于每个未定居机器人对其 ϕ 值达成一致，该组会在同一轮次开始和结束每个阶段。

我们的算法首先在阶段 1 结束时，将最小编号的未定居机器人（假设为 r_1）安置在根节点 u 。已定居机器人 r_1 更新 $r_1.parent \longleftarrow 1$、 $r_1.settled \leftarrow 1$ 和 $r_1.dist \leftarrow 0$。其余未定居机器人将它们的 $state$ 更新为 $explore$ ；由于它们与已定居机器人 r_1同处一处，变量 $r_1.special$ 更新为 1。未定居机器人将端口号更新为 $r_i.portentered = (r_i.portentered + 1) mod \delta$，其中 δ 是节点u的度。未定居机器人沿递增后的端口号移动；在此例中，端口号为 0。它们移动到相邻节点 u(0)，并更新 $r_i.dist = 1$。至此，未定居机器人的阶段 1 结束。到达该节点后，未定居机器人更新 ϕ ，并等待 2ϕ 轮次。由于这是一个有根配置，且 r_1 是第一个定居的机器人（目前与未定居机器人组同处一处），在等待2ϕ 轮次期间，没有其他已定居机器人会访问该群组。这意味着，如果存在未探索边，则可以进一步探索当前节点，即 u(0)的度大于 1。已定居机器人 r_1更新 $r_1.count \leftarrow 1$。设 p_1 为机器人进入 u(0)所经过的端口。已定居机器人 r_1 更新 $r_1.virtualparent \leftarrow p_1$，通过端口 p_1 返回根节点 u ，并更新 $r_1.special = 0$。未定居机器人更新 $r_i.portentered = (p_1 + 1) mod (\delta(u(0)))$，其中 δ(u(0))是节点 u(0)的度。未定居机器人在 $explore$模式下沿 $portentered$ 的更新后的值移动，并在满足 $r_i.portentered \neq r_1.virtualparent$时更新 $r_i.dist = 2$ 。否则，未定居机器人在 $backtrack$ 模式下沿 $portentered$ 的更新后的值移动，并更新$r_i.dist = 0$。至此，未定居机器人组的阶段 2 结束。基本上，未定居机器人组仅在每个阶段结束时移动。

当未定居机器人组到达与 u 相距 2 的新节点后，它们会更新 ϕ ，并等待 2ϕ 轮次；此时，已定居机器人 r_1 现在拥有

$r_1.special = 0$，并持续逐一遍历 u 的每个端口。在此等待期间，如果任何已定居机器人 r_j 到访未定居机器人组，它们就会使用 $r_i.portentered$ 回溯到前一个节点。然而，与该未定居机器人组相遇的已定居机器人 r_j 会与该组一同等待，直到该组离开该节点，并增加 $r_j.count$ 的值。不过，如果没有已定居机器人到访该组，则该组中标识符最小的机器人会在此节点定居。

因此，未定居机器人的每个阶段都对应一次边遍历，且每个阶段需要 2ϕ 轮次。已定居机器人不参与阶段，它们要么与未定居机器人组一起等待，要么继续来回遍历。因此，我们算法的阶段 1 的时间复杂度取决于未定居机器人组在最后一个机器人（即最大 ID 的机器人，设为 r_L）定居之前经历了多少个阶段。已定居和未定居机器人根据以下情况决定如何行动。首先说明已定居机器人的行动。

- 如果已定居机器人 r_j 的值为 $special = 0$，则它会继续逐一访问其邻居，并相应地不断修改 $r_i.dist$。每当已定居机器人遇到未定居机器人组时，它就会增加 $r_j.count$ 的值，并与该群组一起等待，除非它离开该节点。之后，已定居机器人会继续逐一访问其邻居。
- 如果已定居机器人 r_j 的值为 $special = 1$，则在下一轮中，它会与未定居机器人组一起移动到相邻节点，并与该群组一起等待。已定居机器人 r_j 会增加 $r_j.count$的值。在此等待期内，如果有任何 ID 小于 $r_j.id$ 的已定居机器人前来访问，那么 r_j 不会为该节点存储父指针，并会在群组离开后返回其原始位置。然而，如果没有 ID 小于 $r_j.id$ 的已定居机器人前来访问未定居机器人组，那么已定居机器人 r_j 会更新 $r_j.virtualparent = r_j.portentered$。

已定居机器人的伪代码见算法1。

现在我们来看未定居机器人如何行动。当未定居机器人组在某个阶段结束时到达一个节点，那么它们将在下一阶段结束时采取什么行动，将根据以下情况决定：

- 当 $r_i.dist = 1$ 和 $state = explore$ 在某阶段开始时
  - 如果没有任何携带 $special = 0$ 的已定居机器人访问未定居机器人组，且访问的已定居机器人的标识符小于携带 $r_j.special = 1$ 的 r_j 的标识符，那么该群组会更新 $r_i.portentered$ =$(r_i.portentered + 1) mod \delta$。注意，每当未定居机器人组拥有 $r_i.dist = 1$ 和 $state = explore$时，必定有一台携带 $r_j.special = 1$ 的已定居机器人 r_j 与该群组同处一处。如果 $r_i.portentered = r_j.virtualparent$，则未定居机器人组回溯至前一个节点。否则，如果 $r_i.portentered \neq r_j.virtualparent$，那么未定居机器人组会经由已更新状态 $explore$ 中的端口号离开当前节点。

算法1： 针对每个已定居机器人 r_j 的算法

```
如果 r_j.id is the largest among the group from which it settled，则
    设置 r_j.terminate = 1
否则
    如果 r_j.special = 0，则
        通过其所有端口进行往返移动，并更新 r_j.dist
          根据其位置将值设为 0 或 1。
        如果 the settled robot r_j meets the group of unsettled robots，则
            与未定居机器人组一起等待，直到该组离开该节点
            将 r_j.count 的值加一
            如果尚未位于原处，则移动到其已定居时所在的原始位置，
            并恢复往返移动
        否则，如果 r_j meets any robot r_i with r_i.stage = 2，则
            如果尚未回到其已定居时的原始位置，则移动到该位置，并
            等待，直到它遇到具有 r_L.terminate = 1 的机器人 r_L
            不要递增 r_j.count 的值
            设置 r_j.stage = 2
            如果 the robot r_L with r_L.terminate = 1 visits this node where r_j is waiting
              then
                设置 r_j.act settled = 1, r_j.parent = r_L.portentered, r_j.special = 1
    如果 r_j.special = 1，则
        沿着更新后的值为.....的路径，随未定居机器人组一起移动
        r_i.portentered ，并等待该组离开。
        将 r_j.count 的值递增
        如果 no settled robot visits with id lower than r_j.id，则
            设置 r_j.virtualparent = r_i.portentered
            通过 r_j.portentered移动到原始位置
        else
            通过 r_j.portentered移动到原始位置
        更新 r_j.special = 0
```

