Crash-Tolerant Exploration of Trees by Energy-Sharing Mobile Agents

Quentin Bramas
University of Strasbourg, ICUBE, CNRS, France

Toshimitsu Masuzawa
Graduate School of Information Science and Technology, Osaka University, Japan

Sébastien Tixeuil
Sorbonne University, CNRS, LIP6, Institut Universitaire de France, Paris, France

Abstract

We consider the problem of graph exploration by energy sharing mobile agents that are subject to crash faults. More precisely, we consider a team of two agents where at most one of them may fail unpredictably, and the considered topology is that of connected acyclic graphs (*i.e.* trees). We consider both the asynchronous and the synchronous settings, and we provide necessary and sufficient conditions about the energy.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Computing methodologies → Mobile agents

Keywords and phrases Mobile Agents, Distributed Algorithms, Energy sharing

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2024.9

Funding *Toshimitsu Masuzawa*: This work is supported in part by JSPS KAKENHI 20KK0232.
Sébastien Tixeuil: This work is supported in part by ANR SAPPORO (ANR-19-CE25-0005).

1 Introduction

Swarm robotics research has focused on the capabilities of groups of autonomous mobile robots, or agents, with limited individual abilities. These agents collaborate to achieve complex tasks such as pattern formation, object assembly, search, and exploration. Collaboration provides several benefits, including accelerated task completion, fault tolerance potential, reduced construction costs, and energy efficiency compared to larger, more complex agents. This paper investigates the collective exploration of a known edge-weighted graph by mobile agents originating from arbitrary nodes. The objective is to traverse every edge at least once, with edge weights indicating their lengths. Each agent possesses a battery with an initial energy level (that may differ among agents). An agent's battery is depleted by x when it travels a distance of x.

A recently examined collaboration mechanism among agents is energy sharing, which permits one agent to transfer energy to another upon encountering them. This ability introduces new possibilities for tasks that can be accomplished. Energy-sharing capabilities facilitate graph exploration in scenarios where it would otherwise be unattainable. Conversely, an exploration algorithm incorporating energy sharing must assign trajectories to agents to collectively explore the entire graph and schedule feasible energy transfers, making it more intricate to design. This paper also considers the possibility for an agent to crash, or cease functioning indefinitely and unpredictably. An exploration algorithm must now account for not only the feasibility of energy sharing, but also the feasibility of any plan that considers an agent crashing at any point during its prescribed algorithm.

28th International Conference on Principles of Distributed Systems (OPODIS 2024).
Editors: Silvia Bonomi, Letterio Galletta, Etienne Rivière, and Valerio Schiavoni; Article No. 9; pp. 9:1–9:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Energy-constrained agents

Chalopin et al. [5] study mobile agents with limited energy that must collaboratively deliver data from network sources to a central repository. It turns out that the decision problem is NP-hard for a single source, and they present a 2-approximation algorithm to find the minimum energy assignable to each agent to deliver data. In a follow-up paper, Chalopin et al. [6] study n mobile agents with limited energy on a straight line that must collectively deliver data from source point s to target point t, and show that deciding if agents can deliver data is (weakly) NP-complete. Bärtschi et al. [3] study delivering m messages between source-target pairs in an undirected graph using k mobile agents initially at distinct nodes. Anaya et al. [1] study the problem of broadcast and convergecast with energy aware mobile agents. In the centralized setting, they give a linear-time algorithm to compute optimal battery power and strategy for convergecast and broadcast on a line. Finding the optimal battery power for trees is NP-hard. They also give a polynomial algorithm for 2-approximation for convergecast and 4-approximation for broadcast in arbitrary graphs. Czyzowicz et al. [9] studied broadcast and exploration in trees, and give an $O(n \log n)$ time algorithm to solve the problem in a n-nodes tree T. If the number of agents is at least equal to the number of leaves of T, their approach results in an $O(n)$ time algorithm. Bärtschi et al. [2] consider the possibility of gathering k energy-constrained mobile agents in an undirected edge-weighted graph. The authors study three variants of near-gathering to minimize: *(i)* the radius of a ball containing all agents, *(ii)* the maximum distance between any two agents, or *(iii)* the average distance between agents. They prove that *(i)* is polynomial-time solvable, *(ii)* has a polynomial-time 2-approximation with matching NP-hardness lower bound, and *(iii)* admits a polynomial-time $2(1 - \frac{1}{k})$-approximation but no FPTAS unless P=NP. Czyzowicz et al. [11] studied the exit search problem for two robots on an infinite line, starting at the origin. Time-energy trade-offs for this evacuation problem are studied. Contrary to all aforementioned works, the energy consumption of a robot traveling a distance x at speed s is measured as xs^2.

1.2 Energy transfer

Energy transfer by mobile agents was previously considered by Czyzowicz et al. [8]. Agents travel and spend energy proportional to distance traversed. Some nodes have information acquired by visiting agents. Meeting agents may exchange information and energy. They consider communication problems where information held by some nodes must be communicated to other nodes or agents. They deal with data delivery and convergecast problems for a centralized scheduler with full knowledge of the instance. With energy exchange, both problems have linear-time solutions on trees. For general undirected and directed graphs, these problems are NP-complete. Then, Czyzowicz et al. [7] consider the gossiping problem in tree networks. In an edge-weighted tree network, agents spend energy while traveling and collect copies of data packets from visited nodes. They deposit copies of possessed data packets and collect copies of data packets present at the node. Czyzowicz et al. [7] prove that gossiping can be solved in $O(k^2 n^2)$ time for an n-node tree network with k agents.

Most related to our paper are the works by Czyzowicz et al. [10], Sun et al. [12], and Bramas et al. [4]. On the one hand, Czyzowicz et al. [10] study the collective exploration of a known n-node edge-weighted graph by k mobile agents with limited energy and energy transfer capability. The goal is for every edge to be traversed by at least one agent. For an n-node path, they give an $O(n + k)$ time algorithm to find an exploration strategy or report that none exists. For an n-node tree with ℓ leaves, they provide an $O(n + \ell k^2)$ algorithm to find an exploration strategy if one exists. For general graphs, deciding if exploration is possible by energy-sharing agents is NP-hard, even for 3-regular graphs. However, it's always possible to find an exploration strategy if the total energy of agents is at least twice the total weight of edges; this is asymptotically optimal. Next, Sun et al. [12] examines circulating graph exploration by energy-sharing agents on an arbitrary graph. They present the necessary and sufficient energy condition for exploration and an algorithm to find an exploration strategy if one exists. The exploration requires each node to have the same number of agents before and after. Finally, Bramas et al. [4] considered the problem of exploring every weighted edge of a given ring-shaped graph using a team of two mobile energy-sharing agents. They introduce the possibility for one of the two agents to fail unpredictably and cease functioning permanently (i.e., crashing). In this context, Bramas et al. [4] considered two scenarios: asynchronous (where no limit on the relative speed of the agents is known), and synchronous (where the two agents have synchronized clocks and operate at the same speed).

