Collecting information by power-aware mobile agents

Julian Anaya[1], Jérémie Chalopin[2], Jurek Czyzowicz[1], Arnaud Labourel[2], Andrzej Pelc[1,⋆], and Yann Vaxès[2]

[1] Université du Québec en Outaouais, C.P. 1250, succ. Hull, Gatineau, Qc. J8X 3X7 Canada.
ingjulianananaya@gmail.com, jurek@uqo.ca, pelc@uqo.ca
[2] LIF, CNRS & Aix-Marseille University, 13453 Marseille, France.
{jeremie.chalopin,arnaud.labourel,yann.vaxes}@lif.univ-mrs.fr

Abstract. A set of identical, mobile agents is deployed in a weighted network. Each agent possesses a battery - a power source allowing the agent to move along network edges. Agents use their batteries proportionally to the distance traveled. At the beginning, each agent has its initial information. Agents exchange the actually possessed information when they meet. The agents collaborate in order to perform an efficient *convergecast*, where the initial information of all agents must be eventually transmitted to some agent.
The objective of this paper is to investigate what is the minimal value of power, initially available to all agents, so that convergecast may be achieved. We study the question in the centralized and the distributed settings. In the distributed setting every agent has to perform an algorithm being unaware of the network. We give a linear-time centralized algorithm solving the problem for line networks. We give a 2-competitive distributed algorithm achieving convergecast for tree networks. The competitive ratio of 2 is proved to be the best possible for this problem, even if we only consider line networks. We show that already for the case of tree networks the centralized problem is strongly NP-complete. We give a 2-approximation centralized algorithm for general graphs.

1 Introduction

The model and the problem

A set of agents is deployed in a network represented by a weighted graph G. An edge weight represents its length, i.e., the distance between its endpoints along the edge. The agents start at different nodes of G. Every agent has a battery : a power source allowing it to move in a continuous way along the network edges. An agent may stop at any point of a network edge (i.e. at any distance from the edge endpoints, up to the edge weight). The movements of an agent use its battery proportionally to the distance traveled. We assume that all agents move at the same speed that is equal to one, i.e., the values of the distance traveled and the time spent while travelling are equivalent. Each agent starts with the same amount of power noted P, allowing all agents to travel the same distance P.

Initially, each agent has an individual piece of information. When two (or more) agents are at the same point of the network at the same time, they automatically detect

⋆ Partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

each other's presence and they exchange their information, i.e., each agent transmits all its possessed information to all other agents present at the point (hence an agent transmits information collected during all previous meetings). The purpose of a *convergecast* algorithm is to schedule the movements of the agents, so that the exchanges of the currently possessed information between the meeting agents eventually result in some agent, not a priori predetermined, holding the union of individual information of all the agents. This task is important, e.g., when agents have partial information about the topology of the network and the aggregate information can be used to construct a map of it, or when individual agents hold measurements performed by sensors located at their initial positions and collected information serves to make some global decision based on all measurements.

Agents try to cooperate so that the convergecast is achieved with the agent's smallest possible initial battery power P_{OPT}, i.e., minimizing the maximum distance traveled by an agent. We investigate the problem in two possible settings, centralized and distributed.

In the centralized setting, the problem must be solved by a centralized authority knowing the network and the initial positions of all the agents. We define a *strategy* as a finite sequence of movements executed by the agents. During each movement, starting at a specific time, an agent walks between two points belonging to the same network edge. A strategy is a convergecast strategy if the sequence of movements results in one agent possessing the initial information of every agent. We consider two different versions of the problem : the decision problem, i.e., deciding if there exists a convergecast strategy using power P (where P is the input of the problem) and the optimization problem, i.e., computing the smallest amount of power that is sufficient to achieve convergecast.

In the distributed setting, the problem must be approached individually by each agent. Each agent is unaware of the network, of its position in the network and without the knowledge of positions (or even the presence) of any other agents. The agents are anonymous, i.e., they must execute the same algorithm. The agent has a very simple sensing device allowing it to detect the presence of other agents at its current location in the network. The agent is also aware of the degree d of the node at which it is located and it can identify all ports represented by integers $1, 2, \ldots d$. The agent is aware of the directions from which are coming all agents currently present at its location (i.e. their entry ports to the current node or their incoming directions if currently located inside an edge). Each agent has memory sufficient to store all information initially belonging to all agents as well as a small (constant) number of real values. Since the measure of efficiency in this paper is the battery power (or the maximum distance traveled by an agent, which is proportional to the battery power used) we do not try to optimize the other resources (e.g. global execution time, local computation time, memory size of the agents, communication bandwidth, etc.). In particular, we conservatively suppose that, whenever two agents meet, they automatically exchange the entire information they possess (rather than the new information only). This information exchange procedure is never explicitly mentioned in our algorithms, supposing, by default, that it always takes place when a meeting occurs. The efficiency of a distributed solution is expressed by the *competitive ratio*, which is the worst-case ratio of the amount of power necessary to

solve the convergecast by the distributed algorithm with respect to the amount of power computed by the optimal centralized algorithm, which is executed for the same agents' initial positions.

It is easy to see, that in the optimal centralized solution for the case of the line and the tree, the original network may be truncated by removing some portions and leaving only the connected part of it containing all the agents (this way all leaves of the remaining tree contain initial positions of agents). We make this assumption also in the distributed setting, since no finite competitive ratio is achievable if this condition is dropped. Indeed, two nearby anonymous agents inside a long line need to travel a long distance to one of its endpoints to break symmetry in order to meet.

Related work

The rapid development of network and computer industry fueled the research interest in mobile agents (robots) computing. Mobile agents are often interpreted as software agents, i.e., programs migrating from host to host in a network, performing some specific tasks. However, the recent developments in computer technology bring up specific problems related to physical mobile devices. These include robots or motor vehicles, various wireless gadgets, or even living mobile agents: humans (e.g. soldiers on the battlefield or emergency disaster relief personnel) or animals (e.g. birds, swarms of insects).