– 如果至少有一台携带 $special = 0$ 且标识符小于携带 $r_j.special = 1$ 的 r_j 的标识符的已定居机器人访问未定居机器人组，那么该群组会经由 $r_i.portentered$ 回溯。

- 当 $r_i.dist = 2$ 和 $state = explore$ 在阶段开始时

  – 如果在 2ϕ 轮次内没有已定居机器人访问未定居机器人组，则该组中标识符最低的机器人在此节点定居。– 如果有一个或多个已定居机器人访问未定居机器人组，则该组通过 $port\ entered$ 回溯。

然而，当未定居机器人组处于 $state = backtrack$ 时，该组会到达一个之前访问过的节点。因此，在 $backtrack$ 状态下访问的节点上要么有一个已定居机器人，要么其一跳邻居中的某个已定居机器人为它存储了虚拟父节点。回溯到当前节点后，未定居机器人组会根据以下情况决定继续探索还是回溯。

- 当 $r_i.state = backtrack$ 处于一个阶段的开始时

– 如果所有端口都已被探索，即（$r_i.portentered + 1$）$mod\delta$ 等于 $r_m.parentpointer$ ，其中 r_m 是该节点上的已定居机器人；或者等于 $r_j.virtualparent$ ，其中 r_j 是为其邻居存储了父指针的已定居机器人。请注意，这些信息可以在等待时间内交换。在这种情况下，未定居机器人组会通过该节点的 $r_m.parentpointer$ 或 $r_j.virtualparent$ 进行回溯。

– 如果 $(r_i.portentered + 1)mod\delta$ 不等于该节点的 $r_i.parentpointer$，那么未定居机器人组会将其 $state$ 更改为$explore$，并穿过 $r_i.portentered$移动。

未定居机器人的伪代码见算法 2。

利用这一点，我们实现了 D-2-D 配置，阶段 1 随之完成。接下来开始阶段 2。ID 最大的机器人（设为 r_L）被称为最后一个定居机器人；它定居后会设置 $r_L.stage = 2$。为了终止算法， r_L 会从根节点开始，按照阶段 1 中所述重新进行类似的遍历。此时会出现两种情况：机器人 r_L 定居在根节点以外的节点（这种情况下，它会按下文所述前往根节点），或者 r_L 直接定居在根节点。无论哪种情况，r_L 都会在根节点设置 $r_L.terminate = 1$。从根节点出发， r_L 会再次模仿阶段 1，充当未定居机器人组，以帮助其余已定居机器人终止。最后， r_L 到达阶段 1 结束时定居的节点后终止。请注意，如果 r_L 回溯至根节点途中，有些已定居机器人可能会在路径上遇到 r_L ，因为它们仍在继续往返移动，但没有已定居机器人会增加各自的计数变量，因为 r_L 已不再是未定居机器人。

- 当最后一台机器人 r_L 在根节点以外的节点 u_l 定居时

  – 机器人 r_L 沿着 $r_L.portentered$回溯。它沿着父指针继续回溯，直至到达根节点。到达根节点后，机器人 r_L 开始模仿未定居机器人组的阶段 1。

- 当剩余机器人在根节点定居，且 r_L 是群组中 ID 最大的机器人时

  – 机器人 r_L 开始模仿未定居机器人组的阶段 1。

当一台具有 $r.settled = 1$ 的机器人 r_i 遇到具有 $r_L.terminate = 1$ 的 r_L 或任何具有 $r_j.stage = 2$ 的机器人r_j 时，它会明白阶段 2 正在进行，更新 $r_i.stage = 2$，返回其原始位置，并等待具有 $r_L.terminate = 1$ 的 r_L 到达那里。当一台并非 r_L 的机器人 r_j 首次遇到具有 $r_L.terminate = 1$ 的 r_L 时，可能出现以下两种情况。如果这次相遇发生在 r_j于阶段 1 定居的节点上，它会明白自己必须在此等待，直到 r_L 到达这里；它会表现得像 r_j 也是一台与 r_L在一起的未定居机器人。然后， r_j 会更新 $r_j.act\ settled = 1$ ，仿佛自己在本阶段结束时定居，并开始遵循阶段 1 中已定居机器人的算法。唯一的区别是，现在它改为递增 $r_j.count'$