1.3 Our Contribution

We consider the problem of graph exploration by energy-sharing mobile agents that are subject to crash faults. More precisely, we consider a team of two agents where at most one of them may fail unpredictably, and the considered topology is that of connected acyclic graphs (*i.e.* trees). We assume that agents initially know the entire topology, as well as their starting locations. Under these strong assumptions, the lower-bounds we provide are interesting, as they also apply to more general case settings. The algorithms we present for the same setting demonstrate that the lower bounds are tight for path graphs, and almost tight for general trees. In the following, en_0 and en_1 denote the initial energy of the first and second agents, respectively. Also, x denotes the initial distance between the agents.

On the positive side, we show that, two asynchronous agents can explore a weighted tree T with diameter d and total weight W if (Theorem 7):

$$(en_0 \geq x) \wedge (en_1 \geq x) \wedge (en_0 + en_1 \geq 2W + 2d\lceil \log_{3/2} W \rceil + x + 2)$$

On the negative side, we provide a lower bound (Theorem 1) on the total energy for weighted star graphs (each edge has a weight $d/2$, $d \in \mathbb{N}$): $en_0 + en_1$ cannot be in $2W + d\log(o(|E|))$.

In the synchronous case, a sufficient condition (Theorem 9) is:

$$(en_0 \geq x) \wedge (en_1 \geq x) \wedge (en_0 + en_1 \geq 2W + d + x)$$

On the other hand (Theorem 2), we show that there exists an infinite family of trees such that the required total energy is at least $2W + \frac{d}{2} - 3$.

We also provide tight bounds for the case of path graphs, regardless of the initial locations of agents. The algorithms we provide when agents are not initially collocated assume that the agents may initially exchange information, and we discuss the trade-off between the amount of information exchanged and the solvability of the problem in this case. Due to space constraints, some proofs are omitted from the conference version of this paper.

2 Model

Our model is similar to that proposed by Bramas et al. [4]. We are given a weighted graph $G = (V, E)$ where V is a set of n nodes, E is a set of m edges, and each edge $e_i \in E$ is assigned a positive integer $w_i \in \mathbb{N}^+$, denoting its weight (or length). We have k mobile agents (or agents for short) $r_0, r_1, \ldots, r_{k-1}$ respectively placed at some of the nodes $s_0, s_1, \ldots s_{k-1}$ of the graph. We allow more than one agent to be located in the same place. Each agent r_i initially possesses a specific amount en_i of energy for its moves. An agent has the ability to travel along the edges of graph G in any direction. It can pause its movement if necessary and can change its direction either at a node or while traveling along an edge. The energy consumed by a moving agent is equal to the distance x it moved. An agent can move only if its energy is greater than zero. Now, the distance between two agents (that is, the minimum sum of the weights for all the paths connecting them) is the smallest amount of energy needed for them to meet at some point.

In our setting, agents can share energy with each other. When two agents, r_i and r_j, meet at a vertex or edge, r_i can take some energy from r_j. If their energy levels at meeting time are en'_i and en'_j, then r_i can take an amount of energy $0 < en \leq en'_j$ from r_j. After the transfer, their energy levels are $en'_i + en$ and $en'_j - en$, respectively.

Each agent adheres to a pre-established trajectory until encountering another agent. At this point, the agent determines if it acquires energy, and calculates its ensuing trajectory. The definition of a trajectory depends on the synchrony model:

- **In the synchronous model**, a trajectory is a sequence of pairs $((u_0, t_0), (u_1, t_1), \ldots)$, where u_i is a node, and t_i denotes the time at which the agent should reach u_i. For each $i \geq 0$, $t_i < t_{i+1}$, and u_{i+1} is either equal to u_i (i.e., the agent waits at u_i between t_i and t_{i+1}), or is adjacent to u_i (i.e., the agent leaves u_i at time t_i and arrives at u_{i+1} at time t_{i+1}). For simplicity, we assume in our algorithm that the moving speed is always one (it takes time d to travel distance d, so if $u_i \neq u_{i+1}$ and the weight of edge (u_i, u_{i+1}) is w, then $t_{i+1} - t_i = w$).
- **In the asynchronous model**, a trajectory is just a sequence of nodes $(u_0, u_1, u_2, \ldots)$, u_{i+1} being adjacent to u_i for each $i \geq 0$, and the times at which it reaches the nodes are determined by an adversary.

In other words, in the synchronous model, the agent controls its speed and its waiting time at nodes, while an adversary decides them in the asynchronous model.

The computation of the trajectory and the decision to exchange energy is based on a deterministic *localized algorithm* (that is, an algorithm executed by the agent). In a given execution, the configuration, which consists of each agent's location and remaining energy, at time t is denoted by C_t.

Localized algorithm. A *localized algorithm* f_i executed by an agent r_i at time t takes as input the *pasts* of r_i and its collocated agents, and returns *(i)* its ensuing trajectory $traj_i$ and *(ii)* the amount of energy $take_{i,j}$ taken from each collocated agent r_j. The past $Past_i(t)$ of r_i at time t corresponds to the path already traversed by r_i union the pasts of all the previously met agents. More formally:

$$Past_i(t) = \{path_i(t)\} \cup \{Past_j(t') \mid r_i \text{ met } r_j \text{ at time } t' \leq t\}$$

A set of localized algorithms is *valid* for a given initial configuration c if, for any execution starting from c, agents that are ordered to move have enough energy to do so and when an agent r_i takes energy from an agent r_j at time t, then r_j does not take energy from r_i at t.

In this paper, we consider the possibility of agent crashes. At any point in the execution, an agent r_i may crash and stop operating forever. However, if r_i has remaining energy $en'_i > 0$, then other agents meeting r_i may take energy from r_i (and are still able to read its memory without knowing it is crashed). Now, a set of localized algorithms is t-crash-tolerant if it is valid even in executions where at most t agents crash.

We are interested in solving the problem of t-crash-tolerant collaborative exploration:

***t*-crash-tolerant collaborative exploration.** Given a weighted graph $G = (V, E)$ and k mobile agents $r_0, r_1, \ldots, r_{k-1}$ together with their respective initial energies $en_0, en_1, \ldots, en_{k-1}$ and positions $s_0, s_1, \ldots, s_{k-1}$ in the graph, find a valid set of localized algorithms that explore (or cover) all edges of the graph despite the unexpected crashes of at most $t < k$ agents.

This paper focuses on the 1-crash-tolerant collaborative exploration of *trees* by *two* agents.

3 Lower Bounds

In this section, we present two lower bounds, one for the case of asynchronous unweighted stars, and one for the case of synchronous trees. The lower bound for stars is asymptotically tight even considering more general trees, as it matches the complexity of our algorithm solving the exploration in arbitrary asynchronous trees. Only the factor in front of the logarithmic term is different. In the synchronous case, the lower bound is also tight for the factor W, but the coefficient of d in from of the logarithmic term in our algorithm's complexity is larger by a factor of 2.

3.1 Asynchronous Stars

For asynchronous stars, we first show a lower bound of the total energy consumption.