In many applications the involved mobile agents are small and have to be produced at low cost in massive numbers. Consequently, in many papers, the computational power of mobile agents is assumed to be very limited and feasibility of some important distributed tasks for such collections of agents is investigated. For example [6] introduced *population protocols*, modeling wireless sensor networks by extremely limited finite-state computational devices. The agents of population protocols move according to some mobility pattern totally out of their control and they interact randomly in pairs. This is called *passive mobility*, intended to model, e.g., some unstable environment, like a flow of water, chemical solution, human blood, wind or unpredictable mobility of agents' carriers (e.g. vehicles or flocks of birds). On the other hand, [37] introduced anonymous, oblivious, asynchronous, mobile agents which cannot directly communicate, but can occasionally observe the environment. Gathering and convergence [5, 19–21], as well as pattern formation [23, 25, 37, 38] were studied for such agents.

Apart from the feasibility questions for such limited agents, the optimization problems related to the efficient usage of agents' resources have been also investigated. Energy management of (not necessarily mobile) computational devices has been a major concern in recent research papers (cf. [1]). Fundamental techniques proposed to reduce power consumption of computer systems include power-down strategies (see [1, 8, 30]) and speed scaling (introduced in [39]). Several papers proposed centralized [17, 36, 39] or distributed [1, 4, 8, 30] algorithms. However, most of this research on power efficiency concerned optimization of overall power used. Similar to our setting, assignment of charges to the system components in order to minimize the maximal charge has a flavor of another important optimization problem which is load balancing (cf. [10]).

In wireless sensor and ad hoc networks the power awareness has been often related to the data communication via efficient routing protocols (e.g. [4, 36]. However in many

applications of mobile agents (e.g. those involving actively mobile, physical agents) the agent's energy is mostly used for its mobility purpose rather than communication, since active moving often requires running some mechanical components, while communication mostly involves (less energy-prone) electronic devices. Consequently, in most tasks involving moving agents, like exploration, searching or pattern formation, the distance traveled is the main optimization criterion (cf. [2, 3, 11, 12, 15, 16, 22, 24, 26, 33]). Single agent exploration of an unknown environment has been studied for graphs, e.g. [2, 22], or geometric terrains, [12, 16].

While a single agent cannot explore an unknown graph unless pebble (landmark) usage is permitted (see [13]), a pair of robots is able to explore and map a directed graph of maximal degree d in $O(d^2n^5)$ time with high probability (cf. [14]). In the case of a team of collaborating mobile agents, the challenge is to balance the workload among the agents so that the time to achieve the required goal is minimized. However this task is often hard (cf. [28]), even in the case of two agents on a tree, [9]. On the other hand, [26] study the problem of agents exploring a tree showing $O(k/\log k)$ competitive ratio of their distributed algorithm provided that writing (and reading) at tree nodes is permitted.

Assumptions similar to our paper have been made in [11, 16, 24] where the mobile agents are constrained to travel a fixed distance to explore an unknown graph, [11, 16], or tree, [24]. In [11, 16] a mobile agent has to return to its home base to refuel (or recharge its battery) so that the same maximal distance may repeatedly be traversed. [24] gives an 8-competitive distributed algorithm for a set of agents with the same amount of power exploring the tree starting at the same node.

The convergecast problem is sometimes viewed as a special case of the data aggregation question (e.g. [32, 35]) and it has been studied mainly for wireless and sensor networks, where the battery power usage is an important issue (cf. [31, 7]). Recently [18] considered the online and offline settings of the scheduling problem when data has to be delivered to mobile clients while they travel within the communication range of wireless stations. [31] presents a randomized distributed convergecast algorithm for geometric ad-hoc networks and studies the trade-off between the energy used and the latency of convergecast. To the best of our knowledge, the problem of the present paper, when the mobile agents perform convergecast, by exchanging the previously acquired information when meeting, while optimizing the maximal power used by a mobile agent, has never been investigated.

Our results

In the case of centralized setting we give a linear-time deterministic algorithm finding an optimal convergecast strategy for line networks. We show that, already for the case of tree networks, the centralized problem is strongly NP-complete. We give a 2-approximation centralized algorithm for general graphs.

For the distributed setting, we show that the convergecast is possible for tree networks if all agents have the amount of initial power equal to twice the power necessary to achieve centralized convergecast. The competitive ratio of 2 is proved to be the best possible for this problem, even if we only consider line networks. Most proofs are omitted due to lack of space. They will appear in a journal version of this paper.

2 Centralized convergecast on lines

In this section we consider the centralized convergecast problem for lines. We give an optimal, linear-time, deterministic centralized algorithm, computing the optimal amount of power needed to solve convergecast for line networks. As the algorithm is quite involved, we start by observing some properties of the optimal strategies. Already relatively apparent properties permit us to design an intuitive decision procedure, verifying whether a given amount of power is sufficient to perform convergecast. Then we present other ingredients needed for the linear-time optimization procedure.

We order agents according to their positions on the line. Hence we can assume w.l.o.g., that agent a_i, for $1 \leq i \leq n$ is initially positioned at point $Pos[i]$ of the line of length ℓ and that $Pos[1] = 0 < Pos[2] < \ldots < Pos[n] = \ell$.

2.1 Properties of a convergecast strategy

In this subsection, we show that if we are given a convergecast strategy for some configuration, then we can always modify it in order to get another convergecast strategy, using the same amount of maximal power for every agent, satisfying some interesting properties. These observations permit us to restrict the search for the optimal strategy to some smaller and easier to handle subclass of strategies.

Observe that, in order to aggregate the entire information at a single point of the line, every agent a_i, for $1 < i < n$, must learn either the initial information of agent a_1 or a_n. Therefore, we can partition the set of agents performing a convergecast strategy into two subsets LR and RL, such that each agent $a_i \in LR$ learns the initial information of agent a_1 before learning the initial information of agent a_n (or not learning at all the information of a_n). All other agents belong to RL. For any convergecast strategy all the points visited by agent a_i form a real interval containing its initial position $Pos[i]$. We denote by $[b_i, f_i]$ the interval of all points visited by $a_i \in LR$ and by $[f_j, b_j]$ - the points visited by $a_j \in RL$.

