Fast Deterministic Rendezvous in Labeled Lines

Avery Miller
University of Manitoba, Winnipeg, Canada

Andrzej Pelc
Université du Québec en Outaouais, Canada

Abstract

Two mobile agents, starting from different nodes of a network modeled as a graph, and woken up at possibly different times, have to meet at the same node. This problem is known as *rendezvous*. Agents move in synchronous rounds. In each round, an agent can either stay idle or move to an adjacent node. We consider deterministic rendezvous in the infinite line, i.e., the infinite graph with all nodes of degree 2. Each node has a distinct label which is a positive integer. An agent currently located at a node can see its label and both ports 0 and 1 at the node. The time of rendezvous is the number of rounds until meeting, counted from the starting round of the earlier agent.

We consider three scenarios. In the first scenario, each agent knows its position in the line, i.e., each of them knows its initial distance from the smallest-labeled node, on which side of this node it is located, and the direction towards it. For this scenario, we design a rendezvous algorithm working in time $O(D)$, where D is the initial distance between the agents. This complexity is clearly optimal. In the second scenario, each agent knows *a priori* only the label of its starting node and the initial distance D between them. In this scenario, we design a rendezvous algorithm working in time $O(D \log^* \ell)$, where ℓ is the larger label of the starting nodes. We also prove a matching lower bound $\Omega(D \log^* \ell)$. Finally, in the most general scenario, where each agent knows *a priori* only the label of its starting node, we design a rendezvous algorithm working in time $O(D^2 (\log^* \ell)^3)$, which is thus at most cubic in the lower bound. All our results remain valid (with small changes) for arbitrary finite lines and for cycles. Our algorithms are drastically better than approaches that use graph exploration, which have running times that depend on the size or diameter of the graph.

Our main methodological tool, and the main novelty of the paper, is a two way reduction: from fast colouring of the infinite labeled line using a constant number of colours in the $\mathcal{LOCAL}$ model to fast rendezvous in this line, and vice-versa. In one direction we use fast node colouring to quickly break symmetry between the identical agents. In the other direction, a lower bound on colouring time implies a lower bound on the time of breaking symmetry between the agents, and hence a lower bound on their meeting time.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases rendezvous, deterministic algorithm, mobile agent, labeled line, graph colouring

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.29

Funding *Avery Miller*: Supported by NSERC Discovery Grant 2017-05936.
Andrzej Pelc: Partially supported by NSERC Discovery Grant 2018-03899 and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

1 Introduction

Background. Two mobile agents, starting from different nodes of a network modeled as a graph, and woken up at possibly different times, have to meet at the same node. This problem is known as *rendezvous*. The autonomous mobile entities (agents) may be natural, such as people who want to meet in a city whose streets form a network. They may also represent human-made objects, such as software agents in computer networks or mobile robots navigating in a network of corridors in a mine or a building. Agents might want to meet to share previously collected data or to coordinate future network maintenance tasks.

Model and problem description. We consider deterministic rendezvous in the infinite line, i.e., the infinite graph with all nodes of degree 2. Each node has a distinct label which is a positive integer. An agent currently located at a node can see its label and both ports 0 and 1 at the node. Technically, agents are anonymous, but each agent could adopt the label of its starting node as its label. Agents have synchronized clocks ticking at the same rate, measuring *rounds*. Agents are woken up by an adversary in possibly different rounds. The clock of each agent starts in its wakeup round. In each round, an agent can either stay idle or move to an adjacent node. After moving, an agent knows on which port number it arrived at its current node. When agents cross, i.e., traverse an edge simultaneously in different directions, they do not notice this fact. No limitation is imposed on the memory of the agents. Computationally, they are modeled as Turing Machines. The time of rendezvous is the number of rounds until meeting, counted from the starting round of the earlier agent.

In most of the literature concerning rendezvous, nodes are assumed to be anonymous. The usual justification of this weak assumption is that in labeled graphs (i.e., graphs whose nodes are assigned distinct positive integers) each agent can explore the entire graph and then meet the other one at the node with the smallest label. Thus, the rendezvous problem in labeled graphs can be reduced to exploration. While this simple strategy is correct for finite graphs, it requires time at least equal to the size of the graph, and hence very inefficient for large finite graphs, even if the agents start at a very small initial distance. In infinite graphs, this strategy is incorrect: indeed, if label 1 is not used in some labeling, an agent may never learn what is the smallest used label. For large environments that would be impractical to exhaustively search, it is desirable and natural to relate rendezvous time to the initial distance between the agents, rather than to the size of the graph or the value of the largest node label. This motivates our choice to study the infinite labeled line: we must avoid algorithms that depend on exhaustive search, and there cannot be an efficient algorithm that is dependent on a global network property such as size, largest node label, etc. In particular, the initial distance between the agents and the labels of encountered nodes should be the only parameters available to measure the efficiency of algorithms. Results for the infinite line can then be applied to understand what's possible in large path-like environments.

We consider three scenarios, depending on the amount of knowledge available *a priori* to the agents. In the first scenario, each agent knows its position in the line, i.e., each of them knows its distance from the smallest-labeled node, on which side of this node it is located, and the direction towards it. This scenario is equivalent to assuming that the labeling is very particular, such that both above pieces of information can be deduced by the agent by inspecting the label of its starting node and port numbers at it. One example of such a labeling is $...8, 6, 4, 2, 1, 3, 5, 7, ...$ (recall that labels must be positive integers), with the following port numbering. For nodes with odd labels, port 1 always points to the neighbour with larger label and port 0 points to the neighbour with smaller label, while for nodes with even label, port 1 always points to the neighbour with smaller label and port 0 points to the neighbour with larger label. We will call the line with the above node labeling and port numbering the *canonical line*, and use, for any node, the term "right of the node" for the direction corresponding to port 1 and the term "left of the node" for the direction corresponding to port 0. In the second scenario that we consider, the labeling and port numbering are arbitrary and each agent knows *a priori* only the label of its starting node and the initial distance D between the agents. Finally, in the third scenario, the labeling and port numbering are arbitrary and each agent knows *a priori* only the label of its starting node.

Our results. We start with the scenario of the canonical line. This scenario was previously considered in [13], where the authors give a rendezvous algorithm with optimal complexity $O(D)$, where D is the (arbitrary) initial distance between the agents. However, they use the strong assumption that both agents are woken up in the same round. In our first result we get rid of this assumption: we design a rendezvous algorithm with complexity $O(D)$, regardless of the delay between the wakeup of the agents. This complexity is clearly optimal.

The remaining results are expressed in terms of the *base-2 iterated logarithm* function, denoted by $\log^*$. The value of $\log^*(n)$ is 0 if $n \leq 1$, and otherwise its value is $1 + \log^*(\log_2 n)$. This function grows extremely slowly with respect to n, e.g., $\log^*(2^{65536}) = 5$.

For the second scenario (arbitrary labeling with known initial distance D) we design a rendezvous algorithm working in time $O(D \log^* \ell)$, where ℓ is the larger label of the two starting nodes. We also prove a matching lower bound $\Omega(D \log^* \ell)$, by constructing an infinite labeled line in which this time is needed, even if D is known and if agents start simultaneously. As a corollary, we get the following impossibility result: for any deterministic algorithm and for any finite time T, there exists an infinite labeled line such that two agents starting from some pair of adjacent nodes cannot meet in time T.

Finally, for the most general scenario, where each agent knows *a priori* only the label of its starting node, we design a rendezvous algorithm working in time $O(D^2(\log^* \ell)^3)$, for arbitrary unknown D. This complexity is thus at most cubic in the above lower bound that holds even if D is known.

It should be stressed that the complexities of our algorithms in the second and third scenarios depend on D and on the larger label of the two starting nodes. No algorithm whose complexity depends on D and on the maximum label in even a small vicinity of the starting nodes would be satisfactory, since neighbouring nodes can have labels differing in a dramatic way, e.g., consider two agents that start at adjacent nodes, and these nodes are labeled with small integers like 2 and 3, but other node labels in their neighbourhoods are extremely large. Our approach demonstrates that this is an instance that can be solved very quickly, but an approach whose running time depends on the labels of other nodes in the neighbourhood can result in arbitrarily large running times (which are very far away from the lower bound).

An alternative way of looking at the above three scenarios is the following. In the first scenario, the agent is given *a priori* the entire (ordered) labeling of the line. Of course, since this is an infinite object, the labeling cannot be given to the agent as a whole, but the agent can *a priori* get the answer to any question about it, in our case answers to two simple questions: how far is the smallest-labeled node and in which direction. In the second and third scenarios, the agent gets information about the label of a given node only when visiting it. It is instructive to consider another, even weaker, scenario: the agent learns the label of its initial node but the other labels are never revealed to it. This weak scenario is equivalent to the scenario of labeled agents operating in an anonymous line: the labels of the agents are guaranteed to be different integers (they are the labels of the starting nodes) but all other nodes look the same. It follows from [16] that in this scenario the optimal time of rendezvous is $\Theta(D \log \ell)$, where ℓ is, as usual, the larger label of the starting nodes.[1] Hence, at least for known D, we get strict separations between optimal rendezvous complexities, according to three ways of getting knowledge about the labeling of the line: $\Theta(D)$, if all knowledge is given at once, $\Theta(D \log^* \ell)$ if knowledge about the labeling is given as the agents visit new nodes, and $\Theta(D \log \ell)$, if no knowledge about the labeling is ever given to the agents, apart

[1] In [16] cycles are considered instead of the infinite line, and the model and result are slightly different but obtaining, as a corollary, this complexity in our model is straightforward.

from the label of the starting node. Of course, in the ultimately weak scenario where even the label of the starting node is hidden from the agents, deterministic rendezvous would become impossible: there is no deterministic algorithm guaranteeing rendezvous of anonymous agents in an anonymous infinite line, even if they start simultaneously from adjacent nodes.