▶ **Theorem 1.** *Consider a star of size $\Delta + 1$ where the weight of each edge is $d/2$. If two asynchronous agents start at the center of the star, then the total energy consumption of any algorithm cannot be in $2W + d\log(o(\Delta)) = 2W + d\log(o(W/d))$, where $W = d\Delta/2$ (i.e., the total weight).*

Proof. Assume for the purpose of contradiction that there exists an algorithm that explores any such star of degree Δ with $2W + d\log(\Delta/g(\Delta))$ total energy in the worst case, with a non-decreasing function g in $\omega(1)$ (also $g \in O(\Delta)$ as we know $2W$ is a trivial lower bound)[1].

Consider a Δ-star and assume the total amount of energy is $2W + d\log(\Delta/g(\Delta))$. If it is possible for the scheduler to ensure that the two agents, following the paths dictated by the algorithm, never meet, then each agent must have enough energy to explore the whole star, i.e., $2W - d/2$. Thus, the total amount of energy is $4W - d$, which is a contradiction. This means that the two paths chosen by the algorithm for the two agents correspond to two walks, with opposite directions, on the same cycle C covering a sub-star. The scheduler can ensure that the agents meet at a leaf node. When this happens, then $d|C|/2$ energy has been spent, where $|C|$ is the number of edges in C. The agents have explored a star of degree at most $|C|/2$. Since, before they start, they must each have enough energy to explore the entire cycle, it follows that $d|C|/2 \leq W + d\log(\Delta/g(\Delta))/2$. Before using the complexity of the algorithm recursively on the remaining star, we can ensure that the degree of the star that the agents have explored is close to the upper bound (which is in fact the best case for the agents). This can be done by activating one agent until the degree Δ' of the star explored by the agent is (with $1 > a \geq 0$)

$$\Delta' = \frac{W + d\log(\Delta/g(\Delta))/2}{d} - a = \frac{\Delta + \log(\Delta/g(\Delta))}{2} - a \quad \Rightarrow \Delta - \Delta' = \frac{\Delta - \log(\Delta/g(\Delta))}{2} + a \tag{1}$$

[1] It is equivalent to say that there exists f in $O(\log(\Delta))$ and in $\omega(1)$ such that the algorithm uses $2\Delta + 2\log(\Delta) - f(\Delta)$, and then define $g(x) = 2^{f(x)}$

The remaining unexplored edges form a $(\Delta - \Delta')$-star. The agents must first move towards this new star, which uses $2 \times d/2 = d$ energy. So the total remaining energy is

$$R = 2W + d\log(\Delta/g(\Delta) - d\Delta' - d = W + \frac{d\log(\Delta/g(\Delta)}{2} + d(a-1)$$

Then, by using the same algorithm, this star is explored, in the worst case using

$$d(\Delta - \Delta') + d\log\left(\frac{\Delta - \Delta'}{g\left(\Delta - \Delta'\right)}\right)$$

energy. We show that there exists a Δ large enough so that the remaining energy is not enough to explore the remaining star in the worst case.

Replace $\Delta - \Delta'$ by its computed value (thanks to Equation (1)) and we obtain that in the worst case, the energy required to explore the remaining star is at least (for Δ large enough)

$$\begin{aligned} &d(\frac{\Delta - \log(\Delta/g(\Delta))}{2} + a) + d\log(\frac{\Delta - \log(\Delta/g(\Delta)) + 2a}{2g(\frac{\Delta - \log(\Delta/g(\Delta) + 2a)}{2})}) \\ &\geq d(\frac{\Delta - \log(\Delta/g(\Delta))}{2} + a) + d\log\left(\frac{\Delta - \log(\Delta)}{2g\left(\Delta/2\right)}\right) \end{aligned}$$

where the a in the denominator can be removed because it is smaller than $\log(\Delta/g(\Delta))$ when Δ is large enough. Hence, the previous value is equal to

$$\begin{aligned} &= W + \frac{d\log(\Delta/g(\Delta))}{2} + d(a-1) - d\log(\Delta/g(\Delta)) + d + d\log\left(\frac{\Delta - \log(\Delta)}{g\left(\Delta/2\right)}\right) + d\log\left(\frac{1}{2}\right) \\ &= W + \frac{d\log(\Delta/g(\Delta))}{2} + d(a-1) + d\log\left(\frac{g(\Delta)}{\Delta}\frac{\Delta - \log(\Delta)}{g\left(\Delta/2\right)}\right) \\ &= R + d\log\left(\frac{g(\Delta)}{g\left(\Delta/2\right)}\left(1 - \frac{\log(\Delta)}{\Delta}\right)\right) \end{aligned}$$

To show the contradiction, it is sufficient to show that, for a value Δ large enough, the factor inside the logarithm is strictly greater than 1. Indeed, this implies that the amount of energy required in the worst case is strictly more than R, a contradiction. Assume that

$$g(\Delta) \leq g\left(\Delta/2\right)\frac{1}{\left(1 - \frac{\log(\Delta)}{\Delta}\right)}$$

for all Δ, in particular for 2^i for all $i \geq 0$. Then,

$$g(2^k) \leq g\left(1\right)\prod_{i=1}^{k}\frac{1}{\left(1 - \frac{i}{2^i}\right)}$$

This is bounded as the product series converges, contradicting that g is in $\omega(1)$. ◀

3.2 Synchronous Trees

We now present a lower bound on the total energy for exploring a weighted tree with the total weight W and a diameter d.

▶ **Theorem 2.** *There exists an infinite family of trees such that the required total energy by two synchronous agents is at least* $2W + \frac{d}{2} - 4$

Proof. In the proof, we assume trees are unweighted, that is, $w(e) = 1$ for each edge e. The theorem for weighted trees is obtained by setting the weight of each edge to some appropriate value w so that the total cost becomes W.

Let $d \geq 5$ and consider the sequence $m_5, m_6, \ldots, m_d$ such that $m_i = 3^{i+1} - 2$. Let T_d be the following tree of diameter d:

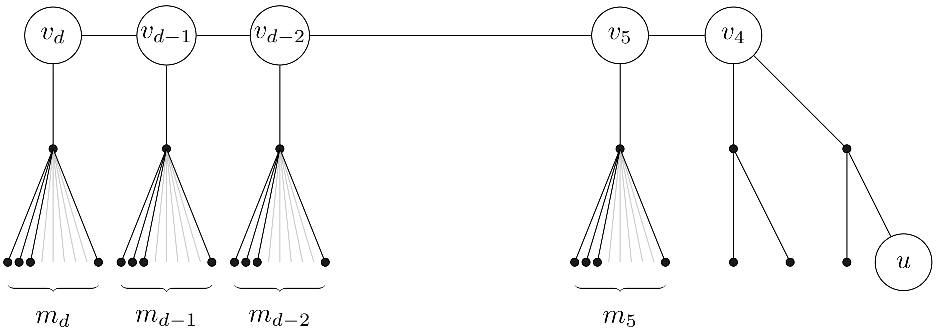

Figure 1 Illustration of the tree T_d of diameter d. $m_i = 3^{i+1} - 2$, so that T_d has $6 + \frac{9}{2}(3^d - 81)$ nodes.

By convention, let T_4 be the perfect binary tree of height 2 (visible in the figure as the bottom-sub-tree rooted at v_4).

