Asynchronous approach in the plane: A deterministic polynomial algorithm

Sébastien Bouchard[†], Marjorie Bournat[†], Yoann Dieudonné[‡], Swan Dubois[†], Franck Petit[†]
[†] Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA, LIP6 UMR 7606, Paris, France
[‡] MIS Lab., Université de Picardie Jules Verne, France

February 16, 2017

Abstract

In this paper we study the task of *approach* of two mobile agents having the same limited range of vision and moving asynchronously in the plane. This task consists in getting them in finite time within each other's range of vision. The agents execute the same deterministic algorithm and are assumed to have a compass showing the cardinal directions as well as a unit measure. On the other hand, they do not share any global coordinates system (like GPS), cannot communicate and have distinct labels. Each agent knows its label but does not know the label of the other agent or the initial position of the other agent relative to its own. The route of an agent is a sequence of segments that are subsequently traversed in order to achieve approach. For each agent, the computation of its route depends only on its algorithm and its label. An adversary chooses the initial positions of both agents in the plane and controls the way each of them moves along every segment of the routes, in particular by arbitrarily varying the speeds of the agents. Roughly speaking, the goal of the adversary is to prevent the agents from solving the task, or at least to ensure that the agents have covered as much distance as possible before seeing each other. A deterministic approach algorithm is a deterministic algorithm that always allows two agents with any distinct labels to solve the task of approach regardless of the choices and the behavior of the adversary. The cost of a complete execution of an approach algorithm is the length of both parts of route travelled by the agents until approach is completed.

Let Δ and l be the initial distance separating the agents and the length of (the binary representation of) the shortest label, respectively. *Assuming that Δ and l are unknown to both agents, does there exist a deterministic approach algorithm always working at a cost that is polynomial in Δ and l?*

Actually the problem of approach in the plane reduces to the network problem of rendezvous in an infinite oriented grid, which consists in ensuring that both agents end up meeting at the same time at a node or on an edge of the grid. By designing such a rendezvous algorithm with appropriate properties, as we do in this paper, we provide a positive answer to the above question.

Our result turns out to be an important step forward from a computational point of view, as the other algorithms allowing to solve the same problem either have an exponential cost in the initial separating distance and in the labels of the agents, or require each agent to know its starting position in a global system of coordinates, or only work under a much less powerful adversary.

Keywords: mobile agents, asynchronous rendezvous, plane, infinite grid, deterministic algorithm, polynomial cost.

1 Introduction

1.1 Model and Problem

The distributed system considered in this paper consists of two *mobile agents* that are initially placed by an adversary at arbitrary but distinct positions in the plane. Both agents have a *limited sensory radius* (in the sequel also referred to as *radius of vision*), the value of which is denoted by ϵ, allowing them to sense (or, to see) all their surroundings at distance at most ϵ from their respective current locations. We assume that the agents know the value of ϵ. As stated in [12], when $\epsilon = 0$, if agents start from arbitrary positions of the plane and can freely move on it, making them occupy the same location at the same time is impossible in a deterministic way. So, we assume that $\epsilon > 0$ and we consider the task of *approach* which consists in bringing them at distance at most ϵ so that they can see each other. In other words, the agents completed their approach once they mutually sense each other and they can even get closer. Without loss of generality, we assume in the rest of this paper that $\epsilon = 1$.

The initial positions of the agents, arbitrarily chosen by the adversary, are separated by a distance Δ that is initially unknown to both agents and that is greater than $\epsilon = 1$. In addition to the initial positions, the adversary also assigns a different non-negative integer (called label) to each agent. The label of an agent is the only input of the deterministic algorithm executed by the agent. While the labels are distinct, the algorithm is the same for both agents. Each agent is equipped with a compass showing the cardinal directions and with a unit of length. The cardinal directions and the unit of length are the same for both agents.

To describe how and where each agent moves, we need to introduce two important notions that are borrowed from [12]: The *route* and the *walk* of an agent. The *route* of an agent is a sequence $(S_1, S_2, S_3 \ldots)$ of segments $S_i = [a_i, a_{i+1}]$ traversed in stages as follows. The route starts from a_1, the initial position of the agent. For every $i \geq 1$, starting from the position a_i, the agent initiates Stage i by choosing a direction α using its compass as well as a distance x expressed in its own unit of length. Stage i ends as soon as the agent either sees the other agent or reaches a_{i+1} corresponding to the point at distance x from a_i in direction α. Stages are repeated indefinitely (until the approach is completed). Since both agents never know their positions in a global coordinate system, the directions they choose at each stage can only depend on their (deterministic) algorithm and their labels. So, the route (the actual sequence of segments) followed by an agent depends on its algorithm and its label, but also on its initial position. By contrast, the *walk* of each agent along every segment of its route is controlled by the adversary. More precisely, within each stage S_i and while the approach is not achieved, the adversary can arbitrarily vary the speed of the agent, stop it and even move it back and forth as long as the walk of the agent is continuous, does not leave S_i, and ends at a_{i+1}. Roughly speaking, the goal of the adversary is to prevent the agents from solving the task, or at least to ensure that the agents have covered as much distance as possible before seeing each other. We assume that at any time an agent can remember the route and the walk it has followed since the beginning.