In the next lemma, we show a necessary and sufficient condition for the existence of a convergecast strategy. It also shows that any convergecast strategy may be converted to a strategy that we call *regular* having particular properties. Firstly, each agent from LR of a regular strategy is initially positioned left to all agents of RL. Secondly, each agent of regular strategy needs to change its direction at most once. More precisely, each agent $a_i \in LR$ first goes back to a point $b_i \leq Pos[i]$, getting there the information from the previous agent (except a_1 that has no information to collect), then it goes forward to a point $f_i \geq b_i$. Similarly, each agent in RL first goes back to a point $b_i \geq Pos[i]$ and then moves forward to a point $f_i \leq b_i$. Moreover, we assume that each agent of a regular strategy travels the maximal possible distance, i.e., it spends all its power.

Lemma 1. *There exists a convergecast strategy $\mathcal{S}$ for a configuration $Pos[1:n]$ if and only if there exists a partition of the agents into two sets LR and RL and if for each agent a_i, there exist two points b_i, f_i of segment $[0, \ell]$ such that*

1. *there exists p such that $LR = \{a_i \mid i \leq p\}$ and $RL = \{a_i \mid i > p\}$,*
2. *if $a_i \in LR$, $b_i = \min\{f_{i-1}, Pos[i]\}$ ($b_1 = Pos[1] = 0$) and $f_i = \min\{2b_i + P - Pos[i], \ell\}$,*

3. *if* $a_i \in RL$, $b_i = \max\{f_{i+1}, Pos[i]\}$ *(*$b_n = Pos[n] = \ell$*) and* $f_i = \max\{2b_i - P - Pos[i], 0\}$,
4. $\max\{f_i \mid a_i \in LR\} \geq \min\{f_i \mid a_i \in RL\}$.

In the following, we only consider regular strategies. Note that a regular strategy is fully determined by the value of P and by the partition of the agents into the two sets LR and RL. For each agent $a_i \in LR$ (resp. $a_i \in RL$), we denote f_i by $Reach_{LR}(i, P)$ (resp. $Reach_{RL}(i, P)$). Observe that $Reach_{LR}(i, P)$ is the rightmost point on the line to which the set of i agents at initial positions $Pos[1 : i]$, each having power P, may transport the union of their initial information. Similarly, $Reach_{RL}(i, P)$ is the leftmost such point for agents at positions $Pos[i : n]$.

Lemma 1 permits to construct a linear-time decision procedure verifying if a given amount P of battery power is sufficient to design a convergecast strategy for a given configuration $Pos[1 : n]$ of agents. We first compute two lists $Reach_{LR}(i, P)$, for $1 \leq i \leq n$ and $Reach_{RL}(i, P)$, for $1 \leq i \leq n$. Then we scan them to determine if there exists an index j, such that $Reach_{LR}(j, P) \geq Reach_{RL}(j + 1, P)$. In such a case, we set $LR = \{a_r \mid r \leq j\}$ and $RL = \{a_r \mid r > j\}$ and we apply Lemma 1 to obtain a convergecast strategy where agents a_j and a_{j+1} meet and exchange their information which totals to the entire initial information of the set of agents. If there is no such index j, no convergecast strategy is possible. This implies

Corollary 1. *In $O(n)$ time we can decide if a configuration of n agents on the line, each having a given maximal power P, can perform convergecast.*

The remaining lemmas of this subsection bring up observations needed to construct an $O(n)$ algorithm designing an optimal centralized convergecast strategy.

Note that if the agents are not given enough power, then it can happen that some agent a_p may never learn the information from a_1 (resp. from a_n). In this case, a_p cannot belong to LR (resp. RL). We denote by $Act_{LR}(p)$ the minimum amount of power we have to give the agents to ensure that a_p can learn the information from a_1: if $p > 0$, $Act_{LR}(p) = \min\{P \mid Reach_{LR}(p - 1, P) + P \geq Pos[p]\}$. Similarly, we have $Act_{RL}(p) = \min\{P \mid Reach_{RL}(p + 1, P) - P \leq Pos[p]\}$.

Given a strategy using power P, for each agent $p \in LR$, we have $P \geq Act_{LR}(p)$ and either $Reach_{LR}(p - 1, P) \geq Pos[p]$, or $Reach_{LR}(p - 1, P) \leq Pos[p]$. In the first case, $Reach_{LR}(p, P) = Pos[p] + P$, while in the second case, $Reach_{LR}(p, P) = 2Reach_{LR}(p - 1, P) + P - Pos[p]$.

We define threshold functions $TH_{LR}(p)$ and $TH_{RL}(p)$ that compute for each index p, the minimal amount of agents' power ensuring that agent a_p does not go back when $a_p \in LR$ or $a_p \in RL$ respectively (i.e. such that $b_p = Pos[p]$). For each p, let $TH_{LR}(p) = \min\{P \mid Reach_{LR}(p, P) = Pos[p] + P\}$ and $TH_{RL}(p) = \min\{P \mid Reach_{RL}(p, P) = Pos[p] - P\}$. Clearly, $TH_{LR}(1) = TH_{RL}(n) = 0$.

The next lemma illustrates how to compute $Reach_{LR}(q, P)$ and $Reach_{RL}(q, P)$ if we know $TH_{LR}(p)$ and $TH_{RL}(p)$ for every agent p.

Lemma 2. *Consider an amount of power P and an index q. If $p = \max\{p' \leq q \mid TH_{LR}(p') < P\}$, then $Reach_{LR}(q, P) = 2^{q-p}Pos[p] + (2^{q-p+1} - 1)P - \sum_{i=p+1}^{q} 2^{q-i}Pos[i]$. Similarly, if $p = \min\{p' \geq q \mid TH_{RL}(p') < P\}$, then $Reach_{RL}(q, P) = 2^{p-q}Pos[p] - (2^{p-q+1} - 1)P - \sum_{i=q}^{p-1} 2^{i-q}Pos[i]$.*

Observe that the previous lemma implies that, for each q, the function $Reach_{LR}(q,\cdot)$ is an increasing, continuous, piecewise linear function on $[Act_{LR}(q),+\infty)$ and that $Reach_{RL}(q,\cdot)$ is a decreasing, continuous, piecewise linear function on $[Act_{RL}(q),+\infty)$.

In the following, we denote $S_{LR}(p,q) = \sum_{i=p+1}^{q} 2^{q-i} Pos[i]$ and $S_{RL}(p,q) = \sum_{i=q}^{p-1} 2^{i-q} Pos[i]$.