Assume both agents are initially located at v_d. We show by induction on d that exploring T_d requires a total of $2|E| + d - 4$, where

$$|E| = 6 + \sum_{i=5}(m_i + 2) = 6 + \frac{9}{2}(3^d - 81).$$

If $d = 4$, then one can see that if an agent crashes at a leaf, then the required total energy is $2|E| + 4 - 4$. Indeed, if an agent crashes at a leaf, the energy to reach the leaf where the crashed agent is 2, which has been consumed twice (by the crashed and by the non-crashed agent). Then, all the other edges are explored twice except for the path to the last leaf (of length 2), so in total it is $2 + 2|E| - 2 = |E|$.

Now assume that the result is true for any d' such that $4 \leq d' < d$, then we prove the result for $d > 4$.

We consider two cases, depending on whether u is visited before at least one of the m_d nodes below v_d, or if the m_d nodes below v_d are all visited before u (or at the same time as).

If the $m_d = 3^{d+1} - 2$ nodes below v_d are explored before node u (or at the same time), then, if the total energy is at most $2|E| + d - 4$, we can show that the first bottom-edge of v_d has been traversed by the two agents (hence 4 times). Indeed, one agent cannot explore alone the bottom-sub-tree of v_d since it requires $2m_d + 2$ energy to be visited (and come back to v_d) and the other agent needs to keep $2m_d$ energy in case the first one crashes. So it requires at least $4m_d + 2 = 4 \times (3^{d+1} - 2) + 2 = 4 \times 3^{d+1} - 6$ but the total amount of energy available is at most $2|E| + d - 4 = 2(9(3^d - 81)/2 + 6) + d - 4 = 3^{d+2} - 721 + d$, which is less than $4 \times 3^{d+1} - 6$ when $d \geq 0$. So the exploration of the bottom-sub-tree of v_d and of the edge $v_d v_{d-1}$ costs at least $2m_d + 4 + 2$. The remaining tree is exactly T_{d-1}, rooted at v_{d-1}, and we know by induction that the exploration of T_{d-1} requires $2(|E| - m_d - 2) + (d-1)/2 - 4$. The energy consumed to explore the tree T_d is at least the sum of the energy consumed to explore each sub-tree so in total, the energy consumed is at least

$$2(|E| - m_d - 2) + (d - 1)/2 - 4 + 2m_d + 4 + 2 = 2|E| + d/2 - 5/2$$

Now consider the case where u is visited first, say by r_1. If the agents remain together, then the path $v_d u$ is traversed 4 times. Otherwise, when the two agents split, r_0 is left with EN_0 energy and r_1 with EN_1. Consider for simplicity that split before an agent travels edge $v_d v_{d-1}$ (if the agents travel together a path $v_d v_{d'}$ then in the end this path is traversed four times, and the remaining result holds).

Before the agents meet again, r_1 is exploring a subtree T' containing u with at most $EN_1/2$ edges, and we know that $EN_1 \geq d-2$ because T' contains u. Let d_1 be the smallest index such that the tree below v_{d_1} contains a leaf that is not in T'. If r_1 does not crash, it has to meet with r_0 again, before all the leaves below v_{d_1} are explored, so an agent has to traverse the path $v_d v_{d_1}$ again to explore the unexplored leaves below v_{d_1}. So in the end, the path $v_d v_{d_1}$, of length $d - d_1$, is traversed at least three times. In total, at least $2|E| + d - d_1$ energy is consumed when the exploration is complete.

Now consider that r_1 crashes at u. When r_0 realizes that r_1 is crashed, it can explore some edges and it then eventually reaches u (it has to do so, at least to confirm the exploration of the tree). Let d_0 be the largest index such that the sub-tree below v_{d_0} contains a leaf that is not explored by r_0 before r_0 reaches u. After visiting u, r_0 has to visit a leaf below v_{d_0} so the path $v_{d_0}u$, of length $d_0 - 2$, is traversed at least three times. So in total, at least $2|E| - 2 + d_0 - 2$ energy is consumed when the exploration is complete. So in the worst case, in total, at least $E_{\max} = \max(2|E| + d_0 - 4, 2|E| + d - d_1)$ energy is consumed.

Assume $EN_0 + EN_1 \leq 2|E| + d/2 - 4$ (otherwise the theorem is proved). We now prove that $d_0 \geq d_1$. Indeed, if we assume for the sake of contradiction that $d_0 < d_1$, it means that, when r_1 crashes while exploring a subtree that includes u, and when r_0 realizes that r_1 is crashed, then r_0 visits all the subtrees below v_i, $i \geq d_0$, and then has to be able to visit all the subtrees below v_i, $i \leq d_1$ since r_1 can be crashed in any of these nodes. If $d_0 < d_1$, this means that r_0 must be able to visit the entire T_d tree, so $EN_0 \geq 2|E| - 2$. Since $EN_1 \geq d-2$, we obtain $EN_0 + EN_1 > 2|E| + d/2 - 4$, a contradiction.

Hence we have $d_0 \geq d_1$, so either $d_1 \geq d/2$, which implies $d_0 - 4 \geq d/2 - 4$ and $E_{\max} \geq 2|E| + d/2 - 4$, or $d_1 < d/2$, which implies $d - d_1 > d/2$ and $E_{\max} \geq 2|E| + d/2$. Hence the Theorem is proved. ◀

4 Crash-tolerant algorithms for two energy-sharing agents in trees

Our algorithms are presented as a set of rules. Each rule is composed of a condition (that must be true to execute the rule action) and an action (the rest of the rule, that is executed when the condition is satisfied). Each action can be:

- A move in a prescribed direction toward a node or an agent. For example, in the line topology, "$(v_0) \leftarrow$ " prescribes the agent to go left until node v_0 is reached, while "$\rightarrow (r_1)$" prescribes the agent to go right until agent r_1 is met.
- A sequence of actions, two consecutive actions being separated by a semicolon ";". Sometimes, a sequence is given a name for brevity. For example, *EulerianExplore*(T) (resp. *ReverseEulerianExplore*(T)) performs a clockwise Eulerian (resp. a counter clockwise Eulerian) tour of tree T.
- An alternation of actions, depending on a Boolean condition. For example, "**if** c **then** a_1 **else** a_2" prescribes that action a_1 should be executed if condition c is satisfied, while action a_2 should be executed otherwise. The condition can be related to the topology (e.g., the agent is closer to a point p_1 than a point p_2) or be related to a collocated agent and its past (e.g., the collocated agent is coming from a given edge). In the line, we consider only three conditions: *(i)* $\overleftarrow{r}$ returns true if agent r was going left, and false otherwise; *(ii)* $\overrightarrow{r}$ returns true if agent r was going right, and false otherwise; and *(iii)* $p_1 \prec p_2$ returns true if p_1 is closer to the observing agent than p_2, and false otherwise.

- When agent are synchronous, a waiting instruction, that is stopped either if the other agent is met, or after a given number of time instants. For example, "**wait** t *time units for* r_1 **then** a_1 **timeout** a_2" prescribes the agent to wait at most t time instants for agent r_1 to meet at the current position. If the agent meets r_1, then execute a_1, otherwise execute a_2.

Before presenting our algorithms for trees, we show two general Lemmas that hold regardless of the topology, and give necessary conditions about the initial energy of the agents. The third Lemma presents two general properties of crash-tolerant exploration algorithms on trees.