A *deterministic approach algorithm* is a deterministic algorithm that always allows two agents to solve the task of approach regardless of the choices and the behavior of the adversary. The *cost* of an accomplished approach is the length of both parts of route travelled by the agents until they see each other. An approach algorithm is said to be *polynomial* in Δ and in the length of the binary representation of the shortest label between both agents if it always permits to solve the problem of approach at a cost that is polynomial in the two aforementioned parameters, no matter what the adversary does.

It is worth mentioning that the use of distinct labels is not fortuitous. In the absence of a way of distinguishing the agents, the task of approach would have no deterministic solution. This is especially the case if the adversary handles the agents in a perfect synchronous manner. Indeed, if the agents act synchronously and have the same label, they will always follow the same deterministic rules leading to a situation in which the agents will always be exactly at distance Δ from each other.

1.2 Our Results

In this paper, we prove that the task of approach can be solved deterministically in the above asynchronous model, at a cost that is polynomial in the unknown initial distance separating the agents and in the length of the binary representation of the shortest label. To obtain this result, we go through the design of a deterministic algorithm for a very close problem, that of rendezvous in an infinite oriented grid which consists in ensuring that both agents end up meeting either at a node or on an edge of the grid. The tasks of approach and rendezvous are very close as the former can be reduced to the latter.

It should be noticed that our result turns out to be an important advance, from a computational point of view, in resolving the task of approach. Indeed, the other existing algorithms allowing to solve the same problem either have an exponential cost in the initial separating distance and in the labels of the agents [12], or require each agent to know its starting position in a global system of coordinates [10], or only work under a much less powerful adversary [18] which initially assigns a possibly different speed to each agent but cannot vary it afterwards.

1.3 Related Work

The task of approach is closely linked to the task of rendezvous. Historically, the first mention of the rendezvous problem appeared in [33]. From this publication until now, the problem has been extensively studied so that there is henceforth a huge literature about this subject. This is mainly due to the fact that there is a lot of alternatives for the combinations we can make when addressing the problem, *e.g.,* playing on the environment in which the agents are supposed to evolve, the way of applying the sequences of instructions (*i.e.,* deterministic or randomized) or the ability to leave some traces in the visited locations, etc. Naturally, in this paper we focus on work that are related to deterministic rendezvous. This is why we will mostly dwell on this scenario in the rest of this subsection. However, for the curious reader wishing to consider the matter in greater depth, regarding randomized rendezvous, a good starting point is to go through [2, 3, 28]. Concerning deterministic rendezvous, the literature is divided according to the way of modeling the environnement: Agents can either move in a graph representing a network, or in the plane.

For the problem of rendezvous in networks, a lot of papers considered synchronous settings, *i.e.,* a context where the agents move in the graph in synchronous rounds. This is particularly the case of [17] in which the authors presented a deterministic protocol for solving the rendezvous problem, which guarantees a meeting of the two involved agents after a number of rounds that is polynomial in the size n of the graph, the length l of the shortest of the two labels and the time interval τ between their wake-up times. As an open problem, the authors asked whether it was possible to obtain a polynomial solution to this problem which would be independent of τ. A positive answer to this question was given, independently of each other, in [27] and [35]. While these algorithms ensure rendezvous in polynomial time (*i.e.,* a polynomial number of rounds), they also ensure it at polynomial cost because the cost of a rendezvous protocol in a graph is the number of edges traversed by the agents until they meet—each agent can make at most one edge traversal per round. Note that despite the fact a polynomial time implies a polynomial cost in this context, the reciprocal is not always true as the agents can have very long waiting periods, sometimes interrupted by a movement. Thus these parameters of cost and time are not always linked to each other. This was highlighted in [31] where the authors studied the tradeoffs between cost and time for the deterministic rendezvous problem. More recently, some efforts have been dedicated to analyse the impact on time complexity of rendezvous when in every round the agents are brought with some pieces of information by making a query to some device or some oracle [14, 30]. Along with the work aiming at optimizing the parameters of time and/or cost of rendezvous, some other work have examined the amount of required memory to solve the problem, *e.g.,* [24, 25] for tree networks and in [11] for general networks. In [6], the problem is approached in a fault-prone framework, in which the adversary can delay an agent for a finite number of rounds, each time it wants to traverse an edge of the network.