Remark 1. For every $p \leq q \leq r$, $S_{LR}(p,r) = 2^{r-q} S_{LR}(p,q) + S_{LR}(q,r)$.

We now show that for an optimal convergecast strategy, the last agent of LR and the first agent of RL meet at some point between their initial positions and that they need to use all the available power to meet.

Lemma 3. *Suppose there exists an optimal convergecast strategy for a configuration $Pos[1:n]$, where the maximum power used by an agent is P. Then, there exists an integer $1 \leq p < n$ such that $Pos[p] < Reach_{LR}(p,P) = Reach_{RL}(p+1,P) < Pos[p+1]$.*

Moreover, $\forall q \leq p$, $Act_{LR}(q) < P < TH_{RL}(q)$ and $\forall q > p$, $Act_{RL}(q) < P < TH_{LR}(q)$.

2.2 A linear algorithm to compute the optimal power needed for convergecast

In this section, we prove the following theorem.

Theorem 1. *An optimal convergecast strategy for the line can be computed in linear time.*

We first explain how to compute a stack of couples $(p, TH_{LR}(p))$that we can subsequently use to compute $Reach_{LR}(p,P)$ for any given P. Then, we present a linear algorithm that computes the value needed to solve convergecast when the last index $r \in LR$ is provided: given an index r, we compute the optimal power needed to solve convergecast assuming that $LR = \{a_q \mid q \leq r\}$ and $RL = \{a_q \mid q > r\}$. Finally, we explain how to use techniques introduced for the two previous algorithms in order to compute the optimal power needed to solve convergecast. These computations directly imply the schedule of the agents' moves of the optimal convergecast strategy.

Computing the thresholds values. To describe explicitly the function $Reach_{LR}(q,\cdot)$, we need to identify the indexes p such that for every $r \in [p+1,q]$, we have $TH_{LR}(r) > TH_{LR}(p)$. They correspond to the breakpoints at which the slopes of the piecewise linear function $Reach_{LR}(q,\cdot)$ change. Indeed, if we are given such an index p, then for every P comprised between $TH_{LR}(p)$ and $\min\{TH_{LR}(r) \mid p < r \leq q\}$, we have $Reach_{LR}(q,P) = 2^{q-p} Pos[p] + (2^{q-p+1}-1)P - S_{LR}(p,q)$. We denote by $X_{LR}(q)$ this set of indexes $\{p \leq q \mid \forall r \in [p+1,q], TH_{LR}(r) > TH_{LR}(p)\}$.

In particular, if we want to compute $TH_{LR}(q+1)$, we just need to find $p = \max\{r \leq q \mid Reach_{LR}(q, TH_{LR}(r)) < Pos[q+1]\}$, and then $TH_{LR}(q+1)$ is the value of power P such that $2^{q-p} Pos[p] + (2^{q-p+1}-1)P - S_{LR}(p,q) = Pos[q+1]$. Moreover, by the choice of p, we have $X_{LR}(q+1) = \{r \in X_{LR}(q) \mid r \leq p\} \cup \{q+1\}$.

Using these remarks, the function ThresholdLR, having been given an agent index r, returns a stack $\mathtt{TH_{LR}}$ containing couples (p, P) such that $p \in X_{LR}(r)$ and $P = TH_{LR}(p)$. Note that in the stack $\mathtt{TH_{LR}}$, the elements (p, P) are sorted along both components, the largest being on the top of the stack.

The algorithm proceeds as follows. Initially, the stack $\mathtt{TH_{LR}}$ contains only the couple $(1, TH_{LR}(1))$. At each iteration, given the stack corresponding to the index q, in order to compute the stack for the index $q+1$, we first pop out all elements (p, P) such that $Reach_{LR}(q, P) > Pos[q+1]$. After that, the integer p needed to compute $TH_{LR}(q+1)$ is located on the top of the stack. Finally, the couple $(q+1, TH_{LR}(q+1))$ is pushed on the stack before we proceed with the subsequent index q. At the end of the procedure, we return the stack $\mathtt{TH_{LR}}$ corresponding to the index r.

The number of stack operations performed during the execution of this function is $O(r)$. However, in order to obtain a linear number of arithmetic operations, we need to be able to compute 2^{q-p} and $S_{LR}(p, q)$ in constant time.

In order to compute 2^{q-p} efficiently, we can store the values of 2^i, $i \in [1, n-1]$ in an auxiliary array, that we have precomputed in $O(n)$ time. We cannot precompute all values of $S_{LR}(p, q)$ since this requires calculating $\Theta(n^2)$ values. However, from Remark 1, we know that $S_{LR}(p, q) = S_{LR}(1, q) - 2^{q-p} S_{LR}(1, p)$. Consequently, it is enough to precompute $S_{LR}(1, i)$ for each $i \in [2, n]$. Since $S_{LR}(1, i+1) = 2S_{LR}(1, i) + Pos[i+1]$, this can be done using $O(n)$ arithmetic operations.

Function ThresholdLR(array Pos[1:n] of real; r:integer):stack

---

$\mathtt{TH_{LR}} =$ **empty_stack**;
push $(\mathtt{TH_{LR}}, (1, 0))$;
for $q = 1$ **to** $r - 1$ **do**
  $(p, P) =$ **pop**$(\mathtt{TH_{LR}})$; /* $p = q$ and $P = TH_{LR}(p)$ */
  while $2^{q-p} * Pos[p] + (2^{q-p+1} - 1) * P - S_{LR}(p, q) \geq Pos[q+1]$ **do**
  $(p, P) =$ **pop**$(\mathtt{TH_{LR}})$;
  /* while $Reach_{LR}(q, P) \geq Pos[q+1]$ we consider the next element in $\mathtt{TH_{LR}}$ */
  push $(\mathtt{TH_{LR}}, (p, P))$;
  $Q = (2^{q-p} * Pos[p] - Pos[q+1] - S_{LR}(p, q))/(2^{q-p+1} - 1)$;
  /* Q is the solution of $Reach_{LR}(q, P) = Pos[q+1]$ */
  push $(\mathtt{TH_{LR}}, (q+1, Q))$;
return $(\mathtt{TH_{LR}})$;

Similarly, we can define the function ThresholdRL (array Pos[1 : n] of real, r:integer):stack that returns a stack $\mathtt{TH_{RL}}$ containing all pairs $(q, TH_{RL}(q))$ such that for every $p \in [r, q-1]$, we have $TH_{RL}(p) > TH_{RL}(q)$.