▶ **Lemma 3.** *Let $G = (V, E)$ be a connected graph to be explored by two agents r_0 and r_1. If the initial distance between r_0 and r_1 is d_{init}, then $en_0 \geq d_{\text{init}}$ and $en_1 \geq d_{\text{init}}$ are necessary for exploring G.*

▶ **Lemma 4.** *Let $G = (V, E)$ be a connected graph to be explored by two agents r_0 and r_1. If the total weight of G is W, then $en_0 + en_1 \geq W$ is necessary for exploring G.*

▶ **Lemma 5.** *In any crash-tolerant exploration algorithm $\mathcal{A}$ by two agents on a tree T, the following holds:*

1. *Each agent has to eventually move unless it confirms the completion of exploration.*
2. *Each agent r confirms the completion of exploration only when, for each leaf u, u is visited by r or r meets the other agent that already visited leaf u.*

4.1 Asynchronous trees

We now consider the case of two asynchronous agents in trees. Let $T = (V, E)$ be a weighted tree of n nodes, and let d be the weighted diameter of T. First, we construct a family of k connected non-empty subtrees of T named $T_1, T_2, \ldots, T_k$, where $T_i = (V_i, E_i)$ and $(E_i)_{1 \leq i \leq k}$ forms a partition of E. At the beginning, the agents meet to share energy. Then the agents repeat a procedure $explore(T_i)$ for all $i \in \{1, \ldots, k\}$ from 1 to k. The procedure assumes that the agents are initially at the same location (possibly on an edge), and ensure that after execution the agents are at the same location (not necessarily the same as the initial one) if $i < k$ (when $i = k$ the agents can terminate anywhere on completion of the exploration).

Agent r_0 (resp. r_1) executing $explore(T_i)$ first moves to the closest node v_i of T_i, executes $EulerianExplore(T_i)$ (resp. $ReverseEulerianExplore(T_i)$), and moves back to its initial location, until it meets the other agent, and T_i is explored. If the agents meet before ending this sequence of moves and E_i is explored, then the procedure terminates. This occurs during the exploration of the Eulerian tour from v_i, or when one of the agents r comes back from v_i to its initial location after completing its Eulerian tour while the other has not started it (it is still moving towards v_i from the location where it started executing $explore(T_i)$).

Since the length of the Eulerian tour is $2w(T_i)$ (where $w(T_i)$ denotes the weight of T_i) and the distance to v_i from their initial location is d in the worst case, each agent must have, at the beginning of the procedure, the energy of at least $2d + 2w(T_i)$ if $i < k$ (to terminate even when the other agent remains at the initial location), at least $d + 2w(T_i)$ if $i = k$. When the procedure terminates the total energy consumed during the procedure is at most $2d + 2w(T_i)$ (because every edge traversed in the procedure is traversed exactly twice if $i < k$, and at most twice if $i = k$).

Consequently, to complete all $explore(T_i)$, for every i, sequentially, our algorithm requires that the total remaining energy EN_i at the beginning of the procedure $explore(T_i)$ is as follows, where x is the initial distance between the agents:

- $EN_k \geq 2d + 4w(T_k)$
- $EN_i \geq \max(2d + 2w(T_i) + EN_{i+1}, 4d + 4w(T_i))$ $(2 \leq i \leq k-1)$
- $EN_1 \geq x + \max(2d + 2w(T_1) + EN_2, 4d + 4w(T_1))$ where x is the initial weighted distance between the agents.

Moreover, the total energy consumption for exploring T is at most $x + \sum_{i=1..k}(2d + 2w(T_i)) = 2W + 2kd + x$.

We now have to construct the partition $T_1, T_2, \ldots, T_k$ of T so that k should be small to reduce the number of calls to $explore()$, but each $w(T_i)$ should not be too large to avoid increasing the energy required at the beginning of $explore(T_i)$. A good partition could be to have $w(T_k) = 1$ and $w(T_i) = 2w(T_{i+1})$, which results in $k = \lceil \log W \rceil$. In general trees, such a partition does not exist, but we can obtain a similar result using the centroid-based partition recursively.

Let $T = (V, E)$ be a weighted tree with total weight W. The centroid of T is defined as follows. In the following, for a tree T and a node u of T, T can be regarded as a rooted tree, denoted by T^u, rooted at u. For the root u and its neighbor v, let T^u_v be the subtree of T^u rooted at v.

1. When there exists an edge $(u, v) \in E$ satisfying $w(T^u_v) < W/2$ and $w(T^v_u) < W/2$, the centroid of T is the point p on edge (u, v) such that $w(T^u_v) + w(v, p) = w(T^v_u) + w(u, p) = W/2$. We call p the *edge centroid*.
2. When there exists a node $u \in V$ satisfying $w(T^u_v) + w(u, v) \leq W/2$ for each neighbor v of u, the centroid of T is node u. We call u the *node centroid*.

▶ **Lemma 6.** *Any weighted tree $T = (V, E)$ has a unique centroid.*

We now construct a new tree $T' = (V', E')$ from T by inserting nodes onto edges when necessary to simplify the construction of the partition. Observe that exploring the edges of T' is equivalent to exploring the edges of T, as exploring the two edges obtained after adding a node is equivalent to exploring the initial edge.

To construct $T' = (V', E')$ and obtain $T_1, T_2, \ldots, T_k$ of subgraphs of T', we recursively use the centroid-based partition of a tree. Let $T' = T$ temporally and c be the centroid of T'. If c is an edge centroid on an edge e, T' is updated by inserting a node at c to partition e into two edges. We denote the inserted node as c. So now, c is the node centroid of T'. Consider the subtrees T'^c_u for each neighbor u of c.

We can show that there exists a subset $N'(c) \subset N(c)$ of neighbors of c such that $W/3 \leq \sum_{u \in N'(c)}(w(T^c_u) + w(c, u)) \leq W/2$. Let T_1 be the connected subtree of T' consisting of nodes $\cup_{u \in N'(c)} V(T^c_u) \cup \{c\}$ and edges among them, where $V(T^c_u)$ denotes the set of nodes in T^c_u. Hence T_1 satisfies

$$W/3 \leq w(T_1) \leq W/2 \tag{2}$$

Temporary tree T' is further updated (if necessary) and T_2 is obtained by applying the same method to the remaining tree consisting of the node set $V' \setminus \cup_{u \in N'(c)} V(T^c_u)$. Trees $T_3, T_4, \ldots$ are obtained by recursively applying the same method until the weight of the remaining tree becomes one or smaller, where the last subtree T_k is the remaining tree. Each time the method is applied, the weight of the remaining tree is reduced by at least $2/3$, which implies the method is applied at most $k = \lceil \log_{3/2} W \rceil$ times. Thus, the initial amount of the total energy should be at least $2W + 2d\lceil \log_{3/2} W \rceil + x$. Actually, we can derive the following sufficient condition on the initial amount of the energy.

ct_1 : $(en_0 \geq x) \wedge (en_1 \geq x) \wedge (en_0 + en_1 \geq 2W + 2d\lceil \log_{3/2} W \rceil + x)$

The following shows the actions of the agents. It is assumed that subtrees $T_1, T_2, \ldots T_k$ ($k \leq \lceil \log_{3/2} W \rceil$) are a priori determined by the recursive centroid-based partitions.