Rendezvous is the term that is usually used when the task of meeting is restricted to a team of exactly two agents. When considering a team of two agents or more, the term of gathering is commonly used. Still in the context of synchronous networks, we can cite some work about gathering two or more agents. In [19], the task of gathering is studied for anonymous agents while in [5, 15, 20] the same task is studied in presence of byzantine agents that are, roughly speaking, malicious agents with an arbitrary behavior.

Some studies have been also dedicated to the scenario in which the agents move asynchronously in a network [12, 21, 29], *i.e.,* assuming that the agent speed may vary, controlled by the adversary. In [29], the authors investigated the cost of rendezvous for both infinite and finite graphs. In the former case, the graph is reduced to the (infinite) line and bounds are given depending on whether the agents know the initial distance between them or not. In the latter case (finite graphs), similar bounds are given for ring shaped networks. They also proposed a rendezvous algorithm for an arbitrary graph provided the agents initially know an upper bound on the size of the graph. This assumption was subsequently removed in [12]. However, in both [29] and [12], the cost of rendezvous was exponential in the size of the graph. The first rendezvous algorithm working for arbitrary finite connected graphs at cost polynomial in the size of the graph and in the length of the shortest label was presented in [21]. (It should be stressed that the algorithm from [21] cannot be used to obtain the solution described in the present paper: this point is fully explained in the end of this subsection). In all the aforementioned studies, the agents can remember all the actions they have made since the beginning. A different asynchronous scenario for networks was studied in [13]. In this paper, the authors assumed that agents are oblivious, but they can observe the whole graph and make navigation decisions based on these observations.

Concerning rendezvous or gathering in the plane, we also found the same dichotomy of synchronicity *vs.* asynchronicity. The synchronous case was introduced in [34] and studied from a fault-tolerance point of view in [1, 16, 22]. In [26], rendezvous in the plane is studied for oblivious agents equipped with unreliable compasses under synchronous and asynchronous models. Asynchronous gathering of many agents in the plane has been studied in various settings in [7, 8, 9, 23, 32]. However, the common feature of all these papers related to rendezvous or gathering in the plane – which is not present in our model – is that the agents can observe all the positions of the other agents or at least the global graph of visibility is always connected (*i.e.,* the team cannot be split into two groups so that no agent of the first group can detect at least one agent of the second group).

Finally, the closest works to ours allowing to solve the problem of approach under an asynchronous framework are [10, 4, 12, 18]. In [10, 12, 18], the task of approach is solved by reducing it to the task of rendezvous in an infinite oriented grid. In [4], the authors present a solution to solve the task of approach in a multidimensional space by reducing it to the task of rendezvous in an infinite multidimensional grid. Let us give some more details concerning these four works to highlight the contrasts with our present contribution. The result from [12] leads to a solution to the problem of approach in the plane but has the disadvantage of having an exponential cost. The result from [10] and [4] also implies a solution to the problem of approach in the plane at cost polynomial in the initial distance of the agents. However, in both these works, the authors use the powerful assumption that each agent knows its starting position in a global system of coordinates (while in our paper, the agents are completely ignorant of where they are). Lastly, the result from [18] provides a solution at cost polynomial in the initial distance between agents and in the length of the shortest label. However, the authors of this study also used a powerful assumption: The adversary initially assigns a possibly different and arbitrary speed to each agent but cannot vary it afterwards. Hence, each agent moves at constant speed and uses clock to achieve approach. By contrast, in our paper, we assume basic asynchronous settings, *i.e.,* the adversary arbitrarily and permanently controls the speed of each agent.

To close this subsection, it is worth mentioning that it is unlikely that the algorithm from [21] that we referred to above, which is especially designed for asynchronous rendez-vous in arbitrary finite graphs, could be used to obtain our present result. First, in [21] the algorithm has not a cost polynomial in the initial distance separating the agents and in the length of the smaller label. Actually, ensuring rendezvous at this cost is even impossible in arbitrary graph, as witnessed by the case of the clique with two agents labeled 0 and 1: the adversary can hold one agent at a node and make the other agent traverse $\Theta(n)$ edges before rendezvous, in spite of the initial distance 1. Moreover, the validity of the algorithm given in [21] closely relies on the fact that both agents must evolve in the same finite graph, which is clearly not the case in our present scenario. In particular even when considering the task of rendezvous in an infinite oriented grid, the natural attempt consisting in making each agent to apply the algorithm from [21] within bounded grids of increasing size and centered in its initial position, does not permit to claim that rendezvous ends up occurring. Indeed, the bounded grid considered by an agent is never exactly the same than the bounded grid considered by the other one (although they may partly overlap), and thus the agents never evolve in the same finite graph which is a necessary condition to ensure the validity of the solution of [21] and by extension of this natural attempt.