It should be mentioned that the $O(D \log \ell)$-time rendezvous algorithm from [16] is also valid in our scenario of arbitrary labeled lines, with labels of nodes visible to the agents at their visit and with unknown D: the labels of the visited nodes can be simply ignored by the algorithm from [16]. This complexity is incomparable with our complexity $O(D^2(\log^* \ell)^3)$. For bounded D our algorithm is much faster but for large D compared to ℓ it is less efficient.

As usual in models where agents cannot detect crossing on the same edge when moving in opposite directions, we guarantee rendezvous by creating time intervals in which one agent is idle at its starting node and the other searches a sufficiently large neighbourhood to include this node. Since the adversary can choose the starting rounds of the agents, it is difficult to organize these time intervals to satisfy the above requirement. Our main methodological tool, and the main novelty of the paper is a two way reduction: from fast colouring of the infinite labeled line using a constant number of colours in the $\mathcal{LOCAL}$ model to fast rendezvous in this line, and vice-versa. In one direction we use fast node colouring to quickly break symmetry between the identical agents. In the other direction, a lower bound on colouring time implies a lower bound on the time of breaking symmetry between the agents, and hence a lower bound on their meeting time.

As part of our approach to solve rendezvous using node colouring, we provide a result (based on the idea of Cole and Vishkin [12]) that might be of independent interest: for the $\mathcal{LOCAL}$ model, we give a deterministic distributed algorithm EarlyStopCV that properly 3-colours any infinite labeled line such that the execution of EarlyStopCV at any node x with initial label ID_x terminates in time $O(\log^*(\mathrm{ID}_x))$, and the algorithm does not require that the nodes have any initial knowledge about the network other than their own label.

All our results remain valid for finite lines and cycles: in the first scenario without any change, and in the two other scenarios with small changes. As previously mentioned, it is always possible to meet in a labeled line or cycle of size n in time $O(n)$ by exploring the entire graph and going to the smallest-labeled node. Thus, in the second scenario the upper and lower bounds on complexity change to $O(\min(n, D \log^* \ell))$ and $\Omega(\min(n, D \log^* \ell))$, respectively, and in the third, most general scenario, the complexity of (a slightly modified version of) our algorithm changes to $O(\min(n, D^2(\log^* \ell)^3))$.

Related work. Rendezvous has been studied both under randomized and deterministic scenarios. A survey of randomized rendezvous under various models can be found in [3], cf. also [1, 2, 4, 7]. Deterministic rendezvous in networks has been surveyed in [27, 28]. Many authors considered geometric scenarios (rendezvous in the real line, e.g., [7, 8], or in the plane, e.g., [5, 6, 11, 14]). Gathering more than two agents was studied, e.g., in [18, 24, 31].

In the deterministic setting, many authors studied the feasibility and time complexity of synchronous rendezvous in networks. For example, deterministic rendezvous of agents equipped with tokens used to mark nodes was considered, e.g., in [22]. In most of the papers concerning rendezvous in networks, nodes of the network are assumed to be unlabeled and agents are not allowed to mark the nodes in any way. In this case, the symmetry is usually broken by assigning the agents distinct labels and assuming that each agent knows its own label but not the label of the other agent. Deterministic rendezvous of labeled agents in rings was investigated, e.g., in [16, 20] and in arbitrary graphs in [16, 20, 30]. In [16], the authors present a rendezvous algorithm whose running time is polynomial in the size of the

graph, in the length of the shorter label and in the delay between the starting times of the agents. In [20, 30], rendezvous time is polynomial in the first two of these parameters and independent of the delay. In [9, 10] rendezvous was considered, respectively, in unlabeled infinite trees, and in arbitrary connected unlabeled infinite graphs, but the complexities depended, among others, on the logarithm of the size of the label space. Gathering many anonymous agents in unlabeled networks was investigated in [17]. In this weak scenario, not all initial configurations of agents are possible to gather, and the authors of [17] characterized all such configurations and provided universal gathering algorithms for them. On the other hand, time of rendezvous in labeled networks was studied, e.g., in [26], in the context of algorithms with advice. In [13], the authors studied rendezvous under a very strong assumption that each agent has a map of the network and knows its position in it.

Memory required by the agents to achieve deterministic rendezvous was studied in [19] for trees and in [15] for general graphs. Memory needed for randomized rendezvous in the ring was discussed, e.g., in [21].

Roadmap. In Section 2, we state two results concerning fast colouring of labeled lines. In Section 3, we present our optimal rendezvous algorithm for the canonical line. In Section 4, we present our optimal rendezvous algorithm for arbitrary labeled lines with known initial distance between the agents, we analyze the algorithm, and we sketch the proof of the matching lower bound. In Section 5, we give our rendezvous algorithm for arbitrary labeled lines with unknown initial distance between the agents. In Section 6, we conclude the paper and present some open problems. In the appendix, we describe a fast colouring algorithm for the infinite line, as announced in Section 2.

Due to lack of space, the analysis of the algorithms for the canonical line and for arbitrary labeled lines with unknown initial distance between the agents, as well as detailed proofs of all results concerning the lower bound from Section 4, are omitted and will appear in the journal version of the paper.

2 Tools

We will use two results concerning distributed node-colouring of lines and cycles in the $\mathcal{LOCAL}$ communication model [29]. In this model, communication proceeds in synchronous rounds and all nodes start simultaneously. In each round, each node can exchange arbitrary messages with all of its neighbours and perform arbitrary local computations. Recall that, in the problem of k-colouring of a graph, we start with a graph whose nodes are labeled with distinct labels, each node knows only its own label, and at the end each node has to adopt one of k colours in such a way that adjacent nodes have different colours.

The first result is based on the 3-colouring algorithm of Cole and Vishkin [12], but improves on their result in two ways. First, our algorithm does not require that the nodes know the size of the network or the largest label in the network. Second, the running time of our algorithm at any node x with initial label ID_x is $O(\log^*(\mathrm{ID}_x))$. The running-time guarantee is vital for later results in this paper, and it is not provided by the original $O(\log^*(n))$ algorithm of Cole and Vishkin since the correctness of their algorithm depends on the fact that all nodes execute the algorithm for the same number of rounds. Our algorithm from Proposition 1, called EarlyStopCV, is described in the appendix.

▶ **Proposition 1.** *There exists an integer constant $\kappa > 1$ and a deterministic distributed algorithm EarlyStopCV such that, for any infinite line G with nodes labeled with distinct integers greater than 1, EarlyStopCV 3-colours G and the execution at any node x with initial label ID_x terminates in time at most $\kappa \log^*(\mathrm{ID}_x)$.*

The second result is a lower bound due to Linial [23, 25]. Note that Linial's original result was formulated for cycles labeled with integers in the range $1, \ldots, n$, but the simplified proof in [23] can be adapted to hold in our formulation below.

▶ **Proposition 2.** *Fix any positive integer n and any set $\mathcal{I}$ of n integers. For any deterministic algorithm $\mathcal{A}$ that 3-colours any path on n nodes with distinct labels from $\mathcal{I}$, there is such a path and at least one node for which algorithm $\mathcal{A}$'s execution takes time at least $\frac{1}{2}\log^*(n) - 1$.*

Finally, we highlight an important connection between the agent-based computational model considered in this paper (where algorithm executions may start in different rounds) and the node-based $\mathcal{LOCAL}$ model (where all algorithm executions start in the same round). Consider a fixed network G in which the nodes are labeled with fixed distinct integers. From the communication constraints imposed by the $\mathcal{LOCAL}$ model, for any deterministic algorithm $\mathcal{A}$, each node v's behaviour in the first i rounds is completely determined by the labeled $(i-1)$-neighbourhood of v in G (i.e., the induced labeled subgraph of G consisting of all nodes within distance $i-1$ from v). By Proposition 1, we know that each node x executing EarlyStopCV in G determines its final colour within $\kappa \log^*(\mathrm{ID}_x)$ rounds, so its colour is completely determined by its labeled $(\kappa \log^*(\mathrm{ID}_x) - 1)$-neighbourhood in G. So, in the agent-based computational model, an agent operating in the same labeled network G with knowledge of EarlyStopCV and the value of κ from Proposition 1 could, starting in any round: visit all nodes within a distance $\kappa \log^*(\mathrm{ID}_x) - 1$ from x, then simulate in its local memory (in one round) the behaviour of x in the first $\kappa \log^*(\mathrm{ID}_x)$ rounds of EarlyStopCV in the $\mathcal{LOCAL}$ model, and thus determine the colour that would be chosen by node x as if all nodes in G had executed EarlyStopCV in the $\mathcal{LOCAL}$ model starting in round 1.

▶ **Proposition 3.** *Consider a fixed infinite line G such that the nodes are labeled with distinct integers greater than 1. Consider any two nodes labeled v_α and v_β in G. Suppose that all nodes in G execute algorithm EarlyStopCV in the $\mathcal{LOCAL}$ model starting in round 1, and let c_α and c_β be the colours that nodes v_α and v_β, respectively, output at the end of their executions. Next, consider two agents α and β that start their executions at nodes labeled v_α and v_β in G, respectively (and possibly in different rounds). Suppose that there is a round r_α in which agent α knows the $(\kappa \log^*(v_\alpha))$-neighbourhood of node v_α in G, and suppose that there is a round r_β in which agent β knows the $(\kappa \log^*(v_\beta))$-neighbourhood of node v_β in G (and note that we may have $r_\alpha \neq r_\beta$). Then, agent α can compute c_α in round r_α, and, agent β can compute c_β in round r_β.*

3 The canonical line

In this section, we describe an algorithm $\mathcal{A}_{\text{rv-canon}}$ that solves rendezvous on the canonical line in time $O(D)$ when two agents start at arbitrary positions and the delay between the rounds in which they start executing the algorithm is arbitrary. The agents do not know the initial distance D between them, and do not know the delay between the starting rounds.