Step 0: The agents meet on the shortest path between their initial locations. (This is executed only once at the beginning of the execution.) Set $i = 1$.

Step 1: When they meet at a point, say p (possibly on an edge), they evenly share the remaining energy.

Step 2 Agent r_0 (resp. r_1) performs the following sequence of moves: move to the nearest node v_i of T_i; traverse T_i along a Eulerian tour of T_i in the clockwise direction (resp. the counter-clockwise direction) until the agent *(i)* meets the other, or *(ii)* completes the Eulerian tour traversal without meeting the other; move toward p until the agent meets the other in case of *ii)* if $i < k$; If $i < k$, set p be the meeting point, set $i = i + 1$, and continue from **Step 1**.

▶ **Theorem 7.** *If condition ct_1 holds, then if at most one agent crashes, two asynchronous agents executing the localized algorithms prescribed above explore the entire tree.*

Notice that in unweighted stars of degree Δ, hence with $W = \Delta$, we have $k = \lceil \log \Delta \rceil$ holds for the number of the subtrees $T_1, T_2, \ldots, T_k$ and the closest node v_i of T_i is the center node of the star graph for each i ($1 \leq i \leq k$). Consequently, the sufficient condition is refined and becomes tight.

▶ **Corollary 8.** *The prescribed algorithm explores the entire unweighted star of degree Δ when the following condition is satisfied:*

$$(en_0 \geq x) \wedge (en_1 \geq x) \wedge (en_0 + en_1 \geq 2\Delta + 2\lceil \log \Delta \rceil + x)$$

4.2 Synchronous trees

We now present an upper bound for synchronous tree exploration. Our proof is constructive, as we present an algorithm to solve the problem. Our strategy for exploring T is as follows. First, the two agents meet at the initial location v_0 of agent r_0, then they execute $SyncTreeExplore(T)$ that orders one agent to explore all the subtrees $T_1, \ldots, T_{k-1}$, rooted at the children of v_0, except T_k with the largest weight. When only T_k is unexplored, the agents both move towards its root and execute recursively $SyncTreeExplore(T_k)$. If the exploring agent crashes, the other one can reach it to retrieve the remaining energy and explore the remaining edges on its own. The pseudo-code of *SyncTreeExplore* is given in Algorithm 1 and is illustrated by Figure 2.

We now give a more formal description of the algorithm. First, the two agents meet at the node v_0 where agent r_0 is initially located. If x is the initial distance between the two agents, r_0 waits for r_1 until the time x, the time r_1 should take to reach v_0 without crashing. If r_1 does not reach v_0 by the time, r_0 moves toward the initial location of r_1 until it meets r_1. When the agents are collocated, a total of x energy is consumed. If r_1 is crashed, r_0 retrieves all the energy from r_1 and explores the entire tree using at most $2W$ energy.

If both agents are in v_0, then the agents execute *SyncTreeExplore* as follows. Consider T as a tree rooted at v_0 and let T_i ($1 \leq i \leq k$) be the subtree rooted at a child v_i of v_0, and let $w_i = w(T_i) + w(v_0, v_i)$. Without loss of generality, assume that w_k is the largest among all w_i ($1 \leq i \leq k$). Then, we explain how to explore $T_1, T_2, \ldots, T_{k-1}$ (subtrees except for T_k) one by one, followed by the exploration of T_k in a recursive way.

Algorithm 1 *SyncTreeExplore*(T), executed by collocated agents.

```
r_0 and r_1 evenly share their energy ;
Let T_1, ..., T_k be the subtrees rooted at the children of the current node ;
for i = 1, ..., k − 1 do
    r_0 executes EulerianExplore(T_i) in time 2w_i;
    if r_0 is crashed then
        r_1 executes ReverseEulerianExplore(T_i) until it meets r_0 and T_i is explored, takes all the
          energy from r_0, then explores the remaining unexplored edges on its own ;
        return
    end
    r_0 and r_1 share their energy ;
end
Move to the root of T_k ;
if r_i is crashed, i ∈ {0, 1} then
    r_{1−i} takes all the energy from r_i, then explores the remaining unexplored edges on its own;
    return
end
Execute SyncTreeExplore(T_k) ;
```

To explore T_i $(1 \leq i \leq k-1)$, the agents first evenly share energy so that each has the energy of at least $2w_i$, which is sufficient to solely complete the exploration of T_i and come back to v_0. Agent r_0 moves to the root of T_i, traverses T_i along an Eulerian tour, and then comes back to v_0. While r_0 explores T_i, r_1 is waiting at v_0. If r_0 does not come back to v_0 in time $2w_i$ after leaving v_0 for T_i (which implies r_0 crashes during the traversal), r_1 explores T_i along the same Eulerian tour of T_i but in the opposite direction until it meets r_0 and T_i is completely explored. When r_1 meets r_0, it gets all the remaining energy from r_0, comes back at v_0 and explores all the remaining part $T_{i+1}, T_{i+2}, \ldots, T_k$ by itself. The energy consumption for exploring T_i is $2w_i$ with additionally at most d (the diameter) if r_0 crashes during the exploration (but this can occur at most once). If r_0 does not crash and completes the exploration of T_i, then the agents share their energy and r_0 explores T_{i+1} in a similar way. If agent r_0 explores $T_1, T_2, \ldots, T_{k-1}$, then the agents evenly share the remaining energy, move to the root of T_k, and explore T_k using the same algorithm *SyncTreeExplore*(T_k).

The remarkable point is that the edge between v_0 and the root of T_k is traversed by the two agents together but is never traversed afterward. Hence each edge is traversed exactly twice if no crash occurs. If a crash occurs, each edge is traversed at most twice, except for a path of length at most d that is traversed three times. Thus, the total energy consumption for exploring T is at most $2W + d + x$; x moving to v_0 and $2W + d$ for exploring the tree.

The condition for the above method is as follows.

ct_2 : $(en_0 \geq x) \wedge (en_1 \geq x) \wedge (en_0 + en_1 \geq 2W + d + x)$

▶ **Theorem 9.** *If condition ct_2 holds, then if at most one agent crashes, two synchronous agents executing the localized algorithms prescribed above explore the entire unweighted tree.*

5 Optimal algorithms in lines

5.1 Asynchronous Lines

Let $G = (V, E)$ be a path graph (called line thereafter) such that $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and $E = \{e_i = (v_i, v_{i+1}) \mid 0 \leq i < n-1\}$. We consider that v_i is on the left of v_{i+1} and says, for example, "an agent moves left". Let w_i be the weight of edge e_i. For each node v_i, let $dis(v_i)$ be the weighted distance from v_0 to v_i, that is, $dis(v_i) = \sum_{j=0..i-1} w_j$, and let ℓ be the total weight of G, that is, $\ell = dis(v_{n-1})$. Consider that agents r_0 and r_1 are initially located at