算法 2:每个未定居机器人的算法 r_i

初始化 $r_i.portentered = -1$, $r_i.dist = 0$, $r_i.state = explore$, $r_i.special = -1$, $r_i.settled = 0$, $r_i.virtualparent = -1$, $\phi = 0$, $r_i.terminate = 0$, $r_i.stage = 1$
对于 $phase = 0$ **执行**
    最小 ID 机器人r_j 等待 2ϕ 轮次后定居在节点上
    设置 $r_j.parent = -1$
    设置 $\phi = \delta(u)$
    $r_i.portentered = (r_i.portentered + 1)$模$\delta$
    通过 $r_i.portentered$ 移动
    $r_i.dist$=$r_i.dist + 1$
对于 $phase > 0$ **执行**
    如果 $r_i.dist = 0$ **则**
        $r_i.dist = r_i.dist + 1$
    如果 $r_i.dist = 1$ *and* $r_i.state = explore$ **则**
        等待 2ϕ 轮次
        如果 *any settled robot visits with* $r_j.special = 0$ *and* $r_j.id$ *is smaller than the settled robot already present with the group* **则**
            $r_i.state = backtrack$
            更新 $\phi = max\{\phi, \delta(u)\}$
            通过 $r_i.portentered$移动
            将 $r_i.dist$ 值递减至 0
        else
            $r_i.state = explore$
            更新 $\phi = max\{\phi, \delta(u)\}$
            $r_i.portentered = (r_i.portentered + 1)$模$\delta$
            如果 $r_i.portentered = r_j.virtualparent$ **则**
                $r_i.state = backtrack$
                穿过 $r_i.portentered$ 移动
            else
                穿过 $r_i.portentered$ 移动
                将 $r_i.dist$ 递增至 2
    如果 $r_i.dist = 2$ *and* $r_i.state = explore$ **则**
        等待 2ϕ 轮次
        更新 $\phi = max\{\phi, \delta(u)\}$
        如果 *no settled robot visit the group of unsettled robots* **则**
            编号最小的机器人 r_i 停留在该节点上，并设置 $r_i.parent = r_i.portentered$ 和 $r_i.special = 1$
            每个未定居机器人执行 $r_i.portentered = (r_i.portentered + 1)mod\delta$ 并且 $r_i.dist = 0$
            如果 $r_i.portentered$ = *parent pointer of the settled robot at the current node* **then**
                $r_i.state = backtrack$
            经由 $r_i.portentered$ 移动
        else
            $r_i.state = backtrack$
            经由 $r_i.portentered$ 移动
    如果 $r_i.state = backtrack$, **则**
        等待 2ϕ 轮次
        将 $r_i.dist$ 减 1
        $r_i.portentered = (r_i.portentered + 1)$模$\delta$
        如果 *the settled robot* r_j *has* $r_j.parent = -1$ **则**
            如果 $r_i.portentered = 0$ **则**
                未定居机器人在根节点定居
            else
                $r_i.state = explore$
                穿过 $r_i.portentered$移动
        else
            如果 $portentered = parent$ *or* $virtualparent$, **则**
                $r_i.state = backtrack$
                穿过 $portentered$移动
            else
                $r_i.state = explore$
                穿过 $portentered$移动
                将 $r_i.dist$ 递增1

的 $r_j.count$。否则，如果此次相遇发生在r_j 在阶段 1 定居的节点的相邻节点上，它会改变 $r_j.stage = 2$，返回其原始位置，并在那里等待 r_L 与 $r_L.terminate = 1$ 一起到达。

上述描述表明，阶段 2 中的任何已定居机器人 r 只有在成为 $act\ settled$后才会递增其变量 $r.count'$，也就是说，只有在 r 变为已定居并开始递增其$r.count$的阶段 1 的相应轮次之后才会递增。由于 r_L 按照阶段 1 中未定居机器人的算法作为未定居机器人运行，且其标识符最大，因此 r_L 最终将在阶段 2 再次定居，并终止。到那时，所有已定居机器人的 $count$ 值都将与其 $count'$ 值匹配，并且所有机器人都将终止。下面，我们以形式化方式给出阶段 2 的算法。

当携带 $r_L.terminate = 1$ 和 $r_L.stage = 2$ 的机器人 r_L 到达根节点 u 时，它会初始化 $\phi = \delta(u)$ 的值，并开始遍历，以终止已定居机器人。与上文所述阶段 1 中针对未安置机器人群组的算法类似，每当 r_L 到达一个节点（例如 v）时，它会在该节点等待 2ϕ 轮次。此外，它会将 ϕ 的值更新为 $max\{\phi, \delta(v)\}$。到达新节点后，将根据以下情况作出决定：

- 当 $r_L.dist = 1$ 和 $r_L.state = explore$

  – 如果任何已定居机器人携带 $r_j.act\ settled = 1$ 到访，且 $r_j.special = 0$ 和 $r_j.id$ 小于已在场的 $act\ settled$ 机器人的标识符，并携带 r_L，那么 r_L 会通过 $r_L.portentered$ 回溯– 如果任何已定居机器人携带 $r_j.act\ settled = 1$ 到访，且携带 $r_j.special = 0$，但 $r_j.id >$ 已在场的 $_act\ settled$ 机器人的标识符，且该机器人携带 r_L，或者没有携带 $act\ settled = 1$ 的机器人到访，那么 r_L 会被设为探索，并沿着 $portentered$ 的递增值移动

- 当 $r_L.dist = 2$ 和 $r_L.state = explore$

  – 如果没有已定居机器人携带 $r_j.act\ settled = 1$ 访问此处，则表示这是 r_L 定居的原始位置，因此将 $r_L.act\ settled =$ 设为 1。终止 r_L。– 如果有已定居机器人携带 $r_j.act\ settled = 1$ 访问此处，则 r_L 沿着 $r_L.portentered$ 回溯。– 如果节点上存在机器人 r_j，且其携带 $r_j.stage = 2$，但 $r_j.act\ settled =$ 为 0，则将 $r_j.parent$ 设为 $r_L.portentered$ 和 $r_j.special = 1$。

- 当 $r_L.state = backtrack$

  – 如果到达的节点是根节点，且还有一些端口需要探索，即（$r_L.portentered + 1) mod\delta \neq 0$，则将 $r_L.state$ 更新为 $explore$；否则为 r_L，并在此节点终止其余已定居机器人。

  – 如果到达的节点不是根节点，则将 $r_L.portentered$ 与 $r_j.parent$ 或 $r_j.virtualparent$ 的值进行比较。如果所有

端口都已探索，则状态将更改为 $backtrack$；否则， r_L 将通过递增后的 $r_L.portentered$ 值继续探索。

最后一个已定居机器人的算法伪代码见算法4。算法4将算法3用作子程序；算法3是最后一个已定居机器人 r_L 到达根节点后所执行操作的伪代码。

具有 $r_j.settled = 1$ 和 $r_j.act\ settled = 0/1$ 的机器人根据以下情况进行操作。

- 如果 r_j 是一个具有 $r_j.settled = 1$ 但没有 $r_j.act\ settled = 0$ 的已定居机器人，那么当它遇到任何具有 $r_i.stage = 2$ 的机器人 r_i 时，就会返回原始位置并在那里等待。此时，它会意识到终止阶段已经开始。因此，它不会增加 $r_j.count$ 的值。

- 如果 $r_j.act\ settled = 1$ 和 $r_j.special = 0$，那么它会继续在所有邻居之间往返移动，并在每次遇到 r_L 时将 $r_j.count'$ 的值加 1。当 $r_j.count$ 等于 $r_j.count'$ 时，它会在其已定居的原始位置终止。