Denote by α and β the two agents. Denote by v_α and v_β the starting nodes of α and β, respectively. Denote by $\mathcal{O}$ the node on the canonical line with the smallest label. For $a \in \{\alpha, \beta\}$, we will write $d(v_a, \mathcal{O})$ to denote the distance between v_a and $\mathcal{O}$.

Algorithm $\mathcal{A}_{\text{rv-canon}}$ proceeds in phases, numbered starting at 0. Each phase's description has two main components. The first component is a colouring of all nodes on the line. At a high level, in each phase $i \geq 0$, the line is partitioned into segments consisting of 2^i consecutive nodes each, and the set of segments is properly coloured, i.e., all nodes in the same segment get the same colour, and two neighbouring nodes in different segments get

different colours. The second component describes how an agent behaves when executing the phase. At a high level, the phase consists of equal-sized blocks of rounds, and in each block, an agent either stays idle at its starting node for all rounds in the block, or, it spends the block performing a search of nearby nodes in an attempt to find the other agent (and if not successful, returns back to its starting node). Whether an agent idles or searches in a particular block of the phase depends on the colour of its starting node. The overall idea is: there exists a phase in which the starting nodes of the agents will be coloured differently, and this will result in one agent idling while the other searches, which will result in rendezvous.

The above intuition overlooks two main difficulties. The first difficulty is that the agents do not know the initial distance between them, so they do not know how far they have to search when trying to find the other agent. To deal with this issue, the agents will use larger and larger "guesses" in each subsequent phase of the algorithm, and eventually the radius of their search will be large enough. The second difficulty is that the agents do not necessarily start the algorithm in the same round, so the agents' executions of the algorithm (i.e., the phases and blocks) can misalign in arbitrary ways. This makes it difficult to ensure that there is a large enough set of contiguous rounds during which one agent remains idle while the other agent searches. To deal with this issue, we carefully choose the sizes of blocks and phases, as well as the type of behaviour (idle vs. search) carried out in each block.

We now give the full details of an arbitrary phase $i \geq 0$ in the algorithm's execution.

Colouring. When an agent starts executing phase i, it first determines the colour of its starting node. From a global perspective, the idea is to assign colours to nodes on the infinite line in the following way:

1. Partition the nodes into segments consisting of 2^i nodes each, such that node $\mathcal{O}$ is the leftmost node of its segment. Denote the segment containing $\mathcal{O}$ by S_0, denote each subsequent segment to its right using increasing indices (i.e., $S_1, S_2, \ldots$) and denote each subsequent segment to its left using decreasing indices (i.e., $S_{-1}, S_{-2}, \ldots$).
2. For each segment with even index, colour all nodes in the segment with "red". For each segment with odd index, colour all nodes in the segment with "blue". As a result, neighbouring segments are always assigned different colours.

However, the agent executing phase i does not compute this colouring for the entire line, it need only determine the colour of its starting node. To do so, it uses its knowledge about the distance and direction from its starting node to node $\mathcal{O}$. In particular,

- if the agent's starting node s is $\mathcal{O}$ or to the right of $\mathcal{O}$, it computes the index of the segment in which s is contained, i.e., $index = \lfloor d(s, \mathcal{O})/2^i \rfloor$. If $index$ is even, then s has colour red, and otherwise s has colour blue.
- if the agent's starting node s is to the left of $\mathcal{O}$, it computes the index of the segment in which s is contained, i.e., $index = -\lceil (d(s, \mathcal{O}))/2^i \rceil$. If $index$ is even, then s has colour red, and otherwise s has colour blue.

Behaviour. Phase i consists of $44 \cdot 2^{i+1}$ rounds, partitioned into 11 equal-sized blocks of $4 \cdot 2^{i+1}$ rounds each. The number 2^{i+1} has a special significance: it is the search radius used by an agent during phase i whenever it is searching for the other agent. We use the notation $SR(i)$ to represent the value 2^{i+1} in the remainder of the algorithm's description and analysis. In each of the 11 blocks of the phase, the agent behaves in one of two ways: if a block is designated as a *waiting block*, the agent stays at its starting node v for all $4 \cdot SR(i)$ rounds of the block; otherwise, a block is designated as a *searching block*, in which the agent moves right $SR(i)$ times, then left $2 \cdot SR(i)$ times, then right $SR(i)$ times. In other words, during a

searching block, the agent explores all nodes within its immediate $SR(i)$-neighbourhood, and ends up back at its starting node. Whether a block is designated as "waiting" or "searching" depends on the agent's starting node colour in phase i. In particular, if its starting node is red, then blocks 1,8,9 are searching blocks and all others are waiting blocks; otherwise, if its starting node is blue, then blocks 1,10,11 are searching blocks and all others are waiting blocks. This concludes the description of $\mathcal{A}_{\text{rv-canon}}$.

4 Arbitrary lines with known initial distance between agents

4.1 The algorithm

In this section, we describe an algorithm called $\mathcal{A}_{\text{rv-D}}$ that solves rendezvous on lines with arbitrary node labelings when two agents start at arbitrary positions and when the delay between the rounds in which they start executing the algorithm is arbitrary. The algorithm works in time $O(D \log^* \ell)$, where D is the initial distance between the agents, and ℓ is the larger label of the two starting nodes. The agents know D, but they do not know the delay between the starting rounds. Also, we note that the agents have no global sense of direction, but each agent can locally choose port 0 from its starting node to represent "right" and port 1 from its starting node to represent "left". Further, using knowledge of the port number of the edge on which it arrived at a node, an agent is able to choose whether its next move will continue in the same direction or if it will switch directions. Without loss of generality, we may assume that all node labels are strictly greater than one, since the algorithm could be re-written to add one to each label value in its own memory before carrying out any computations involving the labels. This assumption ensures that, for any node label v, the value of $\log^*(v)$ is strictly greater than 0.

Algorithm $\mathcal{A}_{\text{rv-D}}$ proceeds in two stages. In the first stage, the agents assign colours to their starting nodes according to a proper colouring of the nodes that are integer multiples of D away. They each accomplish this by determining the node labels within a sufficiently large neighbourhood and then executing the 3-colouring algorithm EarlyStopCV from Proposition 1 on nodes that are multiples of D away from their starting node. The second stage consists of repeated periods, with each period consisting of equal-sized blocks of rounds. In each block, an agent either stays idle at its starting node, or, it spends the block performing a search of nearby nodes in an attempt to find the other agent (and if not successful, returns back to its starting node). Whether or not an agent idles or searches in a particular block of the period depends on the colour it picked for its starting node. The overall idea behind the algorithm's correctness is: the agents are guaranteed to pick different colours for their starting node in the first stage, and so, when both agents are executing the second stage, one agent will search while the other idles, which will result in rendezvous.

Stage 1: Colouring. Let v_x be the starting node of agent x. We identify the node with its label. Let $r = D \cdot \kappa \log^*(v_x)$, where $\kappa > 1$ is the constant defined in the running time of the algorithm EarlyStopCV from Proposition 1. Denote by $\mathcal{B}_r$ the r-neighbourhood of v_x. First, agent x determines $\mathcal{B}_r$ (including all node labels) by moving right r times, then left $2r$ times, then right r times, ending back at its starting node v_x. Let V be the subset of nodes in $\mathcal{B}_r$ whose distance from v_x is an integer multiple of D. In its local memory, agent x creates a path graph G_x consisting of the nodes in V, with two nodes connected by an edge if and only if their distance in $\mathcal{B}_r$ is exactly D. This forms a path graph centered at v_x with $\kappa \log^*(v_x)$ nodes in each direction. Next, the agent simulates an execution of the algorithm EarlyStopCV by the nodes of G_x to assign a colour $c_x \in \{0, 1, 2\}$ to its starting node v_x.

Stage 2: Search. The agent repeatedly executes periods consisting of $8D$ rounds each, partitioned into two equal-sized blocks of $4D$ rounds each. In each of the two blocks, the agent behaves in one of two ways: if a block is designated as a *waiting block*, then the agent stays at its starting node for all $4D$ rounds; otherwise, a block is designated as a *searching block*, in which the agent moves right D times, then left $2D$ times, then right D times. Whether a block is designated as a "waiting" or "searching" block depends on the agent's starting node colour that was determined in the first stage. In particular, if $c_x = 0$, then both blocks are waiting blocks; if $c_x = 1$, then block 1 is a searching block and block 2 is a waiting block; and, if $c_x = 2$, then both blocks are searching blocks.

4.2 Analysis of the algorithm

In this section, we prove that Algorithm $\mathcal{A}_{\text{rv-D}}$ solves rendezvous within $O(D \log^* \ell)$ rounds, where ℓ is the larger of the labels of the two starting nodes of the agents.

Consider an arbitrary instance on some line L with an arbitrary labeling of the nodes with positive integers. Suppose that two agents α and β execute the algorithm. For each $x \in \{\alpha, \beta\}$, we denote by v_x the label of agent x's starting node, and we denote by c_x the colour assigned to node v_x at the end of Stage 1 in x's execution of the algorithm. To help with the wording of the analysis only, fix a global orientation for L so that v_α appears to the "left" of v_β (and recall that the agents have no access to this information).

First, we argue that after both agents have finished Stage 1 of their executions, they have assigned different colours to their starting nodes.