Computing the optimal power when LR and RL are known. Suppose now that we are given an agent index r and we want to compute the optimal power needed to solve convergecast when $LR = \{a_p \mid p \leq r\}$ and $RL = \{a_q \mid q > r\}$. From Lemma 3, we know that there exists a unique P_{OPT} such that $Reach_{LR}(r, P_{OPT}) = Reach_{RL}(r+1, P_{OPT})$.

As previously, by Lemma 2, we know that the value of $Reach_{LR}(r, P_{OPT})$ depends on $p = \max\{p' \leq r \mid TH_{LR}(p') < P_{OPT}\}$. Similarly, $Reach_{RL}(r+1, P_{OPT})$ depends on $q = \min\{q' \geq r+1 \mid TH_{RL}(q') < P_{OPT}\}$. If we are given the values of p and q, then P_{OPT} is the value of P such that

$$2^{r-p}Pos[p] - (2^{r-p+1}-1)P - S_{LR}(p,r) = 2^{q-r-1}Pos[q] - (2^{q-r}-1)P - S_{RL}(q,r+1).$$

In Algorithm `OptimalAtIndex`, we first use the previous algorithm to compute the two stacks $\texttt{TH}_{\texttt{LR}}$ and $\texttt{TH}_{\texttt{RL}}$ containing respectively $\{(p, TH_{LR}(p)) \mid p \in X_{LR}(r)\}$ and $\{(q, TH_{RL}(q)) \mid q \in X_{RL}(r+1)\}$. Then at each iteration, we consider the two elements (p, P_{LR}) and (q, P_{RL}) that are on top of both stacks. If $P_{LR} \geq P_{RL}$ (the other case is symmetric), we check whether $Reach_{LR}(r, P_{LR}) \geq Reach_{RL}(r+1, P_{LR})$. In this case, we have $P > P_{OPT}$, so we remove (p, P_{LR}) from the stack $\texttt{TH}_{\texttt{LR}}$ and we proceed to the next iteration. If $Reach_{LR}(r, P_{LR}) < Reach_{RL}(r+1, P_{LR})$, we know that $P_{OPT} \geq P_{LR} \geq P_{RL}$ and we can compute the value of P_{OPT} using Lemma 2.

Function `OptimalAtIndex(array` Pos`[1:n] of real; r:integer):stack`

$\texttt{TH}_{\texttt{LR}} = \texttt{ThresholdLR}(r)$; $\texttt{TH}_{\texttt{RL}} = \texttt{ThresholdRL}(r+1)$;
$(p, P_{LR}) =$ **pop**$(\texttt{TH}_{\texttt{LR}})$; $(q, P_{RL}) =$ **pop**$(\texttt{TH}_{\texttt{RL}})$; $P = \max\{P_{LR}, P_{RL}\}$;
/* $p = r$, $P_{LR} = TH_{LR}(r)$, $q = r+1$, $P_{RL} = TH_{RL}(r+1)$. */
while
$2^{r-p}Pos[p] + (2^{r-p+1}-1)P - S_{LR}(p,r) \geq 2^{q-r-1}Pos[q] - (2^{q-r}-1)P - S_{RL}(q,r+1)$
do /* While $Reach_{LR}(r,P) \geq Reach_{RL}(r+1,P)$ do */
  if $P_{LR} \geq P_{RL}$ **then** $(p, P_{LR}) =$ **pop**$(\texttt{TH}_{\texttt{LR}})$;
  else $(q, P_{RL}) =$ **pop**$(\texttt{TH}_{\texttt{RL}})$;
  $P = \max\{P_{LR}, P_{RL}\}$;
$P_{OPT} =$
$(2^{q-r-1}Pos[q] - S_{RL}(q,r+1) - 2^{r-p}Pos[p] + S_{LR}(p,r))/(2^{r-p+1} + 2^{q-r} - 2)$;
/* P_{OPT} is the solution of
  $Reach_{LR}(r, P_{OPT}) = Reach_{RL}(r+1, P_{OPT})$ */
return (P_{OPT});

Let $Y_{LR}(r, P)$ denote $\{(p, TH_{LR}(p)) \mid p \in X_{LR}(r)$ and $TH_{LR}(p) < P\}$ and $Y_{RL}(r+1, P) = \{(q, TH_{RL}(q)) \mid q \in X_{RL}(r+1)$ and $TH_{RL}(q) < P\}$.

Remark 2. At the end of the execution of the function `OptimalAtIndex`, $\texttt{TH}_{\texttt{LR}}$ and $\texttt{TH}_{\texttt{RL}}$ contain respectively $Y_{LR}(r, P_{OPT})$ and $Y_{RL}(r+1, P_{OPT})$.

Moreover, if initially the two stacks $\texttt{TH}_{\texttt{LR}}$ and $\texttt{TH}_{\texttt{RL}}$ contain respectively $Y_{LR}(r, P)$ and $Y_{RL}(r+1, P)$ for some $P \geq P_{OPT}$, then the value computed by the algorithm is also P_{OPT} .

Computing the optimal power for convergecast. We now explain how to compute the optimal amount of power needed to achieve convergecast using a linear number of operations.

Let $P_{<r}$ be the optimal value needed to solve convergecast when $\max\{s \mid a_s \in LR\} < r$, i.e., when the two agents whose meeting results in merging the entire information are a_i and a_{i+1} for some $i < r$. If $Reach_{LR}(r, P_{<r}) \leq Reach_{RL}(r+1, P_{<r})$, then $P_{<r+1} = P_{<r}$. However, if $Reach_{LR}(r, P_{<r}) > Reach_{RL}(r+1, P_{<r})$, then $P_{<r+1} < P_{<r}$ and $P_{<r+1}$ is the unique value of P such that $Reach_{LR}(r, P) = Reach_{RL}(r+1, P)$. This corresponds to the value returned by `OptimalAtIndex` (Pos, r).