- 如果 $r_j.act\ settled = 1$ 且 $r_j.special = 1$，那么它会与机器人 r_L 一起通过 $r_L.portentered$ 的更新值移动到相邻节点，并与 r_L一起等待。它将 $r_j.count'$ 的值加 1。现在分为两种情况:
  - 如果没有 $act\ settled$ 编号低于 $r_j.id$ 的机器人前来访问，那么 r_j 设置$r_j.virtualparent = r_L.portentered$ 否则，移动回原来的已定居位置并设置 $r_j.special = 0$

每个已定居机器人 r_j （其 $r_j.act\ settled = 1$）的算法伪代码见算法5。

带终止的距离-2分散算法的伪代码见算法 6。图 2 展示了我们的算法阶段 1 的运行过程。

3.2 算法分析

定义 3.1. 树边：如果阶段 *1* 中的未定居机器人组通过 (u, v) 到达 v，并且节点 u 上的已定居机器人（如果存在）存储了节点 v 的父指针，或者未定居机器人组中标识符最小的机器人在 v定居，那么边 (u, v) 称为树边。

注释 3.1。 阶段 *1* 中的已定居机器人 r_i 只有在满足 $r_i.special = 1$, $r_j.dist = 1$，且对于 u 处的任何未定居机器人 r_j，没有标识符低于 $r_i.id$ 的已定居机器人在 2ϕ 轮次期间（即等待期内）访问 r_j 时，才会在某一轮次 t 为其相邻节点 u 存储父指针。

定理 3.1。 算法 *6* 结束时，不存在两个机器人分别定居在相邻节点上。

算法3: 帮助终止(): _机器人 r_L 的算法

设置 $\phi = \delta_u$、 $r_L.state = explore$ 和 $r_L.portentered = 0$
沿着 $r_L.portentered$ 移动
$r_L.dist = r_L.dist + 1$
对于 $phase > 0$ **，执行**
    如果 $r_L.dist = 1$ *and* $r_L.state = explore$ **，则**
        等待 2ϕ 轮次
        如果 *any settled robot visits with* $r_j.act\ settled = 1$, $r_j.special = 0$, *and* $r_j.id <$ *id of act-settled robot already present with* r_L **那么**
            $r_L.state = backtrack$
            更新 $\phi = max\{\phi, \delta(u)\}$
            通过 $r_L.portentered$ 移动
            将 $r_L.dist$ 的值递减至 0
        如果 *any settled robot visits with* $r_j.act\ settled = 1$, $r_j.special = 0$, *but* $r_j.id >$ *id of act_settled robot already present with* r_L *or no* $r_j.act_settled = 1$ *robot visits* **然后**
            $r_L.state = explore$
            更新 $\phi = max\{\phi, \delta(u)\}$
            $r_L.portentered = (r_L.portentered + 1)$模$\delta$
            如果 $r_L.portentered = r_j.virtualparent$ **然后**
                $r_L.state = backtrack$
                通过 $r_L.portentered$移动
            else
                通过 $r_L.portentered$移动
                将 $r_L.dist$ 递增至 2
    如果 $r_L.dist = 2$ *and* $r_L.state = explore$ **，则**
        等待 2ϕ 轮次
        更新 $\phi = max\{\phi, \delta(u)\}$
        如果 *no settled robot with* $r_j.act\ settled = 1$ *visits* **，则**
            设置 $r_L.act\ settled = 1$
            终止 r_L
        如果 *any settled robot visits with* $r_j.act\ settled = 1$ **，则**
            设置 $r_L.state = backtrack$
            穿过 $r_L.portentered$ 移动
            将 $r_L.dist$ 的值减一
        如果 *there is a robot* r_j *present at the node with* $r_j.stage = 2$ **，则**
            设置 $r_L.dist = 0$
            设置 $r_L.portentered = (r_L.portentered + 1) mod \delta$
            移动穿过 $r_L.portentered$
    如果 $r_L.state = backtrack$ **则**
        等待 2ϕ 轮次
        设置 $r_L.portentered = (r_L.portentered + 1) mod \delta$
        如果 *the act settled robot* r_j *has* $r_j.parent = -1$ **则**
            如果 $r_L.portentered = 0$ **则**
                设置 $r_L.act\ settled = 1$
                终止 r_L 以及该节点上的所有其余已定居机器人
            else
                $r_L.state = explore$
                穿过 $r_L.portentered$移动
        else
            如果 $r_L.portentered = r_j.parent$ *or* $r_j.virtualparent$ **则**
                设为 $r_L.state = backtrack$
                移动穿过 $r_L.portentered$
            else
                $r_L.state = explore$
                移动穿过 $r_L.portentered$
                将 $r_L.dist$ 增加 1