▶ **Lemma 4.** *In any execution of $\mathcal{A}_{rv\text{-}D}$ by agents α and β, in every round after both agents finish their execution of Stage 1, we have $c_\alpha \neq c_\beta$.*

Proof. Without loss of generality, assume that $v_\alpha > v_\beta$. Let y_0 be the node in L to the left of v_α at distance exactly $D \cdot \kappa \log^*(v_\alpha)$. For each $i \in \{1, \ldots, 2\kappa \log^*(v_\alpha) + 1\}$, let y_i be the node in L at distance $i \cdot D$ to the right of y_0. Note that $v_\alpha = y_{\kappa \log^*(v_\alpha)}$ and $v_\beta = y_{\kappa \log^*(v_\alpha)+1}$.

Create a path graph P consisting of $2\kappa \log^*(v_\alpha) + 2$ nodes. Label the leftmost node in P with y_0, and label the node at distance i from y_0 in P using the label y_i.

By the definition of P, note that $y_{\kappa \log^*(v_\alpha)}$ and $y_{\kappa \log^*(v_\alpha)+1}$ are neighbours in P, which implies that v_α and v_β are neighbours in P. This is important because it implies that, if the nodes of P run the algorithm EarlyStopCV from Proposition 1, the nodes labeled v_α and v_β will choose different colours from the set $\{0, 1, 2\}$.

The rest of the proof shows that, for $x \in \{\alpha, \beta\}$, the graph G_x built in Stage 1 by agent x is an induced subgraph of P. This is sufficient since it implies that having each agent x simulate the algorithm EarlyStopCV on their local G_x results in the same colour assignment to the node labeled v_x as an execution of EarlyStopCV on the nodes of P, so v_α and v_β will be assigned different colours at the end of Stage 1 of Algorithm $\mathcal{A}_{\text{rv-D}}$. Since the colour assignment is not changed in any round after Stage 1, the result follows.

First, consider v_α. Let w_0 be the label of the leftmost node of G_α. By the definition of G_α, the node labeled w_0 is at distance exactly $\kappa \log^*(v_\alpha)$ to the left of v_α in G_α, so the node labeled w_0 is at distance exactly $D \cdot \kappa \log^*(v_\alpha)$ to the left of v_α in L. This proves that $w_0 = y_0 \in P$. For each $j \in \{0, \ldots, 2\kappa \log^*(v_\alpha)\}$, define w_j to be the label of the node at distance j to the right of the node labeled w_0 in G_α. By induction on j, we prove that $w_j = y_j \in P$ for all $j \in \{0, \ldots, 2\kappa \log^*(v_\alpha)\}$. The base case $w_0 = y_0 \in P$ was proved above. As induction hypothesis, assume that $w_{j-1} = y_{j-1} \in P$ for some $j \in \{1, \ldots, 2\kappa \log^*(v_\alpha)\}$. Consider w_j, which by definition of G_α is the neighbour to the right of w_{j-1} in G_α, and, moreover, is located distance exactly D to the right of w_{j-1} in L. By the induction hypothesis, we know that

$w_{j-1} = y_{j-1}$, so w_j is located distance exactly D to the right of y_{j-1} in L, and so $w_j = y_j$ by the definition of y_j. To confirm that $y_j \in P$, we note that $j \leq 2\kappa \log^*(v_\alpha) < 2\kappa \log^*(v_\alpha) + 1$, and that the rightmost node in P is $y_{2\kappa \log^*(v_\alpha)+1}$. This concludes the inductive step, and the proof that G_α is an induced subgraph of P.

Next, consider v_β. Let u_0 be the label of the leftmost node of G_β. By the definition of G_β, the node labeled u_0 in G_β is at distance exactly $\kappa \log^*(v_\beta)$ to the left of v_β, so, in L, the node labeled u_0 is at distance exactly $D \cdot \kappa \log^*(v_\beta)$ to the left of v_β. However, since v_α is located at distance exactly D to the left of v_β in L, and y_0 is located at distance exactly $D \cdot \kappa \log^*(v_\alpha)$ to the left of v_α in L, it follows that y_0 is located at distance exactly $D \cdot (1 + \kappa \log^*(v_\alpha))$ to the left of v_β in L. Since $v_\alpha > v_\beta$, we conclude that $d(y_0, v_\beta) = D \cdot (1 + \kappa \log^*(v_\alpha)) > D \cdot \kappa \log^*(v_\beta) = d(u_0, v_\beta)$ in L, so y_0 must be to the left of u_0 in L. Further, it means that $d(u_0, y_0) = D \cdot (1 + \kappa \log^*(v_\alpha)) - D \cdot \kappa \log^*(v_\beta) = D \cdot [1 + \kappa \log^*(v_\alpha) - \kappa \log^*(v_\beta)]$ in L. This proves that $u_0 = y_{1+\kappa \log^*(v_\alpha)-\kappa \log^*(v_\beta)}$. We confirm that $y_{1+\kappa \log^*(v_\alpha)-\kappa \log^*(v_\beta)} \in P$ by noticing that the subscript $1 + \kappa \log^*(v_\alpha) - \kappa \log^*(v_\beta)$ lies in the set $\{1, \ldots, 2\kappa \log^*(v_\alpha) + 1\}$ since $v_\alpha > v_\beta$. Next, define $h = 1 + \kappa \log^*(v_\alpha) - \kappa \log^*(v_\beta)$, and, for each $j \in \{0, \ldots, 2\kappa \log^*(v_\beta)\}$, define u_j to be the label of the node at distance j to the right of the node labeled u_0 in G_β. By induction on j, we prove that $u_j = y_{j+h} \in P$ for all $j \in \{0, \ldots, 2\kappa \log^*(v_\beta)\}$. The base case $u_0 = y_h \in P$ was proved above. As induction hypothesis, assume that $u_{j-1} = y_{j-1+h} \in P$ for some $j \in \{1, \ldots, 2\kappa \log^*(v_\beta)\}$. Consider u_j, which by definition of G_β is the neighbour to the right of u_{j-1} in G_β, and, moreover, is located distance exactly D to the right of u_{j-1} in L. By the induction hypothesis, we know that $u_{j-1} = y_{j-1+h}$, so u_j is located distance exactly D to the right of y_{j-1+h} in L, and so $u_j = y_{j+h}$ by the definition of y_h. To confirm that $y_{j+h} \in P$, we note that $j \leq 2\kappa \log^*(v_\beta)$ and $h = 1 + \kappa \log^*(v_\alpha) - \kappa \log^*(v_\beta)$, so $j + h \leq 1 + \kappa \log^*(v_\alpha) + \kappa \log^*(v_\beta) \leq 2\kappa \log^*(v_\alpha) + 1$, where the last inequality is due to $v_\alpha > v_\beta$. As the rightmost node of P is $y_{2\kappa \log^*(v_\alpha)+1}$, it follows that y_{j+h} is at or to the left of the rightmost node in P, so $y_{j+h} \in P$. This concludes the inductive step, and the proof that G_α is an induced subgraph of P. ◀

To prove that the algorithm correctly solves rendezvous within $O(D \log^* \ell)$ rounds for arbitrary delay between starting rounds, there are two main cases to consider. If the delay is large, then the late agent is idling for the early agent's entire execution of Stage 1, and rendezvous will occur while the early agent is exploring its $(D\kappa \log^* \ell)$-neighbourhood. Otherwise, the delay is relatively small, so both agents reach Stage 2 quickly, and the block structure of the repeated periods ensures that one agent will search while the other waits, so rendezvous will occur. These arguments are formalized in the following result.

▶ **Theorem 5.** *Algorithm $\mathcal{A}_{rv\text{-}D}$ solves rendezvous in $O(D \log^* \ell)$ rounds on lines with arbitrary node labelings when two agents start at arbitrary positions at known distance D, and when the delay between the rounds in which they start executing the algorithm is arbitrary.*

Proof. The agent that starts executing the algorithm first is called the *early agent*, and the other agent is called the *late agent*. If both agents start executing the algorithm in the same round, then arbitrarily call one of them early and the other one late. Without loss of generality, we assume that α is the early agent. The number of rounds that elapse between the two starting rounds is denoted by $delay(\alpha, \beta)$.

First, we address the case where the delay between the starting rounds of the agents is large. This situation is relatively easy to analyze, since agent β remains idle during agent α's entire execution up until they meet.

▷ Claim 6. If $delay(\alpha, \beta) > 4D\kappa \log^* v_\alpha$, then rendezvous occurs within $4D\kappa \log^* v_\alpha$ rounds in agent α's execution.

To prove the claim, we note that the total number of rounds that elapse in agent α's execution of Stage 1 is $4D\kappa \log^* v_\alpha$. By the assumption about $delay(\alpha, \beta)$, agent β is idle at its starting node for α's entire execution of Stage 1. Finally, note that α explores its entire $(D\kappa \log^* \alpha)$-neighbourhood during its execution of Stage 1, where $D\kappa \log^* \alpha \geq D$, so it follows that agent α will be located at agent β's starting node at some round in its execution of Stage 1, which completes the proof of the claim.

So, in the remainder of the proof, we assume that $delay(\alpha, \beta) \leq 4D\kappa \log^* \alpha$. We prove that rendezvous occurs during Stage 2 by considering the round τ in which agent β starts Stage 2 of its execution. By Lemma 4, we have $c_\alpha \neq c_\beta$ in every round from this time onward. We separately consider cases based on the values of c_α and c_β.