The general idea of Algorithm `ComputeOptimal` is to iteratively compute the value of $P_{<r}$. If we need a linear time algorithm, we cannot call repeatedly the function `OptimalAtIndex`. However, from Remark 2, in order to compute $P_{<r+1}$ when $P_{<r+1} \leq P_{<r}$, it is enough to know $Y_{LR}(r, P_{<r})$ and $Y_{RL}(r+1, P_{<r})$. If we know $Y_{LR}(r, P_{<r})$ and $Y_{RL}(r+1, P_{<r})$, then we can use the same algorithm as in `OptimalAtIndex` in order to compute $P_{<r+1}$. Moreover, from Remark 2, we also get $Y_{LR}(r, P_{<r+1})$ and $Y_{RL}(r+1, P_{<r+1})$ when we compute $P_{<r+1}$.

Before proceeding to the next iteration, we need to compute $Y_{LR}(r+1, P_{<r+1})$ and $Y_{RL}(r+2, P_{<r+1})$ from $Y_{LR}(r, P_{<r+1})$ and $Y_{RL}(r+1, P_{<r+1})$. Note that if $TH_{LR}(r) > P_{<r+1}$, then $Y_{LR}(r+1, P_{<r+1}) = Y_{LR}(r, P_{<r+1})$. If $TH_{LR}(r) \leq P_{<r+1}$, we can use the same algorithm as in `ThresholdLR` to compute $Y_{LR}(r+1, P_{<r+1}) = \{(p, TH_{LR}(p)) \mid p \in X_{LR}(r)\}$ from $Y_{LR}(r, P_{<r+1})$. Consider now $Y_{RL}(r+2, P_{<r+1})$. If $TH_{RL}(r+1) > P_{<r+1}$, then $(r+1, TH_{RL}(r+1)) \notin Y_{RL}(r+1, P_{<r+1})$, and $Y_{RL}(r+2, P_{<r+1}) = Y_{RL}(r+1, P_{<r+1})$. If $TH_{RL}(r+1) \leq P_{<r+1}$, then either $Pos[r+1] - P_{<r+1} \geq Reach_{RL}(r+1, P_{<r+1})$ if $P_{<r+1} = P_{<r}$, or $Pos[r+1] - P_{<r+1} = Reach_{RL}(r+1, P_{<r+1}) = Reach_{LR}(r, P_{<r+1})$ if $P_{<r+1} < P_{<r}$. In both cases, it implies that $Act_{LR}(r+1) \geq P_{<r+1}$. Therefore, by Lemma 3, $P_{<i} = P_{<r+1}$ for every $i \geq r+1$ and we can return the value of $P_{<r+1}$.

In Algorithm `ComputeOptimal`, at each iteration, the stack $\texttt{TH}_{\texttt{LR}}$ contains $Y_{LR}(r, P_{<r})$ (except its top element) and the stack $\texttt{TH}_{\texttt{RL}}$ contains $Y_{RL}(r+1, P_{<r})$ (except its top element). Initially, $\texttt{TH}_{\texttt{LR}}$ is empty and $\texttt{TH}_{\texttt{RL}}$ contains $O(n)$ elements. In each iteration, at most one element is pushed into the stack $\texttt{TH}_{\texttt{LR}}$ and no element is pushed into the stack $\texttt{TH}_{\texttt{RL}}$. Consequently, the number of stack operations performed by the algorithm is linear.

3 Distributed convergecast on trees

A configuration of convergecast on graphs is a couple (G, A) where G is the weighted graph encoding the network and A is the set of the starting nodes of the agents. Let $D(G, A) = \max_{\emptyset \subsetneq X \subsetneq A}\{\min_{x \in X, y \in A \setminus X}\{d_G(x, y)\}\}$ where $d_G(x, y)$ is the distance between x and y in G. Clearly, we have $D(G, A) \leq 2P_{OPT}$.

We consider weighted trees with agents at every leaf. The next theorem states that there exists a 2-competitive distributed algorithm for the convergecast problem on trees.

Theorem 2. *Consider a configuration (T, A) where T is a tree and A contains all the leaves of T. There is a distributed convergecast algorithm using $D(T, A) \leq 2P_{OPT}$ power per agent.*

Sketch of the proof : In order to perform the convergecast, each agent executes Algorithm 1.

Algorithm 1: UnknownTree

```
collecting = true;
while collecting = true do
    Wait until there is at most one port unused by the agent incoming at the current node;
    if all ports of current node were used by incoming agents then
    collecting = false;
    if the agent has used less power than any other agent present at the node and
    collecting = true then
        Move through the unused incoming port until you meet another agent or reach a node;
    else collecting = false;
    if the agent is inside an edge then collecting = false;
```

The agents traverse the leaf edges then edges between nodes at height one and two and so on. When all the tree is traversed, all the information is collected at the last meeting point. No agent will use more power than $D(T, A) \leq 2P_{OPT}$. □

The following theorem shows that no distributed algorithm may offer a better competitive ratio than 2 even if we only consider line networks.

Theorem 3. *Consider any $\delta > 0$, and any value of power P. There exists an integer n and a configuration $Pos[1 : n]$ of n agents on the line such that there is a convergecast strategy using power P and so that there is no deterministic distributed strategy allowing the agents to solve convergecast when the amount of power given to each agent is $(2 - \delta)P$.*

Sketch of the proof : Let $\varepsilon = \delta P/4$ and $\sigma = \varepsilon/2 = \delta P/8$. Let $l = \lfloor \log(8/\delta) \rfloor$ and $k = l + 2$.

Consider a set of agents positioned on a line as follows (See Figure 1). There is an agent a_0 (resp. a_{2l+1}) at the left (resp. right) end of the line on position $s'_0 = 0$ (resp. $s_{2l+1} = \ell$). For each $1 \leq i \leq 2l$, there is a set A_i of k agents on distinct initial positions within a segment $[s_i, s'_i]$ of length σ such that for each $1 \leq i \leq 2l + 1$, the distance between s_i and s'_{i-1} is $2(P - \varepsilon)$. Using Lemma 2, we can show, that if the amount of power given to each agent is P, then convergecast is achievable.