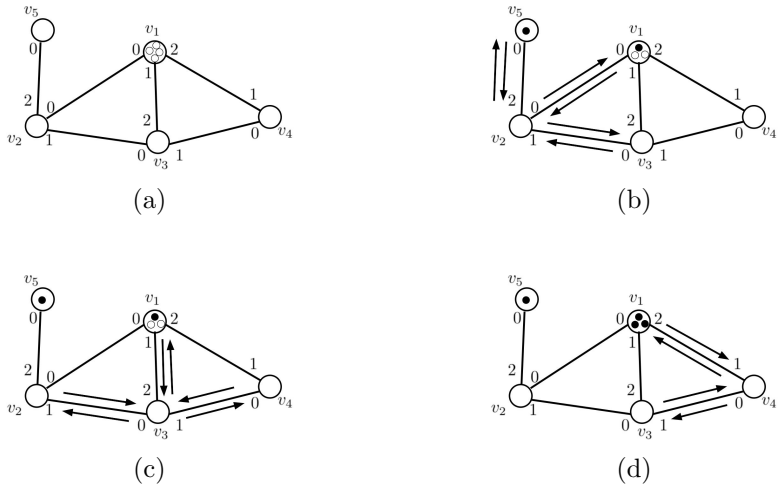


图 2: （a）四个机器人位于 v_1时的初始配置。（b）一个机器人（假设为 r）在 v_1 处定居，并更新 $r.special = 1$。其余机器人通过端口 0 离开，已定居机器人随群组移动到相邻节点 v_2。机器人 r 更新值 $r.count = 1$。它们等待 2ϕ 轮次，由于没有已定居机器人访问该群组，它们继续移动到节点 v_3 并更新 $r_i.dist = 2$；与此同时， r 在将 v_2 在 $r.virtualparent$ 中的父指针存为 0 后，返回 v_1，并更新 $r.special = 0$。未定居机器人组在 v_3 等待 2ϕ 轮次，由于已定居机器人 r 的访问，它们回溯到前一个节点 v_2，与此同时， r 更新值 $r.count = 2$。从 v_2出发，它探索 v_5；由于在等待 2ϕ 轮次期间没有已定居机器人访问v_5，一个机器人（假设为 r'）在那里定居。该群组回溯到 v_2。已定居机器人 r' 更新 $r'.count = 1$。已定居机器人 r 也将 $r.count$ 的值加一。未定居机器人组继续回溯到 v_1。（c）此时，未定居机器人组以及 r通过端口 1 移动并探索 v_3。它从 v_3 探索 v_2，与此同时， r在将 v_3 在 $r.virtualparent$中的父指针存为 2 后，返回 v_1。每当已定居机器人 r 和 r' 在任一节点遇到未定居机器人组时，它们都会将 $count$ 的值加一。在等待 2ϕ 轮次期间， r 访问了 v_2，因此该群组回溯到节点 v_3。随后，该群组探索 v_4，由于类似的原因，该群组回溯到 v_3。等待 2ϕ 轮次后，它从 r 获知父指针，并回溯到 v_1。（d）此时，该群组通过端口 2 移动以探索 v_4，随后继续移动以探索 v_3。由于已定居机器人访问了 v_1，该群组回溯到 v_4，然后继续回溯到 v_1。由于 v_1 是根节点且已探索完 v_1 的所有端口，未定居机器人在此处的 v_1定居。在阶段 1 结束时， $r.count = 15$ 的值，而 $r'.count = 2$。

算法4： 机器人 r_L 使用 $r_L.terminate = 1$的算法
到达根节点

```
如果 r_L.terminate = 1 and settled at a node other than the root node 则
    设置 r_L.stage = 2, r_L.state = backtrack
    通过 r_L.parent移动
    当 reached node is not the root node 时执行
        等待 2φ 轮次以获取父节点端口信息
        沿该节点的父指针移动
    调用帮助终止
否则如果 r_L.terminate = 1 and r_L is settled at the root node 则
    调用帮助终止
```

算法5： 每个机器人使用 r_j 和 $r_j.settled = 1$ 的算法
$r_j.act\text{-}settled = 1$ 在终止阶段

```
如果 r_j.act settled = 1 and r_j.special = 0 则
    通过其所有端口进行往返移动
    如果 r_j meets r_L with r_L.terminate = 1 则
        等待 r_L，直到它离开该节点
        设置 r_j.count' = r_j.count' + 1
        移动到其原始位置，然后恢复往返移动
        如果 r_j.count == r_j.count'，则
            终止 r_j
否则如果 r_j.act settled = 1 and r_j.special = 1，则
    与 r_L 一起移动，经过 r_L.portentered 的更新后的值，并等待直到 r_L
    叶子
    将 r_j.count' 的值递增
    如果no aet settled robot visits with id lower than r_j.id，则
        设置 r_j.virtualparent = r_L.portentered
        通过 r_j.portentered 移动到原始位置
    else
        通过 r_j.portentered 移动到原始位置
    更新 r_j.special = 0
```

证明。 算法1保证阶段 1 中的已定居机器人通过在其一跳邻居处往返移动来显示其存在。因此，当未定居机器人组访问某个节点并等待 2ϕ 轮次时，如果相邻的一跳邻居处有已定居机器人，它们就会遇到该机器人。根据阶段 1 中针对未定居机器人的算法 2，如果某个已定居机器人在某个节点遇到未定居机器人，则不会有未定居机器人定居。此外，根据算法5，阶段 2 中的已定居机器人会在阶段 1 中定居的节点定居。这保证了不会有两个相邻节点被机器人占据。

□

引理3.1. 多个机器人只能在根节点定居。

证明。 当机器人完成对图的遍历，且未找到任何满足 D-2-D 问题条件的可驻留节点时，最终会到达图的根节点，继续执行算法 2 并遍历该根节点。机器人可以轻易识别出根节点，因为该节点的父指针

算法 6:D-2-D 与终止 _ _

```
如果 r_i.settle = 0 则
    调用算法 2
否则如果 r_i.settle = 1 and r_i.aet settled = 0 则
    调用算法1
否则如果 r_i.settle = 1 and r_i.terminate = 1 则
    调用算法4
否则如果 r_i.settle = 1 and r_i.aet settled = 1 则
    调用算法5
```

指向 -1。从根节点依照算法行进并探索所有出边后，只有在找不到可驻留节点时，机器人才会回溯到根节点。在这种情况下，它们会驻留在根节点。因此，算法 2 只会使多个机器人驻留在根节点。

□

引理3.2. 如果在阶段 *1* 有多个机器人在根节点定居，那么可以保证每个节点至少会被未定居机器人组访问一次。

证明。 假设存在一个节点 u，它从未被访问过，但它至少有一个一跳邻居（例如 v）被访问过。这意味着，每当该组到达 v 时，它要么从 v 回溯，要么探索除连接 v 与 u的端口之外的所有端口。后一种情况不可能发生，因为算法 2 会递增 $portentered$ 的值，除非其值等于父指针的值。

现在不失一般性，考虑这样的情况： v 由一个群组从某个节点访问，而该节点包含的已定居机器人编号是 v 的一跳邻居中所有已定居机器人编号里最低的。算法 2 确保未定居机器人组只有在探索完 v 的所有端口后，才会从 v 回溯。因此，必须探索 u，这与存在这样一个节点 u相矛盾。