- **Case 1: $\mathbf{c_\alpha = 0}$ or $\mathbf{c_\beta = 0}$.** Let x be the agent with $c_x = 0$, and let y be the other agent. Since $c_x \neq c_y$, we know that c_y is either 1 or 2. From the algorithm's description, agent y will start a searching block within $8D$ rounds after round τ. Moreover, agent x remains idle for the entirety of Stage 2 of its execution. Since y explores its entire D-neighbourhood during a searching block, it follows that y will be located at x's starting node within $12D$ rounds after τ.
- **Case 2: $\mathbf{c_\alpha = 1}$ or $\mathbf{c_\beta = 1}$.** Let x be the agent with $c_x = 1$, and let y be the other agent. Since $c_x \neq c_y$, we know that c_y is either 0 or 2. As the case $c_y = 0$ is covered by Case 1 above, we proceed with $c_y = 2$. From the algorithm's description, agent x has a waiting block that begins within $8D$ rounds after τ. Suppose that this waiting block starts in some round t of x's execution, then we know that x stays idle at its starting node for the $4D$ rounds after t. But round t corresponds to the i'th round of some searching block in agent y's execution of Stage 2 (since y only performs searching blocks). In the first $4D - i$ rounds, agent y completes its searching block, and then performs the first i rounds of the next searching block. Since the searching blocks are performed using the same movements each time, it follows that y explores its entire D-neighbourhood in the $4D$ rounds after t, so will meet x at x's starting node within those rounds. Altogether, since t occurs within $8D$ rounds after τ, and y's searching block takes another $4D$ rounds, it follows that rendezvous occurs within $12D$ rounds after τ.

In all cases, we proved that rendezvous occurs within $12D$ rounds after τ. But τ is the round in which agent β starts Stage 2, so $\tau \leq 4D\kappa \log^* v_\beta$ in β's execution. By our assumption on the delay, β's execution starts at most $4D\kappa \log^* v_\alpha$ rounds after the beginning of α's execution. Altogether, this means that rendezvous occurs within $4D\kappa \log^* v_\alpha + 4D\kappa \log^* v_\beta + 12D$ rounds from the start of α's execution. Setting $\ell = \max\{v_\alpha, v_\beta\}$, we get that rendezvous occurs within time $8D\kappa \log^* \ell + 12D \in O(D \log^* \ell)$, as desired. ◀

4.3 Lower bound

In this section we prove a $\Omega(D \log^* \ell)$ lower bound for rendezvous time on the infinite line, where ℓ is the larger label of the two starting nodes, even assuming that agents start simultaneously, they know the initial distance D between them, and they have a global sense of direction. We summarize the argument here, and omit the detailed proofs of the results. We start with some terminology about rendezvous executions.

▶ **Definition 7.** *Consider any labeled infinite line L, and any two nodes labeled v, w on L that are at fixed distance D. Consider any rendezvous algorithm $\mathcal{A}$, and suppose that two agents start executing $\mathcal{A}$ in the same round: one agent α_v starting at node v, and the other agent α_w starting at node w. Denote by $\gamma(\mathcal{A}, L, v, w)$ the resulting execution until α_v and*

α_w meet. When $\mathcal{A}$ and L are clear from the context, we will simply write $\gamma(v, w)$ to denote the execution. Denote by $|\gamma(\mathcal{A}, L, v, w)|$ the number of rounds that have elapsed before α_v and α_w meet in the execution.

The following definition formalizes the notion of "behaviour sequence": an integer sequence that encodes the movements made by an agent in each round of an algorithm's execution.

▶ **Definition 8.** *Consider any execution $\gamma(\mathcal{A}, L, v, w)$ by an agent α_v starting at node v and an agent α_w starting at node w, both agents starting simultaneously. Define the behaviour sequence $\mathcal{B}_v(\mathcal{A}, L, v, w)$ as follows: for each $t \in \{1, \ldots, |\gamma(\mathcal{A}, L, v, w)|\}$, set the t'th element to 0 if α_v moves left in round t of the execution, to 1 if α_v stays at its current node in round t of the execution, and to 2 if α_v moves right in round t of the execution. Similarly, define the sequence $\mathcal{B}_w(\mathcal{A}, L, v, w)$ using the moves by agent α_w. When $\mathcal{A}$ and L are clear from the context, we will simply write $\mathcal{B}_v(v, w)$ and $\mathcal{B}_w(v, w)$ to denote the two behaviour sequences of α_v and α_w, respectively.*

As the agents are anonymous and we only consider deterministic algorithms, note that for any fixed $\mathcal{A}, L, v, w$, we have $\gamma(v, w) = \gamma(w, v)$ and $\mathcal{B}_v(v, w) = \mathcal{B}_v(w, v)$ and $\mathcal{B}_w(v, w) = \mathcal{B}_w(w, v)$. Moreover, for a fixed starting node v on a fixed line L, the behaviour of an agent running an algorithm $\mathcal{A}$ does not depend on the starting node (or behaviour) of the other agent, until the two agents meet. This implies the following result, i.e., if we look at two executions of $\mathcal{A}$ where one agent α_v starts at the same node v in both executions, then α_v's behaviour in both executions is exactly the same up until rendezvous occurs in the shorter execution.

▶ **Proposition 9.** *Consider any labeled infinite line L, and any fixed rendezvous algorithm $\mathcal{A}$. Consider any fixed node v in L, and let w_1 and w_2 be two nodes other than v. Let $p = \min\{|\gamma(v, w_1)|, |\gamma(v, w_2)|\}$. Then $\mathcal{B}_v(v, w_1)$ and $\mathcal{B}_v(v, w_2)$ have equal prefixes of length p.*

The following proposition states that two agents running a rendezvous algorithm starting at two different nodes cannot have the same behaviour sequence. This follows from the fact that the distance between two agents cannot decrease if they perform the same action in each round (i.e., both move left, both move right, or both don't move).

▶ **Proposition 10.** *Consider any labeled infinite line L, and any fixed rendezvous algorithm $\mathcal{A}$. For any two nodes x and y in L, in the execution $\gamma(x, y)$ we have $\mathcal{B}_x(x, y) \neq \mathcal{B}_y(x, y)$.*

The remainder of this section is dedicated to proving the $\Omega(D \log^* \ell)$ lower bound for rendezvous on the infinite line when the two agents start at a known distance D apart. We proceed in two steps: first, we prove an $\Omega(\log^* \ell)$ lower bound in the case where $D = 1$, and then we prove the general $\Omega(D \log^* \ell)$ lower bound using a reduction from the $D = 1$ case. Throughout this section, we will refer to the constant κ that was defined in Proposition 1 in order to state the running time bound of the algorithm EarlyStopCV.

4.3.1 The $D = 1$ case

We prove a $\Omega(\log^* \ell)$ lower bound for rendezvous on the infinite line, where ℓ is the larger label of the two starting nodes, in the special case where the two agents start at adjacent nodes. This lower bound applies even to algorithms that start simultaneously and know that the initial distance between the two agents is 1.

The overall idea is to assume that there exists a very fast rendezvous algorithm (i.e., an algorithm that always terminates within $\frac{1}{16\kappa} \log^*(\ell)$ rounds) and prove that this implies the existence of a distributed 3-colouring algorithm for the $\mathcal{LOCAL}$ model whose running time is faster than the lower bound proven by Linial (see Proposition 2). This contradiction proves that any rendezvous algorithm must have running time $\Omega(\log^*(\ell))$.

The first step is to reduce distributed colouring in the $\mathcal{LOCAL}$ model to rendezvous. The following result describes how to use the rendezvous algorithm to create the distributed colouring algorithm. The idea is to record the agent's behaviour sequence in the execution of the rendezvous algorithm, and convert the sequence to an integer colour. Proposition 10 guarantees that the assigned colours are different.

▶ **Lemma 11.** *Consider any rendezvous algorithm $\mathcal{A}_{rv}$ that always terminates within $\frac{1}{16\kappa}\log^*(\ell)$ rounds, where ℓ is the larger label of the two starting nodes. Then there exists a distributed colouring algorithm $\mathcal{A}_{col}$ such that, for any labeled infinite line L, and for any finite subline P of L consisting of nodes whose labels are bounded above by some integer Y, algorithm $\mathcal{A}_{col}$ uses $\lfloor\frac{1}{16\kappa}\log^*(Y)\rfloor+1$ rounds of communication and assigns to each node in P an integer colour from the range $1,\ldots,4^{2\lfloor\frac{1}{16\kappa}\log^*(Y)\rfloor+1}$.*

The second step is to take the algorithm $\mathcal{A}_{\text{col}}$ from Lemma 11 and turn it into a 3-colouring algorithm $\mathcal{A}_{3\text{col}}$ using very few additional rounds, by using the algorithm EarlyStopCV from Proposition 1 to quickly reduce the number of colours down to 3. Combined with the previous lemma, we get the following result that shows how to obtain a very fast distributed 3-colouring algorithm under the assumption that we have a very fast rendezvous algorithm.

▶ **Lemma 12.** *Consider any rendezvous algorithm $\mathcal{A}_{rv}$ that always terminates within $\frac{1}{16\kappa}\log^*(\ell)$ rounds, where ℓ is the larger label of the two starting nodes. Then there exists a distributed 3-colouring algorithm $\mathcal{A}_{3col}$ such that, for any labeled infinite line L, and for any finite subline P of L consisting of nodes whose labels are bounded above by some integer Y, algorithm $\mathcal{A}_{3col}$ uses at most $\left(\frac{1}{4}+\frac{1}{16\kappa}\right)\log^*(Y)+1+3\kappa$ rounds of communication to 3-colour the nodes of P.*

Finally, we demonstrate how to use the above result to prove the desired $\Omega(\log^*\ell)$ lower bound for rendezvous. The idea is to construct an infinite line that contains an infinite sequence of finite sublines, each of which is a worst-case instance (according to Linial's lower bound), and obtaining the desired contradiction by observing that the upper bound on the running time of $\mathcal{A}_{3\text{col}}$ violates the lower bound guaranteed by Linial's result.

▶ **Lemma 13.** *Any algorithm that solves the rendezvous task on all labeled infinite lines, where the two agents start at adjacent nodes, uses $\Omega(\log^*\ell)$ rounds in the worst case, where ℓ is the larger label of the two starting nodes.*

4.3.2 The $D > 1$ case

We prove a $\Omega(D\log^*\ell)$ lower bound for rendezvous on the infinite line, where ℓ is the larger label of the two starting nodes, in the case where the two agents start at nodes that are distance $D>1$ apart. This lower bound applies even to algorithms that start simultaneously and know that the initial distance between the two agents is D. Hence it shows that the running time of Algorithm $\mathcal{A}_{\text{rv-D}}$ has optimal order of magnitude among rendezvous algorithms knowing the initial distance between the agents.