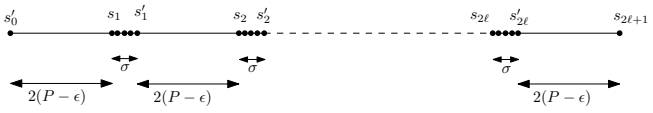

Fig. 1. The configuration in the proof of Theorem 3.

Suppose now that there exists a distributed strategy $\mathcal{S}$ that solves convergecast on the configuration when the amount of power given to each agent is $(2 - \delta)P$. We can

show that for each $i \in [1, l]$, all agents from A_i perform the same moves as long as the leftmost agent of A_i has not met any agent from A_{i-1}. We show by induction on $i \in [1, l]$ that agents in A_i learn the information from a_0 before the one from a_{2l+1} and that no agent in the set A_i knowing the information from a_0 can reach the point $s_{i+1} - (2^{i+2} - 2)\varepsilon$. If we consider the agents from A_l, we get that no agent from A_{l-1} can reach $s_l - (2^{l+1} - 2)\varepsilon \leq s_l - (8/\delta - 2)\delta P/4 = s_l - 2P + \delta P/2 < s_l - 2P + \delta P$ having the initial information of a_0. Since no agent from the set A_l can reach any point on the left of $s_l - 2P + \delta P$, it implies that no agent from A_l can ever learn the information from a_0 and thus, $\mathcal{S}$ is not a distributed convergecast strategy. □

4 Centralized convergecast on trees and graphs

We show in this section that for trees the centralized convergecast problem is substantially harder than for lines.

Theorem 4. *The centralized convergecast decision problem is strongly NP-complete for trees.*

Sketch of the proof : We construct a polynomial-time many-one reduction from the 3-Partition problem which is strongly NP-Complete [27]. The 3-partition problem answers the following question: can a given multiset S of $3m$ positive integers x_i such that $R/4 < x_i < R/2$ be partitioned into m disjoint sets $S_1, S_2, \ldots, S_m$ of size three such that for $1 \leq j \leq m$, $\sum_{x \in S_j} x = R$? The instance of the centralized convergecast problem constructed from an instance of 3-partition is the star depicted in Figure 2. There is an agent at each leaf and each agent is given power equal to $2R + 1$. We can assume that in any convergecast strategy, agents a_i first move to the center u, each agent b_i moves at distance x_i from u and agent c moves at distance $2R$ from u. Agents a_i, for $1 \leq i \leq m$, must then collect the information from agents b_i and finally agent a_{m+1} must move to c in order to complete the convergecast. Since agents a_i, while reaching u have the remaining power of $2R$ and that collecting information from b_i and return to node u costs $2x_i$ power, the instance of convergecast has a solution if and only if the original instance of 3-partition has a solution.

It remains to show that the problem is in NP. Given a strategy $\mathcal{S}$, the certificate of the instance encodes in chronological order the positions of meetings in which at least one agent learns a new piece of information. There is a polynomial number of such meetings called useful meetings. We can show that the strategy $\mathcal{S}'$, in which agents move via shortest paths to their useful meetings, is a convergecast strategy and uses less power than $\mathcal{S}$. If a useful meeting occurs on an edge, the certificate encodes the name of a variable d_i that represents the exact position inside the edge. Checking that we can assign values to d_i, such that each agent moves a distance less than P in $\mathcal{S}'$, can be done in polynomial time using linear programming. □

Even if the exact centralized optimization problem is NP-complete, we can obtain a 2-approximation of the power needed to achieve centralized convergecast in arbitrary graphs in polynomial time.

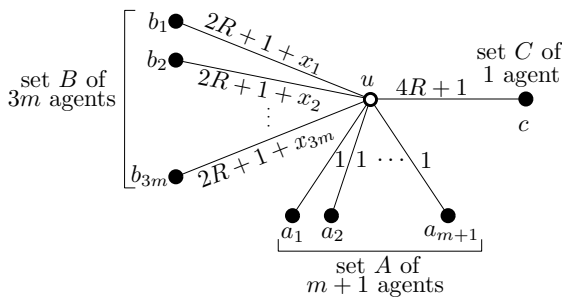

Fig. 2. Instance of centralized convergecast problem from an instance of 3-partition in the proof of Theorem 4.

Theorem 5. *Consider a configuration(G, A) for an arbitrary graph G. There is a polynomial algorithm computing a centralized convergecast strategy using $D(G, A) \leq 2P_{OPT}$ power per agent.*

Sketch of the proof : The idea of the algorithm is to construct a total order u_i on the positions of agents in the following way. We put in set V an arbitrary agent's initial position u_1. Iteratively, we add to the set V of already treated agents' positions a new agent's position u_i, which is at the closest distance from V along some path P_i. Then agents move in the reverse order, starting with agent at u_n. Agent at u_i moves following the path P_i. The length of P_i is less than $D(G, A)$. When all agents have moved, the agent at u_1 gets all the information. □

5 Conclusion and open problems

It is worth pursuing questions related to information transportation by mobile agents to other communication tasks, such as broadcasting or gossiping. Only some of our techniques on convergecast extend to these settings (e.g. NP-hardness for trees).

The problem of a single *information transfer* by mobile agents between two stationary points of the network is also interesting. In particular, it is an open question whether this problem for tree networks is still NP-hard or if a polynomial-time algorithm is possible, since our reduction to 3-partition is no longer valid.

Other related questions may involve agents with unequal power, agents with non-zero visibility, labeled agents, unreliable agents or networks, etc.

References

1. S. Albers: Energy-efficient algorithms. *Comm. ACM 53(5)*, (2010), pp. 86-96.
2. S. Albers, M.R. Henzinger. Exploring unknown environments. *SIAM J. on Comput.*, 29 (4), pp.1164-1188.
3. S. Alpern and S. Gal, The theory of search games and rendezvous. *Kluwer Academic Publ.*, 2002.