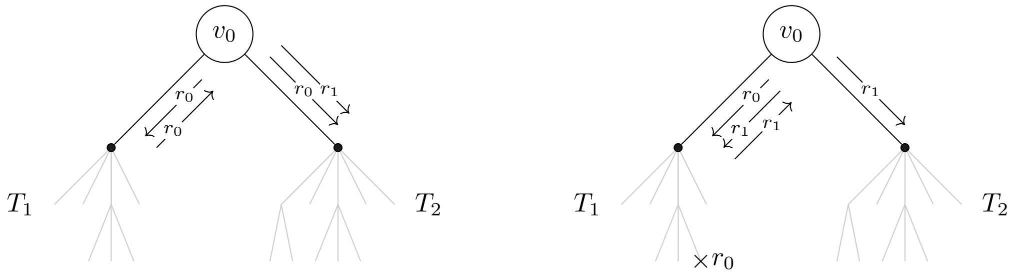

Figure 2 (left) Exploration of T_1 while r_1 is waiting at v_0, then r_0 and r_1 move to the root of T_2 to execute the same algorithm recursively. (right) If r_0 crashes during the exploration of T_1, r_1 explore T_1 using the same Eulerian path but in the opposite direction until it reaches r_0, then it moves to v_0 and explores T_2 on its own.

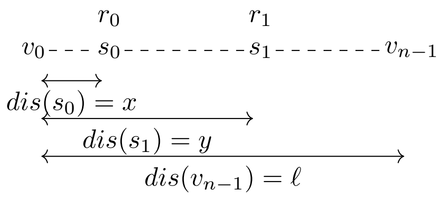

Figure 3 A line of n nodes with two agents r_0 and r_1 hosted by nodes s_0 and s_1, respectively.

nodes s_0 and s_1 respectively. For convenience, we assign $x = dis(s_0)$ and $y = dis(s_1)$, and assume without loss of generality that $x \leq y$ and $x \leq \ell - y$ [2]. Figure 3 illustrates these notations.

Let en_0 and en_1 be the initial energy of agents r_0 and r_1 respectively. The four conditions for the rules are:

c_1 : $(en_0 \geq x + y) \wedge (en_1 \geq y) \wedge (en_0 + en_1 \geq 2\ell + x + y)$

c_2 : $(en_0 \geq \ell - x) \wedge (en_1 \geq 2\ell - (x + y)) \wedge (en_0 + en_1 \geq 4\ell - (x + y))$

c_3 : $(en_0 \geq \ell + x) \wedge (en_1 \geq 2\ell - y)$

c_4 : $(en_0 \geq y - x) \wedge (en_1 \geq y - x) \wedge (en_0 + en_1 \geq min(3\ell + y - x, 2\ell - x + 3y))$

The corresponding actions are denoted by a_i, $i \in \{1, \ldots, 4\}$.

a_1 :

r_0 : $(v_0) \leftarrow; \rightarrow (v_{n-1})$

r_1 : $(r_0) \leftarrow$; **if** $\overleftarrow{r_0}$ **then** $(v_0) \leftarrow$; $\rightarrow (v_{n-1})$ **else** $\rightarrow (v_{n-1})$

a_2 :

r_0 : $\rightarrow (r_1)$; **if** $\overrightarrow{r_1}$ **then** $\rightarrow (v_{n-1})$; $(v_0) \leftarrow$ **else** $(v_0) \leftarrow$

r_1 : $\rightarrow (v_{n-1}); (v_0) \leftarrow$

a_3 :

r_0 : $(v_0) \leftarrow; \rightarrow (v_{n-1});$

r_1 : $\rightarrow (v_{n-1}); (v_0) \leftarrow$

a_4 :

r_0 : $\rightarrow (r_1)$; **if** $v_0 \prec v_{n-1}$ **then** $(v_0) \leftarrow$; $\rightarrow (v_{n-1})$ **else** $\rightarrow (v_{n-1})$; $(v_0) \leftarrow$

r_1 : $(r_0) \leftarrow$; **if** $v_0 \prec v_{n-1}$ **then** $(v_0) \leftarrow$; $\rightarrow (v_{n-1})$ **else** $\rightarrow (v_{n-1})$; $(v_0) \leftarrow$

[2] If $x > \ell - y$, one can reverse the order of the nodes on the line, exchange the positions of r_0 and r_1, and the condition is satisfied in the new graph.

In the above algorithms, when two agents meet, they share energy equally. In more detail, if the energy levels when meeting are en'_i and en'_j with $en'_i < en'_j$, then r_i takes an amount of energy $(en'_j - en'_i)/2$ from r_j. After the transfer, their energy levels are both $(en'_i + en'_j)/2$.

▶ **Lemma 10.** *For $i \in \{1, \ldots, 4\}$, if c_i is satisfied and a_i is executed, then asynchronous exploration completes if at most one agent crashes.*

In the following, we show that satisfying at least one of the conditions c_1, c_2, c_3 or c_4 is necessary for two asynchronous agents to complete exploration of lines. We first state some general facts about asynchronous line exploration in the following Lemma.

▶ **Lemma 11.** *Any line exploration algorithm for two asynchronous agents satisfies:*

1. *Consider the configuration that two agents meet. Suppose v_0 (resp. v_{n-1}) is unvisited although v_{n-1} (resp. v_0) is already visited by either of the agents, each agent has to have enough energy to visit v_0 (resp. v_{n-1}).*
2. *Consider the configuration that two agents meet and both v_0 and v_{n-1} are unvisited by either of the agents. Then each agent has to have enough energy to visit v_0 and v_{n-1} (the amount of energy required is ℓ plus the distance to the closest extremity).*

▶ **Lemma 12.** *If for every $i \in \{1, \ldots, 4\}$, c_i is not satisfied, then asynchronous exploration assuming at most one agent crashes is impossible.*

For two agents without energy sharing, the following lemma shows the amounts of the initial energy of the agents necessary and sufficient for exploration of lines.

▶ **Lemma 13.** *If r_0 and r_1 don't share energy, then exploring the line assuming at most one crash is possible if and only if $en_0 \geq x + \ell$ and $en_1 \geq min\{x, \ell - y\} + \ell$ hold.*

5.2 Synchronous Lines

In this subsection, we consider synchronous lines. We assume that two agents start execution at the same time, and it takes time d for each agent to travel distance d. A remarkable feature of synchronous lines is that an agent can detect the crash of the other when it does not show up as scheduled. This feature enables energy saving with respect to the asynchronous case. We consider four possible conditions that enable exploration:

c_1 : $(en_0 \geq x + y) \wedge (en_1 \geq y) \wedge (en_0 + en_1 \geq max(\ell + x + y, 2\ell + x - y))$
c_2 : $(en_0 \geq \ell - x) \wedge (en_1 \geq 2\ell - (x + y)) \wedge (en_0 + en_1 \geq 3\ell - x - y)$
c_3 : $(en_0 \geq \ell + x) \wedge (en_1 \geq 2\ell - y)$
c_4 : $(en_0 \geq y - x) \wedge (en_1 \geq y - x) \wedge (en_0 + en_1 \geq 2\ell - x + y)$
The corresponding actions are denoted by a_i, $i \in \{1, \ldots, 4\}$ in Figure 4.