The overall idea is to assume that there exists a very fast rendezvous algorithm called $\mathcal{A}_{\text{rv}}$ (that always terminates within $\frac{1}{224\kappa}D\log^*\ell$ rounds) and prove that this implies the existence of a rendezvous algorithm $\mathcal{A}_{\text{rv-adj}}$ for the $D=1$ case that always terminates within $\frac{1}{16\kappa}\log^*\ell$ rounds, which we already proved is impossible in Section 4.3.1. This contradiction proves that any rendezvous algorithm for the $D>1$ case must have running time $\Omega(D\log^*\ell)$.

The proof can be summarized as follows: take any instance where the agents start at adjacent nodes, and "blow it up" by a factor of D, i.e., all node labels are multiplied by a factor of D, and $D-1$ "dummy nodes" are inserted between each pair of nodes. Each node from the original instance, together with the $D-1$ nodes to its right, are called a *segment* in the blown-up instance. Then, the algorithm $\mathcal{A}_{\text{rv}}$ is locally simulated on the blown-up instance in *stages* consisting of D rounds each, and each simulated stage corresponds to 1 round of algorithm $\mathcal{A}_{\text{rv-adj}}$ in the original instance. At the end of every simulated stage, the simulated agent is in some segment that has leftmost node v, so the real agent situates itself at node v for the corresponding round in the original instance. Roughly speaking, since $\mathcal{A}_{\text{rv}}$ guarantees rendezvous in the blown-up instance, the two simulated agents will end up in the same segment (with the same leftmost node v), so the two real agents will end up at node v in the original instance. Further, since each stage of D simulated rounds corresponds to one round in the original instance, the number of rounds used by $\mathcal{A}_{\text{rv-adj}}$ is a factor of D less than the running time of the simulated algorithm $\mathcal{A}_{\text{rv}}$. This gives us a contradiction, as the resulting running time for $\mathcal{A}_{\text{rv-adj}}$ is smaller than the lower bound proven in Lemma 13.

However, the above summary overlooks some complications.

1. **Assigning labels to the dummy nodes in the blown-up instance:** the agent in the original instance needs to assign labels to nearby dummy nodes in the blown-up instance in a way that is consistent with the original instance. However, the agent initially only knows the label at its own node, which is insufficient. To address this issue, before each simulated stage of $\mathcal{A}_{\text{rv}}$, the agent uses 4 rounds to visit its neighbouring nodes in the original instance so that it can learn about neighbouring labels, which it then uses to accurately assign labels to nearby dummy nodes in the blown-up instance.
2. **Guaranteeing rendezvous:** two agents running $\mathcal{A}_{\text{rv-adj}}$ are each independently simulating $\mathcal{A}_{\text{rv}}$ locally in their own memory, so they cannot detect if the simulated agents meet at a node in the blown up instance. So, although $\mathcal{A}_{\text{rv}}$ guarantees rendezvous in the blown up instance, it might be the case that there is never a stage of D simulated rounds after which the two simulated agents end up in the same segment, since the simulated agents might continue moving after the undetected rendezvous and end up in different segments at the end of the simulated stage. To address this issue, we carefully choose the segment length and stage length appropriately to guarantee that, at the end of the stage containing the undetected rendezvous, the two simulated agents are either in the same segment or in neighbouring segments in the blown up instance. After each simulated stage is done, the real agents do a 3-round "dance" in the original instance in such a way that rendezvous will occur after the undetected simulated rendezvous.

▶ **Theorem 14.** *Any algorithm that solves the rendezvous task on all labeled infinite lines, where the two agents start at known distance $D > 1$ apart, uses $\Omega(D \log^* \ell)$ rounds in the worst case, where ℓ is the larger label of the two starting nodes.*

5 Arbitrary lines with unknown initial distance between agents

In this section, we describe an algorithm called $\mathcal{A}_{\text{rv-noD}}$ that solves rendezvous on lines with arbitrary node labelings in time $O(D^2(\log^* \ell)^3)$ (where D is the initial distance between the agents and ℓ is the larger label of the two starting nodes) when two agents start at arbitrary positions and when the delay between the rounds in which they start executing the algorithm is arbitrary. The agents do not know the initial distance D between them, and they do not know the delay between the starting rounds. Also, we note that the agents have no global sense of direction, but each agent can locally choose port 0 from its starting node to represent

"right" and port 1 from its starting node to represent "left". Further, using knowledge of the port number of the edge on which it arrived at a node, an agent is able to choose whether its next move will continue in the same direction or if it will switch directions. Without loss of generality, we may assume that all node labels are strictly greater than one, since the algorithm could be re-written to add one to each label value in its own memory before carrying out any computations involving the labels. This assumption ensures that, for any node label v, the value of $\log^*(v)$ is strictly greater than 0.

As seen in Section 4.1, if the initial distance D between the agent is known, then we have an algorithm $\mathcal{A}_{\text{rv-D}}$ that will solve rendezvous in $O(D \log^* \ell)$ rounds. We wish to extend that algorithm for the case of unknown distance by repeatedly running $\mathcal{A}_{\text{rv-D}}$ with guessed values for D. To get an optimal algorithm, i.e., with running time $O(D \log^* \ell)$, a natural attempt would be to proceed by doubling the guess until it exceeds D, so that the searching range of an agent includes the starting node of the other agent. However, this approach will not work in our case, because our algorithm $\mathcal{A}_{\text{rv-D}}$ requires the exact value of D to guarantee rendezvous. More specifically, the colouring stage using guess g only guarantees that nodes at distance exactly g are assigned different colours. So, instead, our algorithm $\mathcal{A}_{\text{rv-noD}}$ increments the guessed value by 1 so that the guess is guaranteed to eventually be equal to D, which results in a running time quadratic in D instead of linear in D.

At a high level, our algorithm $\mathcal{A}_{\text{rv-noD}}$ consists of phases, where each phase is an attempt to solve rendezvous using a value g which is a guess for the value D. The first phase sets $g = 1$. Each phase has three stages. In the first stage, the agent waits at its starting node for a fixed number of rounds. In the second and third stages, the agent executes a modified version of the algorithm $\mathcal{A}_{\text{rv-D}}$ from Section 4.1 using the value g instead of D. At the end of each phase, if rendezvous has not yet occurred, the agent increments its guess g and proceeds to the next phase. A major complication is that an adversary can choose the wake-up times of the agents so that the phases do not align well, e.g., the agents are using the same guess g but are at different parts of the phase, or, they are in different phases and not using the same guess g. This means we have to very carefully design the phases and algorithm analysis to account for arbitrary delays between the wake-up times.

The detailed description of the algorithm is as follows. Consider an agent x whose execution starts at a node labeled v_x. We now describe an arbitrary phase in the algorithm's execution. Let $g \geq 1$, let $d = 1 + \lfloor \log_2 g \rfloor$, and recall that $\kappa > 1$ is an integer constant defined in the running time of the algorithm EarlyStopCV from Proposition 1. The g-th phase executed by agent x, denoted by $\mathcal{P}_g^x$, consists of executing the following three stages.

Stage 0: Wait. Stay at the node v_x for $36 \cdot 2^d \cdot \kappa \log^*(v_x)$ rounds.

Stage 1: Colouring. Let $r = g \cdot \kappa \log^*(v_x)$. Denote by $\mathcal{B}_r$ the r-neighbourhood of v_x, and let V_g be the subset of nodes in $\mathcal{B}_r$ whose distance from v_x is an integer multiple of g. First, agent x determines $\mathcal{B}_r$ (including all node labels) by moving right r times, then left $2r$ times, then right r times, ending back at its starting node v_x. Then, in its local memory, agent x creates a path graph G_x consisting of the nodes in V_g, with two nodes connected by an edge if and only if their distance in $\mathcal{B}_r$ is exactly g. This forms a path graph centered at v_x with $\kappa \log^*(v_x)$ nodes in each direction. The agent simulates an execution of the algorithm EarlyStopCV from Proposition 1 by the nodes of G_x to obtain a colour $c_x \in \{0, 1, 2\}$. Let CV_x be the 2-bit binary representation of c_x. Transform CV_x into an 8-bit binary string CV'_x by replacing each 0 in CV_x with 0011, and replacing each 1 in CV_x with 1100. Finally, create a 9-bit string S by appending a 1 to CV'_x.

Stage 2: Search. This stage consists of performing $\kappa \log^*(v_x)$ periods of $|S| = 9$ blocks each. Block i of a period is designated as a *waiting* block if the i'th bit of S is 0, else it is designated as a *searching* block. A waiting block consists of $4 \cdot (2^d)$ consecutive rounds during which the agent stays at its starting node. A searching block consists of $4 \cdot (2^d)$ consecutive rounds: the agent first moves right 2^d times, then left $2 \cdot 2^d$ times, then right 2^d times.

6 Conclusion

We presented rendezvous algorithms for three scenarios: the scenario of the canonical line, the scenario of arbitrary labeling with known initial distance D, and the scenario where each agent knows *a priori* only the label of its starting node. While for the first two scenarios the complexity of our algorithms is optimal (respectively $O(D)$ and $O(D \log^* \ell)$, where ℓ is the larger label of the two starting nodes), for the most general scenario, where each agent knows *a priori* only the label of its starting node, the complexity of our algorithm is $O(D^2(\log^* \ell)^3)$, for arbitrary unknown D, while the best known lower bound, valid also in this scenario, is $\Omega(D \log^* \ell)$.