□

定理 3.2。 算法 *6* 结束时，如果有多个机器人已在根节点定居，则不存在这样的空节点：其邻居中没有已定居机器人。

证明。 假设图中存在一个空节点 u，且在阶段 1 中，其任何一跳邻居上都没有已定居机器人。引理 1 证明，节点 u 至少会被访问一次。根据我们针对阶段 1 中未定居机器人的算法（即算法 2），当群组访问 u时，群组中的每个机器人 r_j 都必须设置 $r_j.dist = 2$。在等待期内， u 的邻居中没有已定居机器人前往访问 u。因此，最小 ID 机器人必定已在那里定居。这与图中存在这样的节点相矛盾。

□

观察 3.1。 如果多个机器人在根节点定居，那么根据定理 *3.1* 和定理 *3.2*，存在已定居机器人的节点构成一个极大独立集。

定理 3.3. 机器人运行带终止功能的 *D-2-D* 需要 $O(\log \Delta)$额外内存。

证明。变量 $r_i.state$、$r_i.stage$、$r_i.settled$、$r_i.act\ settled$ 和 $r_i.special$ 需要 1 比特内存，而 $r_i.dist$需要 2 比特内存。变量$r_i.parent$、$r_i.portentered$ 和 $r_i.virtualparent$ 需要 $O(\log \Delta)$ 比特内存。位于节点 v、具有 $\delta(v) \leq \Delta$ 的已定居机器人，最多可在 $(\Delta + 1)$ 个节点（包括节点 v）与未定居机器人组相遇，这些节点关联的边最多有 $O(\Delta^2)$ 条。由于未定居机器人组对任何一条边的访问次数至多为 4，变量 $r_i.count$ 可取的最大值为 $O(\Delta^2)$。类似地，在阶段 2 中，位于节点 v、具有 $\delta(v) \leq \Delta$ 的 $a\bar{c}t\ settled$ 机器人可在 $(\Delta + 1)$ 个节点与 r_L相遇，因此，$r_i.count'$ 可取的最大值为 $O(\Delta^2)$。因此，存储与这些变量相关信息所需的内存量为 $O(log\Delta)$。由此，每个机器人只需 $O(\log \Delta)$ 比特额外内存即可运行该算法。

□

引理 3.3。当阶段 *1* 中的未定居机器人组处于 $explore$状态且 $r_i.dist = 1$ 时，该组中恰有一个已定居机器人同时存在，并且该机器人具有 $r_i.special = 1$。

证明。从表1中所述的已定居机器人 $r_i.special$ 变量的描述可以很容易地看出这一点。除根节点外，没有任何节点包含多个已定居机器人。此外，只有在没有机器人需要定居时，根节点才会包含多个机器人；也就是说，阶段 1 中不再有机器人处于探索状态。因此，该结论成立。

□

引理 3.4。阶段 *1* 中的每条树边都由未定居机器人组恰好遍历两次。

证明。不失一般性，根据定义 3.1，设 u 有一个已定居机器人。树边（$u,\ v$）在 v处有一个已定居机器人，或者在 u 处有一个已定居机器人，该机器人在探索边（$u,\ v$）期间存储了节点 v 的父指针。这确保了 v 是首次被访问，因为其父指针已被存储。因此，边（$u,\ v$）被遍历两次：一次处于 $explore$ 状态，另一次处于 $backtrack$ 状态。如算法1所述，仅当没有机器人以标识符 $< r_u.id$访问 v 时，位于节点 u 的已定居机器人 r_u 才会保存节点 v 的父指针。因此，机器人不会从 v 回溯，目的是从另一个有较低标识符的机器人定居于其上的节点探索节点 v。这证明边（$u,\ v$）恰好被遍历两次。

□

引理 3.5. 每条非树边最多被未定居机器人组遍历四次。

证明。设 (u, v) 是一条非树边。根据定义 3.1，机器人从节点 v 回退，此时尚未存储 v 的父指针。到这一轮为止，边 (u, v) 已被遍历两次。机器人从其邻域中标识符最小的已定居机器人处到达 v，以便稍后探索 v。此时，边 (v, u) 再次被遍历。因此，每条非树边最多被遍历四次。

□

引理 3.6。由树边诱导的图是连通且无环的。

证明。考虑图 G 上的一个有根配置，其根节点为 u，使得 u 的度至少为 2，且 $k \geq 1$ 也满足此条件。首先，我们证明树边构成一个连通分量。很容易看出，最初的两条树边构成一个连通分量。设 e_1、e_2，……，e_h 是最初的 h 条树边，它们构成一个连通分量。假设仍有一组未定居机器人正在进行遍历。设 $e_h = uw$ 对手某些节点 u、w 成立，其中这条树边是在未定居机器人组从 u 访问 w 时形成的。若对于 w 的某个邻居 w'，ww' 成为一条树边，那么我们就完成了证明，即第（$h+1$）条树边也仍属于同一个连通分量。否则，若 w 的其他关联边都不成为树边，该组机器人便沿一条树边从 w 回溯，并到达一个新节点，例如 v。同样，要么 v 的一条相邻边成为新的树边（此时我们就完成了证明），要么该组机器人沿另一条树边回溯。如此一来，该组机器人会继续沿着由树边构成的路径前进，直到找到一条新的树边，或者完成探索，且该组中的所有机器人都在根节点定居。无论是哪种情况，树边都构成一个连通分量。

现在我们证明该导出子图是无环的。反设树边构成一个环。令 $u_1, u_2, ..., u_p, u_1$ 为由树边构成的环。不失一般性，假设 u_iu_{i+1} 是形成该环的最后一条树边，且未定居机器人组从 u_i 移动到 u_{i+1}。现在分两种情况：u_{i+1} 处有已定居机器人，或者 u_{i+1} 处没有已定居机器人。若 u_{i+1} 处有已定居机器人，那么未定居机器人组应当从 u_{i+1} 回溯到 u_i，因此 u_iu_{i+1} 不可能是树边。这与我们的假设矛盾。因此，假设 u_{i+1} 处没有已定居机器人。定义 3.1 意味着 u_i 和 u_{i+2} 处都将有已定居机器人。现在，u_{i+1} 与这两台已定居机器人相距一跳，且探索正从 u_i 进行到 u_{i+1}。位于 u_i 和 u_{i+2} 的两台已定居机器人中，必有一台的标识符较小。若 u_{i+2} 处的已定居机器人标识符较小，那么机器人将从 u_{i+1} 回溯到 u_i，因此 u_iu_{i+1} 不会是树边。然而，若 u_i 处的已定居机器人标识符较小，那么在探索节点 u_{i+2} 并从 u_{i+2} 经过到 u_{i+1} 时，未定居机器人组必定因 u_i 处存在标识符更小的已定居机器人而回溯，从而形成 $u_{i+2}u_{i+1}$ 作为非树边。因此，我们看到 u_iu_{i+1} 和 $u_{i+2}u_{i+1}$ 不可能同时是树边。由此，所有树边构成一个环这一假设不成立，并且由树边导出的图是连通且无环的。