In the synchronous setting, when two agents meet, each agent can detect whether the other agent is crashed or not (the other agent is considered crashed if a timeout occurs). Upon meeting, if the other agent is not crashed (yet), the agents share energy so that both have the same amount. Otherwise, if an agent r crashes, then the other agent takes all the remaining energy from r.

As for the asynchronous line, we now prove that the four conditions are necessary and sufficient for the synchronous line in the two following Lemmas.

▶ **Lemma 14.** *For $i \in \{1, \ldots, 4\}$, if c_i is satisfied and a_i is executed, then synchronous exploration completes if at most one agent crashes.*

▶ **Lemma 15.** *If for every $i \in \{1, \ldots, 4\}$, c_i is not satisfied, then synchronous exploration assuming at most one agent crashes is impossible.*

Action a_1

r_0 :	$(v_0) \leftarrow; \rightarrow (v_{n-1})$
r_1 :	**wait** $x+y$ *time units for* r_0 **then** | $\rightarrow (v_{n-1})$ **timeout** | $(r_0) \leftarrow$; | **if** $\overleftarrow{r_0}$ **then** | | $(v_0) \leftarrow$; $\rightarrow (v_{n-1})$ | **else** $\rightarrow (v_{n-1})$; **end**

Action a_2

r_0 :	**wait** $2\ell-(x+y)$ *time units for* r_1 **then** | $(v_0) \leftarrow$ **timeout** | $\rightarrow (r_1)$; | **if** $\overrightarrow{r_1}$ **then** | | $\rightarrow (v_{n-1})$; $(v_0) \leftarrow$ | **else** $(v_0) \leftarrow$; **end**
r_1 :	$\rightarrow (v_{n-1})$; $(v_0) \leftarrow$

Action a_3

r_0 :	$(v_0) \leftarrow; \rightarrow (v_{n-1})$;
r_1 :	$\rightarrow (v_{n-1})$; $(v_0) \leftarrow$

Action a_4

r_0 :	**wait** $y-x$ *time units for* r_1 **then** | **wait** $2x$ *time units for* r_1 **then** | | $\rightarrow (v_{n-1})$ | **timeout** | | $(r_1) \leftarrow$; | | **if** $\overrightarrow{r_1}$ **then** $\rightarrow (v_{n-1})$ | | **else** $(v_0) \leftarrow$; $\rightarrow (v_{n-1})$ | **end** **timeout** | $\rightarrow (r_1)$; | **if** $v_0 \prec v_{n-1}$ **then** | | $(v_0) \leftarrow$; $\rightarrow (v_{n-1})$; | **else** $\rightarrow (v_{n-1})$; $(v_0) \leftarrow$; **end**
r_1 :	$(v_0) \leftarrow; \rightarrow (v_{n-1})$

Figure 4 Actions for synchronous line exploration.

6 Conclusion

We characterized the solvability of exploration with two crash-prone energy-sharing mobile agents in the case of tree topologies, both in the synchronous and in the asynchronous settings. Obvious open questions include further closing the gap between necessary and sufficient conditions for the initial amounts of energy in the case of trees that are not reduced to a line, solving the problem with more than two agents, and considering other topologies, such as grid, tori, and general graphs.

Also, our model for energy transfer is very simple (all energy can be transferred instantaneously between two agents, at no cost). It would be interesting to study non-linear battery models (where the capacity decreases faster if more instantaneous current is drawn, and the capacity increases less if faster charge is executed) in this context.

References

1 Julian Anaya, Jérémie Chalopin, Jurek Czyzowicz, Arnaud Labourel, Andrzej Pelc, and Yann Vaxès. Convergecast and broadcast by power-aware mobile agents. *Algorithmica*, 74(1):117–155, 2016. doi:10.1007/s00453-014-9939-8.

2 Andreas Bärtschi, Evangelos Bampas, Jérémie Chalopin, Shantanu Das, Christina Karousatou, and Matús Mihalák. Near-gathering of energy-constrained mobile agents. *Theor. Comput. Sci.*, 849:35–46, 2021. doi:10.1016/j.tcs.2020.10.008.

3 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, and Paolo Penna. Energy-efficient delivery by heterogeneous mobile agents. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 10:1–10:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.10.

4 Quentin Bramas, Toshimitsu Masuzawa, and Sébastien Tixeuil. Brief announcement: Crash-tolerant exploration by energy sharing mobile agents. In Shlomi Dolev and Baruch Schieber, editors, *Stabilization, Safety, and Security of Distributed Systems – 25th International Symposium, SSS 2023, Jersey City, NJ, USA, October 2-4, 2023, Proceedings*, volume 14310 of *Lecture Notes in Computer Science*, pages 380–384. Springer, 2023. doi:10.1007/978-3-031-44274-2_28.

5 Jérémie Chalopin, Shantanu Das, Matús Mihalák, Paolo Penna, and Peter Widmayer. Data delivery by energy-constrained mobile agents. In Paola Flocchini, Jie Gao, Evangelos Kranakis, and Friedhelm Meyer auf der Heide, editors, *Algorithms for Sensor Systems – 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers*, volume 8243 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2013. doi:10.1007/978-3-642-45346-5_9.

6 Jérémie Chalopin, Riko Jacob, Matús Mihalák, and Peter Widmayer. Data delivery by energy-constrained mobile agents on a line. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 423–434. Springer, 2014. doi:10.1007/978-3-662-43951-7_36.

7 Jurek Czyzowicz, Dariusz Dereniowski, Robert Ostrowski, and Wojciech Rytter. Gossiping by energy-constrained mobile agents in tree networks. *Theor. Comput. Sci.*, 861:45–65, 2021. doi:10.1016/j.tcs.2021.02.009.

8 Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Communication problems for mobile agents exchanging energy. In Jukka Suomela, editor, *Structural Information and Communication Complexity – 23rd International Colloquium, SIROCCO 2016, Helsinki, Finland, July 19-21, 2016, Revised Selected Papers*, volume 9988 of *Lecture Notes in Computer Science*, pages 275–288, 2016. doi:10.1007/978-3-319-48314-6_18.

9 Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Energy-optimal broadcast and exploration in a tree using mobile agents. *Theor. Comput. Sci.*, 795:362–374, 2019. doi:10.1016/j.tcs.2019.07.018.

10 Jurek Czyzowicz, Stefan Dobrev, Ryan Killick, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, Denis Pankratov, and Sunil M. Shende. Graph exploration by energy-sharing mobile agents. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity – 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 – July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 185–203. Springer, 2021. doi:10.1007/978-3-030-79527-6_11.

11 Jurek Czyzowicz, Konstantinos Georgiou, Ryan Killick, Evangelos Kranakis, Danny Krizanc, Manuel Lafond, Lata Narayanan, Jaroslav Opatrny, and Sunil M. Shende. Time-energy tradeoffs for evacuation by two robots in the wireless model. *Theor. Comput. Sci.*, 852:61–72, 2021. doi:10.1016/j.tcs.2020.11.014.

12 X. Sun, N. Kitamura, T. Izumi, and T. Masuzawa. Circulating exploration of an arbitrary graph by energy-sharing agents. In *Proc. of the 2023 IEICE General Conference*, pages 1–2, 2023.