The natural open problem is the optimal complexity of rendezvous in the most general scenario (with arbitrary labeling and unknown D), both for the infinite labeled line and for the finite labeled lines and cycles. This open problem can be generalized to the class of arbitrary trees or even arbitrary graphs.

References

1 Steve Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683, 1995. doi:10.1137/S0363012993249195.

2 Steve Alpern. Rendezvous search on labeled networks. *Naval Research Logistics (NRL)*, 49(3):256–274, 2002. doi:10.1002/nav.10011.

3 Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*, volume 55 of *International series in operations research and management science*. Kluwer, 2003.

4 E. J. Anderson and R. R. Weber. The rendezvous problem on discrete locations. *Journal of Applied Probability*, 27(4):839–851, 1990. URL: http://www.jstor.org/stable/3214827.

5 Edward J. Anderson and Sándor P. Fekete. Asymmetric rendezvous on the plane. In Ravi Janardan, editor, *Proceedings of the Fourteenth Annual Symposium on Computational Geometry, Minneapolis, Minnesota, USA, June 7-10, 1998*, pages 365–373. ACM, 1998. doi:10.1145/276884.276925.

6 Edward J. Anderson and Sándor P. Fekete. Two dimensional rendezvous search. *Oper. Res.*, 49(1):107–118, 2001. doi:10.1287/opre.49.1.107.11191.

7 Vic Baston and Shmuel Gal. Rendezvous on the line when the players' initial distance is given by an unknown probability distribution. *SIAM Journal on Control and Optimization*, 36(6):1880–1889, 1998. doi:10.1137/S0363012996314130.

8 Vic Baston and Shmuel Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics*, 48(8):722–731, December 2001. doi:10.1002/nav.1044.

9 Subhash Bhagat and Andrzej Pelc. Deterministic rendezvous in infinite trees. *CoRR*, abs/2203.05160, 2022. doi:10.48550/arXiv.2203.05160.

10 Subhash Bhagat and Andrzej Pelc. How to meet at a node of any connected graph. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.DISC.2022.11.

11 Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc, and Franck Petit. Almost universal anonymous rendezvous in the plane. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 117–127. ACM, 2020. doi:10.1145/3350755.3400283.

12 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control.*, 70(1):32–53, 1986. doi:10.1016/S0019-9958(86)80023-7.

13 Andrew Collins, Jurek Czyzowicz, Leszek Gasieniec, Adrian Kosowski, and Russell A. Martin. Synchronous rendezvous for location-aware agents. In David Peleg, editor, *Distributed Computing - 25th International Symposium, DISC 2011, Rome, Italy, September 20-22, 2011. Proceedings*, volume 6950 of *Lecture Notes in Computer Science*, pages 447–459. Springer, 2011. doi:10.1007/978-3-642-24100-0_42.

14 Jurek Czyzowicz, Leszek Gasieniec, Ryan Killick, and Evangelos Kranakis. Symmetry breaking in the plane: Rendezvous by robots with unknown attributes. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 4–13. ACM, 2019. doi:10.1145/3293611.3331608.

15 Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Comput.*, 25(2):165–178, 2012. doi:10.1007/s00446-011-0141-9.

16 Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006. doi:10.1007/s00453-006-0074-2.

17 Yoann Dieudonné and Andrzej Pelc. Anonymous meeting in networks. *Algorithmica*, 74(2):908–946, 2016. doi:10.1007/s00453-015-9982-0.

18 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005. doi:10.1016/j.tcs.2005.01.001.

19 Pierre Fraigniaud and Andrzej Pelc. Delays induce an exponential memory gap for rendezvous in trees. *ACM Trans. Algorithms*, 9(2):17:1–17:24, 2013. doi:10.1145/2438645.2438649.

20 Dariusz R. Kowalski and Adam Malinowski. How to meet in anonymous network. *Theor. Comput. Sci.*, 399(1-2):141–156, 2008. doi:10.1016/j.tcs.2008.02.010.

21 Evangelos Kranakis, Danny Krizanc, and Pat Morin. Randomized rendezvous with limited memory. *ACM Trans. Algorithms*, 7(3):34:1–34:12, 2011. doi:10.1145/1978782.1978789.

22 Evangelos Kranakis, Nicola Santoro, Cindy Sawchuk, and Danny Krizanc. Mobile agent rendezvous in a ring. In *23rd International Conference on Distributed Computing Systems (ICDCS 2003), 19-22 May 2003, Providence, RI, USA*, pages 592–599. IEEE Computer Society, 2003. doi:10.1109/ICDCS.2003.1203510.

23 Juhana Laurinharju and Jukka Suomela. Linial's lower bound made easy. *CoRR*, abs/1402.2552, 2014. URL: http://arxiv.org/abs/1402.2552, arXiv:1402.2552.

24 Wei Shi Lim and Steve Alpern. Minimax rendezvous on the line. *SIAM Journal on Control and Optimization*, 34(5):1650–1665, 1996. doi:10.1137/S036301299427816X.

25 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. doi:10.1137/0221015.

26 Avery Miller and Andrzej Pelc. Tradeoffs between cost and information for rendezvous and treasure hunt. *J. Parallel Distributed Comput.*, 83:159–167, 2015. doi:10.1016/j.jpdc.2015.06.004.

27 Andrzej Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012. doi:10.1002/net.21453.

28 Andrzej Pelc. Deterministic rendezvous algorithms. In Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro, editors, *Distributed Computing by Mobile Entities, Current Research in Moving and Computing*, volume 11340 of *Lecture Notes in Computer Science*, pages 423–454. Springer, 2019. doi:10.1007/978-3-030-11072-7_17.

29 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000. doi:10.1137/1.9780898719772.

30 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Trans. Algorithms*, 10(3):12:1–12:15, 2014. doi:10.1145/2601068.

31 L. C. Thomas. Finding your kids when they are lost. *Journal of the Operational Research Society*, 43(6):637–639, 1992. doi:10.1057/jors.1992.89.

32 Roger Wattenhofer. Principles of distributed computing, 2023. Accessed on 2023-04-16. URL: https://disco.ethz.ch/courses/fs23/podc/.

A The Line Colouring Algorithm

As announced in Proposition 1 in Section 2, we design a deterministic distributed algorithm EarlyStopCV that, in the $\mathcal{LOCAL}$ model, properly 3-colours any infinite labeled line such that the execution of this algorithm at any node x with initial label ID_x terminates in time $O(\log^*(\mathrm{ID}_x))$. Our algorithm builds upon a solution to a problem posed in Homework Exercises 1 from the Principles of Distributed Computing course at ETH Zürich [32].

At a very high level, we want to execute the Cole and Vishkin algorithm [12], but with some modifications. We introduce four special colours, denoted by $\alpha, \beta, Pdone, Cdone$, that are defined in such a way that they are not equal to any colour that could be chosen using the Cole and Vishkin strategy (e.g., they could be defined as negative integers). These colours will be chosen by a node in certain situations where it needs to choose a colour that is guaranteed to be different from its neighbour's, but using the Cole and Vishkin strategy might not work. Another significant modification is that each node will actually choose two colours: its "final colour" that it will output upon terminating the algorithm, but also an intermediate "Phase 1 colour". These roughly correspond to the two parts of the Cole and Vishkin strategy: in Phase 1, each node picks a colour that is guaranteed to be different from each of its neighbours' chosen colour (but selects it from a "large" range of colours), and then in Phase 2, each node picks a final colour from the set $\{0, 1, 2\}$. However, since we want to allow different nodes to execute different phases of the algorithm at the same time (since we want to allow them to terminate at different times), each node must maintain and advertise its "Phase 1" and "final" colours separately, so that another node that is still performing Phase 1 is basing its decisions on its neighbours' Phase 1 colours (and not their final colours). Finally, we note that the original Cole and Vishkin strategy is described for a directed tree, i.e., where each node has at most one parent and perhaps some children, whereas our algorithm must work in an undirected infinite line, so we introduce a pre-processing step (Phase 0) to set up parent/child relationships.

The first part of the algorithm partitions the undirected infinite line into directed sub-lines that will perform the rest of the algorithm in parallel. In round 1, each node shares its unique ID with both neighbours. In round 2 (also referred to as Phase 0), each node compares its own ID with those that it received from its neighbours in round 1. If a node determines that it is a *local minimum* (i.e., its ID is less than the ID's of its neighbours), then it picks the special colour α as its Phase 1 colour and proceeds directly to Phase 2 without performing Phase 1. If a node determines that it is a *local maximum* (i.e., its ID is greater than the ID's of its neighbours), then it picks the special colour β as its Phase 1 colour and proceeds directly to Phase 2 without performing Phase 1. All other nodes, i.e., those that are not a local minimum or a local maximum, pick their own ID as their Phase 1 colour and continue to perform Phase 1. Further, each such node also chooses one *parent* neighbour (the neighbour with smaller ID) and one *child* neighbour (the neighbour with larger ID). In particular, the

nodes that perform Phase 1 are each part of a directed sub-line that is bordered by local maxima/minima, so the sub-lines can all perform Phase 1 in parallel without worrying that the chosen colours will conflict with colours chosen in other sub-lines.