□

引理 3.7。阶段 *2* 结束时，每个机器人都会终止。

证明。由于机器人 r_L 在阶段 1 中复制了未定居机器人组，且所有带有 $r_L.act\ settled = 1$ 的机器人在阶段 1 中复制了已定居机器人，因此，每个已定居机器人在阶段 1 中与未定居机器人组相遇的次数，与每个 $act\ settled$ 机器人与 r_L 相遇的次数相同。如第 3.1 节所述，阶段 2 是对阶段 1 的重放。因此，阶段 1 的正确性意味着阶段 2 的正确性。于是，对于每个已定居机器人 r_i，除 r_L 之外，$r_i.count = r_i.count$，并终止。最后，r_L 定居在阶段 1 结束时所在的节点，并终止。

□

定理 3.4。 算法 *6* 在任意有根图上用 $2\Delta(8m-3n+3)$ 轮次实现 *D-2-D*，并保证终止。

证明。 由引理 3.4 和引理 3.5 可知，除树边外，每条边最多被遍历 4 次。此外，由引理 3.6 可知，树边最多有 $(n-1)$ 条。因此，边遍历总次数不超过 $4(m-(n-1))+2(n-1) = 4m-2n+2$。每次边遍历后，机器人等待 2ϕ 轮次，并且 $\phi \leq \Delta$。因此，所有机器人定居最多需要 $2\Delta(4m-2n+2)$ 轮次。由此，阶段 1 在 $2\Delta(4m-2n+2)$ 轮内完成。最后一个机器人定居后，在最坏情况下，它可能需要最多 $2\Delta(n-1)$ 轮才能到达根节点。现在，阶段 2 的剩余部分是我们算法阶段 1 的副本。因此，实现带终止的 D-2-D 需要 $2\Delta(8m-3n+3)$ 轮。

□

4 下界

本节讨论 D-2-D 问题在机器人额外内存不超过 $O(\log \Delta)$ 时所需轮次数的下界。我们先定义节点对机器人的视图。

定义 4.1。 视图： 节点 v 对机器人的视图，是指其任一一跳邻居处是否有已定居机器人的信息，包括 v。

接下来，我们通过构造一类图来证明该定理。其思路是，这类图中的每个图都是度为 $n-1$ 且有 $2n$ 个节点的正则图。我们从两个机器人开始，其中一个先定居，另一个则寻找定居节点。这些图具有这样的性质：除非未定居机器人到达两个特定节点之一，否则它无法将该图与团区分开来。因此，在到达这两个节点之一之前，如果它决定定居，就会导致错误的解。我们将证明，在内存有限的情况下，找到这两个节点之一至少需要 $\Omega(m\Delta)$ 轮次。

定理 4.1。 对于任意图上的 *D-2-D* 问题，当机器人拥有的额外内存不超过 $O(\log \Delta)$ 时，轮次数的下界为 $\Omega(m\Delta)$ 。

证明。 我们将通过一类图来证明这一点，并说明至少存在一张图，使得机器人至少需要 $\frac{\Delta m}{12}$ 轮次才能完成 D-2-D。考虑两个各有 n 个顶点、但各自缺少一条边的团。令 v_1、v_2、...、v_n 为第一个团 Q_1 的顶点，令 u_1、u_2、...、u_n 为第二个团 Q_2 的顶点。令 v_1v_2 为第一个团中缺失的边，u_1u_2 为第二个团中缺失的边。我们将 v_1 与 u_1 相连，并将 v_2 与 u_2 相连。现在，图 G 有 $2n$ 个节点，且满足 $\Delta = n-1$。考虑该图所有可能的不同端口编号，可得到图类 $\mathscr{G}$，其基数等于 $[(n-1)!]^{2n}$。设两个机器人 r_1 和 r_2 初始时位于 v_j，其中 $j \neq 1$，2。假设存在一个算法 $\mathscr{A}$，能在少于 $\frac{m\Delta}{12}$ 的时间内解决 D-2-D。令 r_1 先停下，并位于节点 w。我们可以断言 ——

至少存在 $\frac{|\mathcal{G}|}{2}$ 张图，其中 $w \notin \{v_1, v_2, u_1, u_2\}$。不失一般性，设 $w = v_i$ 是 Q_1 的某个顶点。我们用 N 表示 $\frac{|\mathcal{G}|}{2}$。

由于机器人拥有 $O(\log \Delta)$ 的内存，因此它们一次只能记住常数个端口号。通过让其知道在 v_i 的两跳距离内存在一个可供停留的节点，我们为 r_2 赋予了更多能力。机器人 r_2 旨在探索全部 $\Delta(\Delta-1)$ 个两跳邻居。存在足够多的图（特别是 $\frac{N}{4}$），在这些图中，机器人 r_2 需要至少探索 $\frac{\Delta}{(\Delta-1)\,2}$ 个顶点，才能探索u_1 或 u_2。除非它到达 u_1 或 u_2 并获得该视图，否则 r_2 无法将我们图类中的任何图与一个包含 n 个节点的团区分开来。