4. Ambuhl, C. An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In *Proc. of 32nd ICALP* (2005), pp. 1139-1150.
5. H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Trans. on Robotics and Automation*, 15(5) (1999), pp. 818-828.
6. D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, *Distributed Computing* (2006), pp. 235-253.
7. V. Annamalai, S. K. S. Gupta, and L. Schwiebert, On Tree-Based Convergecasting in Wireless Sensor Networks, *Wireless Communications and Networking, IEEE*, vol. 3 (2003), pp. 1942 - 1947.
8. J. Augustine, S. Irani, C. Swamy. Optimal powerdown strategies. *SIAM J. Comput. 37* (2008), pp. 1499-1516.
9. I. Averbakh, O. Berman. A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. *Discrete Applied Mathematics*, 68 (1996), pp. 17-32.
10. Y. Azar. On-line load balancing. In: *A. Fiat and G.Woeginger, Online Algorithms: The State of the Art, Springer* LNCS 1442, (1998), pp. 178-195.
11. B. Awerbuch, M. Betke, R. Rivest, M. Singh. Piecemeal graph exploration by a mobile robot. *Information and Computation*, 152 (1999), pp. 155-172.
12. R.A. Baeza Yates, J.C. Culberson, and G.J.E. Rawlins. Searching in the Plane. *Information and Computation*, 106(2), (1993), pp. 234-252.
13. M. Bender, A. Fernandez, D. Ron, A. Sahai, S. Vadhan. The power of a pebble: exploring and mapping directed graphs. In *Proc. 30th STOC*, (1998), pp. 269-278.
14. M. Bender, D. Slonim, D.: The power of team exploration: two robots can learn unlabeled directed graphs. In *Proc. 35th FOCS* (1994), pp. 75-85.
15. M. Betke, R.L. Rivest, M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2/3), (1995), pp. 231-254.
16. A. Blum, P. Raghavan, B. Schieber. Navigating in unfamiliar geometric terrain, *SIAM J. Comput.*, 26(1), (1997), pp. 110-137.
17. Bunde, D. P. Power-aware scheduling for makespan and flow. *SPAA* (2006), pp. 190-196.
18. F. Chen, M. P. Johnson, Y. Alayev, A. Bar-Noy, T. F. La Porta, Who, When, Where: Timeslot Assignment to Mobile Clients, *IEEE Transactions on Mobile Computing*, (2012), vol. 11, no. 1, pp. 73-85.
19. M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro. Solving the Robots Gathering Problem, In *Proc ICALP*, (2003), pp. 1181-1196.
20. R. Cohen and D. Peleg. Convergence Properties of the Gravitational Algorithm in Asynchronous Robot Systems. *SIAM J. on Comput.*, 34(6), (2005), pp. 1516-1528.
21. A. Cord-Landwehr, B. Degener, M. Fischer, M. Hullmann, B. Kempkes, A. Klaas, P. Kling, S. Kurras, M. Martens, F. Meyer auf der Heide, C. Raupach, K. Swierkot, D. Warner, C. Weddemann, D. Wonisch: A New Approach for Analyzing Convergence Algorithms for Mobile Robots. *Proc. ICALP (2)*, (2011), pp. 650-661.
22. X. Deng, C. H. Papadimitriou, Exploring an unknown graph. In *Proc. 31st FOCS*, (1990), vol I, pp. 355-361.
23. S. Das, P. Flocchini, N. Santoro, M. Yamashita. On the Computational Power of Oblivious Robots: Forming a Series of Geometric Patterns, In *Proc. PODC*, (2010), pp. 267-276.
24. M. Dynia, M. Korzeniowski, C. Schindelhauer. Power-aware collective tree exploration. In *Proc. of ARCS* (2006), pp. 341-351.
25. P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer. Gathering of asynchronous robots with limited visibility, *Th. Comp. Science*, 337, (2005), pp. 147-168.
26. P. Fraigniaud, L. Gasieniec, D. Kowalski, A. Pelc. Collective tree exploration. In *Proc. LATIN*, (2004), pp. 141-151.

27. M. R. Garey, and D. S. Johnson, Computers and Intractability; *A Guide to the Theory of NP-Completeness* (1979). pp. 96–105 and 224.
28. G. Frederickson, M. Hecht, C. Kim. Approximation algorithms for some routing problems. *SIAM J. on Comput.*, 7 (1978), pp. 178-193.
29. W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, An Application-Specific Protocol Architecture for Wireless Microsensor Networks, *Transactions on wireless communication*, Vol. 1, No. 4, (2002), pp. 660-670.
30. S. Irani, S.K. Shukla, R. Gupta. Algorithms for power savings. *ACM Trans. on Algorithms*, Vol. 3, No. 4, Article 41, (2007).
31. A. Kesselman and D. R. Kowalski, Fast distributed algorithm for convergecast in ad hoc geometric radio networks, *Journal of Parallel and Distributed Computing*, Vol. 66, No. 4 (2006), pp. 578-585.
32. L. Krishnamachari, D. Estrin, S. Wicker. The impact of data aggregation in wireless sensor networks. *ICDCS Workshops*, (2002), pp. 575-578.
33. N. Megow, K. Mehlhorn, P. Schweitzer. Online Graph Exploration: New Results on Old and New Algorithms. *Proc. ICALP (2)*, (2011), pp. 478-489.
34. S. Nikoletseas and P. G. Spirakis. Distributed Algorithms for Energy Efficient Routing and Tracking in Wireless Sensor Networks, *Algorithms*, 2, (2009), pp. 121-157.
35. R. Rajagopalan, P. K. Varshney, Data-aggregation techniques in sensor networks: a survey *Communications Surveys and Tutorials, IEEE*, Vol. 8 , No. 4, (2006), pp. 48 - 63.
36. Stojmenovic, I. and Lin, X. Power-Aware Localized Routing in Wireless Networks. *IEEE Trans. Parallel Distrib. Syst.* 12(11), (2001), pp. 1122-1133.
37. I. Suzuki and M. Yamashita, Distributed Anonymous Mobile Robots: Formation of Geometric Patterns, *SIAM J. Comput.*, vol. 28(4), (1999), pp. 1347-1363.
38. M. Yamashita, I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Th. Comp. Science*, 411(26-28), 2010, pp. 2433-2453.
39. F.F. Yao, A.J. Demers, S. Shenker, A scheduling model for reduced CPU energy. In *Proc. of 36th FOCS* (1995), pp. 374-382.