The second part of the algorithm, referred to as Phase 1, essentially implements the Cole and Vishkin strategy within each directed sub-line. The idea is that each node follows the Cole and Vishkin algorithm using its own Phase 1 colour and the Phase 1 colour of its parent until it has selected a Phase 1 colour in the range $\{0, \ldots, 51\}$ that is different from its parent's Phase 1 colour, and then the node will proceed to Phase 2 where it will pick a final colour in the range $\{0, 1, 2\}$. However, this idea must be modified to avoid three potential pitfalls:

1. Suppose that, in a round t, a node v has a parent or child that stopped performing Phase 1 in an earlier round (and possibly has finished Phase 2 and has stopped executing the algorithm). If node v still has a large colour in round t, i.e., not in the range $\{0, \ldots, 51\}$, but continues to use the Cole and Vishkin strategy, then there is no guarantee that the new colour it chooses will still be different than those of its neighbours. This is not an issue for the original Cole and Vishkin algorithm, since it was designed in such a way that all nodes execute the algorithm the exact same number of times (using *a priori* knowledge of the network size). To deal with this scenario, node v first looks at the Phase 1 colours that were most recently advertised by its parent and child, and sees if either of them is in the range $\{0, \ldots, 51\}$. If node v sees that its parent's Phase 1 colour is in the range $\{0, \ldots, 51\}$, then it knows that its parent is not performing Phase 1 in this round, so instead of following the Cole and Vishkin strategy, it immediately adopts the special colour *Pdone* as its Phase 1 colour and moves on to Phase 2 of the algorithm. On the other hand, if node v sees that its child's colour is in the range $\{0, \ldots, 51\}$, then it knows that its child is not performing Phase 1 in this round, so instead of following the Cole and Vishkin strategy, it immediately adopts the special colour *Cdone* as its Phase 1 colour and moves on to Phase 2 of the algorithm. By adopting the special colours in this way, node v's Phase 1 colour is guaranteed to be different than any non-negative integer colour that was previously adopted by its neighbours.
2. Suppose that a node v's parent has a Phase 1 colour equal to a special colour (i.e., one of $\alpha, \beta, Pdone, Cdone$). The Cole and Vishkin strategy is not designed to work with such special colours, so, a node v with such a parent will pretend that its parent has colour 0 instead. By doing this, it will choose some non-negative integer as dictated by the Cole and Vishkin strategy, and this integer is guaranteed to be different from all special colours, so v's chosen Phase 1 colour will be different from its parent's.
3. Suppose that a node v has a parent with an extremely large integer colour. The Cole and Vishkin strategy will make sure that v chooses a new Phase 1 colour that is different than the Phase 1 colour chosen by its parent, however, it is not guaranteed that this new colour is significantly smaller than the colour v started with. In particular, we want to guarantee that node v terminates Phase 1 within $\log^*(\mathrm{ID}_v)$ rounds, so we want node v's newly-chosen Phase 1 colour to be bounded above by a logarithmic function of its *own* colour in every round (and never depend on its parent's much larger colour). To ensure this, we apply a suffix-free encoding to the binary representation of each node's colour **before** applying the Cole and Vishkin strategy. Doing this guarantees that the smallest index where two binary representations of colours differ is bounded above by the length of the binary representation of the *smaller* colour.

When a node is ready to proceed to Phase 2, it has chosen a Phase 1 colour from the set $\{0, \ldots, 51\} \cup \{\alpha, \beta\} \cup \{Pdone, Cdone\}$, and its chosen Phase 1 colour is guaranteed to be different from the Phase 1 colours of its two neighbours.

The third part of the algorithm, referred to as Phase 2, uses a round-robin strategy over the 56 possible Phase 1 colours, which guarantees that any two neighbouring nodes pick their final colour in different rounds. In particular, when executing each round of Phase 2, a node calculates the current *token* value, which is defined as the current round number modulo 56. We assume that all nodes start the algorithm at the same time, so the current token value is the same at all nodes in each round. In each round, each node that is performing Phase 2 compares the token value to its own Phase 1 colour. If a node v's Phase 1 colour is in $\{0, \ldots, 51\}$ and is equal to the current token value, then it proceeds to choose its final colour (as described below). Otherwise, if a node v's Phase 1 colour is one of the special colours, it waits until the current token value is equal to a value that is dedicated to that special colour (i.e., 52 for α, 53 for β, 54 for *Pdone*, 55 for *Cdone*), then chooses its final colour in that round. To choose its final colour, node v chooses the smallest colour from $\{0, 1, 2\}$ that was never previously advertised as a final colour by its neighbours. Then, v immediately sends out a message to its neighbours to advertise the final colour that it chose, then v terminates. Since two neighbouring nodes are guaranteed to have different Phase 1 colours, they will choose their final colour in different rounds, so the later of the two nodes always avoids the colour chosen by the earlier node, and there is always a colour from $\{0, 1, 2\}$ available since each node only has two neighbours.

Algorithm pseudocode

For any two binary strings S_1, S_2, denote by $S_1 \cdot S_2$ the concatenation of string S_1 followed by string S_2. For any positive integer i, the function BinaryRep(i) returns the binary string consisting of the base-2 representation of i. Conversely, for any binary string S, the function IntVal(S) returns the integer value when S is interpreted as a base-2 integer. For any binary string S of length $\ell \geq 1$, the string is a concatenation of bits, i.e., $S = s_{\ell-1} \cdots s_0$, and we will write $S[i]$ to denote the bit s_i. The notation $|S|$ denotes the length of S.

For any binary string S of length $\ell \geq 1$, we define a function EncodeSF(S) that returns the binary string obtained by replacing each 0 in S with 01, replacing each 1 in S with 10, then prepending 00 to the result. More formally, EncodeSF(S) returns a string S' of length $2\ell + 2$ such that $S'[2\ell + 1] = S'[2\ell] = 0$, and, for each $i \in \{0, \ldots, \ell - 1\}$, $S'[2i + 1] = S[i]$ and $S'[2i] = 1 - S[i]$. For example, EncodeSF(101) = 00100110. The function EncodeSF is an encoding method with two important properties: an encoded string is uniquely decodable, and, no encoded string is a suffix of another encoded string.

We define four special colours $\alpha, \beta, Pdone, Cdone$ that are not positive integers (i.e., they cannot be confused with any node's ID, and they cannot be confused with any non-negative integer colour chosen by a node during the algorithm's execution). Practically speaking, one possible implementation is to use $\alpha = -4$, $\beta = -3$, $Pdone = -2$, and $Cdone = -1$.

Algorithm 1 provides the pseudocode for the algorithm's execution at a node. The node's two neighbours are referred to as A and B. Algorithms 2 and 3 are the subroutines that a node uses to compute its new Phase 1 colour in each round.

Algorithm 1 EarlyStopCV.

```
%% clockVal is assumed to contain the current local round number, starting at 1.
%% myID is assumed to contain the node's initial identifier.
%% Initially, Acol.P1 = Bcol.P1 = Acol.final = Bcol.final = null
if clockVal == 1 then                                              ▷ Round 1: get IDs of neighbours
    Send myID to both neighbours
    Receive ID_A from A and receive ID_B from B
else if clockVal == 2 then ▷ Phase 0: detect if local max or local min, otherwise assign parent and child
    if myID < ID_A and myID < ID_B then                            ▷ I'm a local minimum
        myPhase1Col ← α
    else if myID > ID_A and myID > ID_B then                       ▷ I'm a local maximum
        myPhase1Col ← β
    else if ID_A < myID and myID < ID_B then                       ▷ neighbourhood IDs increase towards B
        myPhase1Col ← myID
        parent ← A
        child ← B
    else                                                           ▷ neighbourhood IDs increase towards A
        myPhase1Col ← myID
        parent ← B
        child ← A
    end if
    Send myPhase1Col to both neighbours
    Receive msg_A from A and receive msg_B from B
    Acol.P1 ← msg_A
    Bcol.P1 ← msg_B
    doPhase1 ← (myPhase1Col ∉ {0, ..., 51, α, β})
else if doPhase1 == true then ▷ Phase 1: detect if one of my neighbours is settled, otherwise perform CV
    if parent == A then
        myPhase1Col ← ChooseNewPhase1Colour(myPhase1Col, Acol.P1, Bcol.P1)
    else
        myPhase1Col ← ChooseNewPhase1Colour(myPhase1Col, Bcol.P1, Acol.P1)
    end if
    Send ("P1", myPhase1Col) to both neighbours
    if received a message of the form (key, val) from A then: Acol.key ← val
    if received a message of the form (key, val) from B then: Bcol.key ← val
    doPhase1 ← (myPhase1Col ∉ {0, ..., 51, Pdone, Cdone})
else                                                               ▷ Phase 2: colour reduction down to {0, 1, 2}
    token ← clockVal mod 56
    if (myPhase1Col == token) or
       ((myPhase1Col == α) ∧ (token == 52)) or
       ((myPhase1Col == β) ∧ (token == 53)) or
       ((myPhase1Col == Pdone) ∧ (token == 54)) or
       ((myPhase1Col == Cdone) ∧ (token == 55)) then
            myFinalCol ← smallest element in {0, 1, 2} \ {Acol.final, Bcol.final}
            Send ("final", myFinalCol) to both neighbours
            terminate()
    end if
    if received a message of the form (key, val) from A then: Acol.key ← val
    if received a message of the form (key, val) from B then: Bcol.key ← val
end if
```

Algorithm 2 ChooseNewPhase1Colour(myPhase1Col, ParentPhase1Col, ChildPhase1Col).

```
1: if ParentPhase1Col ∈ {0, ..., 51} then
2:     newColour ← Pdone
3: else if ChildPhase1Col ∈ {0, ..., 51} then
4:     newColour ← Cdone
5: else if ParentPhase1Col ∈ {Pdone, Cdone, α, β} then
6:     newColour ← CVChoice(myPhase1Col, 0)
7: else
8:     newColour ← CVChoice(myPhase1Col, ParentPhase1Col)
9: end if
10: return newColour
```

Algorithm 3 CVChoice(MyCol, OtherCol).

```
1: MyString ← EncodeSF(BinaryRep(MyCol))
2: OtherString ← EncodeSF(BinaryRep(OtherCol))
3: i ← smallest x ≥ 0 such that MyString[x] ≠ OtherString[x]
4: newString ← BinaryRep(i) · myString[i]
5: return IntVal(newString)
```