设节点被探索的顺序如下：$\{v_{i_1}, v_{i_2}, \ldots\ldots, v_{i_{\Delta(\Delta-1)}}\}$。当 r_2 到达 v_{i_1}时，它需要知道图的视图。如果 v_{i_1} 是从 v_i 直接到达的，则获取该视图只需一轮，因为 r_2 明白它与 v_i相距一跳。否则，如果v_{i_1} 不是从 v_i直接到达的，那么很容易看出，在至少一半的图中，r_2 至少需要 $\frac{\Delta}{2}$ 轮次才能获得该视图。因此，存在足够多的实例（特别是至少 $\frac{N}{4.2}$个），其中 r_2需要 $\frac{\Delta}{2}$ 轮次才能找到该视图。同样，在到达 v_{i_2} 之后，至少存在 $\frac{N}{4.2^2}$ 个图，在这些图中，需要 $\frac{\Delta}{2}$ 轮次才能找到该节点的视图。类似地，在 $v_{i_{\Delta(\frac{\Delta-1}{2})}}$处，至少存在 $\frac{N}{4.2\,\Delta^{(\Delta-1)\,2}}$个图。现在，$\frac{N}{4.2^{\frac{\Delta(\Delta-1)}{2}}}$ 是 n 的函数，并且对于所有 $n \geq 3$，该值都大于 1。

因此，至少存在一个图，使得机器人 r_2 需要花费至少$\Delta^{\,(\Delta-1)\,2}\cdot\frac{\Delta}{2}$轮次才能定居。对于 $n \geq 3, \Delta \geq \frac{M}{3}$，其中 $M = 2n$。因此，$\Delta^{\,(\Delta-1)\,2}\cdot\frac{\Delta}{2} \geq \frac{M}{3}\cdot\frac{(\Delta-1)}{2}\cdot\frac{\Delta}{2} = m.\frac{\Delta-1}{6} \geq \frac{m\Delta}{12}$。这证明了在类 $\mathcal{G}$ 中至少存在一个这样的实例：机器人 r_2 需要经过 $\frac{m\Delta}{12}$ 多轮才能完成 D-2-D；否则，r_1 和 r_2 都会定居在 Q_1 或 Q_2 上，从而导致 D-2-D 错误。证明至此完成。 □

5 结论与未来工作

我们提出了分散问题的一种变体，并提供了一种算法，可在有根初始配置下解决该问题；每个机器人需要 $O(\log \Delta)$ 的额外内存，且算法在 $2\Delta(8m-3n+3)$ 个同步轮次内完成。我们还给出了该问题在轮次数量上的 $\Omega(m\Delta)$ 下界。在某些情况下，我们保证机器人能够构成一个极大独立集，这一结果本身也可能具有研究价值。如何解决机器人处于任意初始配置时的问题，将是一个值得探讨的方向。

参考文献

[1] Ankush Agarwalla、John Augustine、William K. Moses Jr.、Sankar Madhav K. 和 Arvind Krishna Sridhar。动态环中移动机器人的确定性分散。载于*ICDCN*，第 19:1–19:4 页。ACM，2018年。

[2] John Augustine 和 William K. Moses Jr. 移动机器人的分散：内存-时间权衡研究。载于*ICDCN*，第 1:1–1:10 页，2018年。[3] Lali Barri`ere、P Flocchini、E M Barrameda 和 N Santoro。网格中自主移动机器人的均匀散布。载于 *IPDPS*，第 1–8 页，2009年。[4] Archak Das、Kaustav Bose 和 Buddhadeb Sau。匿名移动机器人实现内存最优分散。载于 *CALDAM*，第 426–439 页，2021年。[5] Shantanu Das。移动智能体的图探索。载于移动实体的分布式计算：移动与计算当前研究，第 403–422 页。Springer，2019年。[6] D. Dereniowski、Y. Disser、Adrian Kosowski、D. Pajak 和 P. Uznanski。快速协作式图探索。*Inf. Comput.*，243:37–49，2015年。[7] Yotam Elor 和 Alfred M. Bruckstein。环上的均匀多智能体部署。*Theor. Comput. Sci.*，412(8-10):783–795，2011年。[8] Barun Gorain、Partha Sarathi Mandal、Kaushik Mondal 和 Supantha Pandit。静默机器人的协作式分散。*CoRR*，abs/2202.05710，2022年。[9] A D. Kshemkalyani、A R Molla 和 G Sharma。任意图上移动机器人的快速分散。载于 *ALGOSENSORS*，第 23–40 页，2019年。[10] A. D. Kshemkalyani、A. R. Molla 和 G. Sharma。网格上移动机器人的分散。载于*WALCOM*，第 12049 卷，第 183–197 页。Springer，2020年。[11] Ajay D. Kshemkalyani 和 Faizan Ali。图上移动机器人的高效分散。载于*ICDCN*，第 218–227 页，2019 年。[12] Ajay D. Kshemkalyani 、 Anisur Rahaman Molla 和 Gokarna Sharma。利用全局通信实现移动机器人分散。*J. Parallel DistributedComput.*，161:100–117，2022年。[13] Ajay D. Kshemkalyani 和 Gokarna Sharma。任意匿名图上的近最优分散。载于 *OPODIS*，第 8:1–8:19 页，2021年。[14] Anisur Rahaman Molla 和 William K. Moses Jr. 移动机器人的分散：随机性的力量。载于*TAMC*，第 11436 卷，第 481–500 页，2019年。[15] Anisur Rahaman Molla、Kaushik Mondal 和 William K. Moses Jr. 弱拜占庭机器人存在时匿名环上的高效分散。载于*ALGOSENSORS*，第 12503 卷，第 154–169 页。Springer，2020年。[16] Anisur Rahaman Molla、Kaushik Mondal 和 William K. Moses Jr. 图上的拜占庭分散。载于*IPDPS*，第 942–951 页。IEEE，2021年。[17] Anisur Rahaman Molla、Kaushik Mondal 和 William K. Moses Jr. 弱拜占庭机器人存在时匿名环上的最优分散。*Theor. Comput. Sci.*，887:111–121，2021年。[18] S. Vamshi Samala、S. Pra manick、D. Pattanayak 和 P. S. Mandal。短视发光机器人填充极大独立集顶点。*CoRR*，abs/2107.04885，2021年。

[19] Masahiro Shibata、ToshiyaMega、FukuhitoOoshita、HirotsuguKakugawa 和 Toshimitsu Masuzawa。异步环中移动代理的均匀部署。*J. Parallel Distributed Comput.*，119：92–106，2018年。[20] Takahiro Shintaku、Yuichi Sudo、Hirotsugu Kakugawa 和 Toshimitsu Masuzawa。无需全局知识的移动代理高效分散。载于*SSS*，第 12514 卷，第 280–294 页，2020年。