1.4 Roadmap

The next section (Section 2) is dedicated to the computational model and basic definitions. We sketch our solution in Section 3, formally described in Sections 4 and 5. Section 6 presents the correctness proof and cost analysis of the algorithm. Finally, we make some concluding remarks in Section 7.

2 Preliminaries

We know from [12, 18] that the problem of approach in the plane can be reduced to that of rendezvous in an infinite grid specified in the next paragraph.

Consider an *infinite square grid* in which every node u is adjacent to 4 nodes located North, East, South, and West from node u. We call such a grid a *basic grid*. Two agents with distinct labels (corresponding to non-negative integers) starting from arbitrary and distinct nodes of a basic grid G have to meet either at some node or inside some edge of G. As for the problem of approach (in the plane), each agent is equipped with a compass showing the cardinal directions. The agents can see each other and communicate only when they share the same location in G. In other words, in the basic grid G we assume that the sensory radius (or, radius of vision) of the agents is equal to zero. In such settings, the only initial input that is given to a rendezvous algorithm is the label of the executing agent. When occupying a node u, an agent decides (according to its algorithm) to move to an adjacent node v via one of the four cardinal directions: the movement of the agent along the edge $\{u, v\}$ is controlled by the adversary in the same way as in a section of a route (refer to Subsection 1.1), *i.e.,* the adversary can arbitrarily vary the speed of the agent, stop it and even move it back and forth as long as the walk of the agent is continuous, does not leave the edge, and ends at v.

The *cost* of a rendezvous algorithm in a basic grid is the total number of edge traversals by both agents until their meeting.

From the reduction described in [18], we have the following theorem.

Theorem 1. *If there exists a deterministic algorithm solving the problem of rendezvous between any two agents in a basic grid at cost polynomial in D and in the length of the binary representation of the shortest of their labels where D is the distance (in the Manhattan metric) between the two starting nodes occupied by the agents, then there exists a deterministic algorithm solving the problem of approach in the plane between any two agents at cost polynomial in Δ and in the length of the binary representation of the shortest of their labels where Δ is the initial Euclidean distance separating the agents.*

For completeness let us now outline the reduction described in [18]. Consider an infinite square grid with edge length 1. More precisely, for any point v in the plane, we define the *basic grid* G_v to be the infinite graph, one of whose nodes is v, and in which every node u is adjacent to 4 nodes at Euclidean distance 1 from it, and located North, East, South, and West from node u. We now focus on how to transform any rendezvous algorithm in the grid G_v to an algorithm for the task of approach in the plane.

Let A be any rendezvous algorithm for any basic grid. Algorithm A can be executed in the grid G_w, for any point w in the plane. Consider two agents in the plane starting respectively from point v and from another point w in the plane. Let V' be the set of nodes in G_v that are the closest nodes from w. Let v' be a node in V', arbitrarily chosen. Notice that v' is at distance at most $\sqrt{2}/2 < 1$ from w. Let α be the vector $v'w$. Execute algorithm A on the grid G_v with agents starting at nodes v and v'. Let p be the point in G_v (either a node of it or a point inside an edge), in which these agents meet at some time t. The transformed algorithm A^* for approach in the plane works as follows: Execute the same algorithm A but with one agent starting at v and traveling in G_v and the other agent starting at w and traveling in G_w, so that the starting time of the agent starting at w is the same as the starting time of the agent starting at v' in the execution of A in G_v. The starting time of the agent starting at v does not change. If approach has not been accomplished before, in time t the agent starting at v and traveling in G_v will be at point p, as previously. In the same way, the agent starting at w and traveling in G_w will get to some point q at time t. Clearly, $q = p + \alpha$. Hence both agents will be at distance less than 1 at time t, which means that they accomplish approach in the plane because $\epsilon = 1$ (refer to Subsection 1.1).

Hence in the rest of the paper we will consider rendezvous in a basic grid, instead of the task of approach. We use N (resp. E, S, W) to denote the cardinal direction North (resp. East, South, West) and an instruction like "Perform NS" means that the agent traverses one edge to the North and then traverses one edge to the South (by the way, coming back to its initial position). We denote by D the initial (Manhattan) distance separating two agents in a basic grid. A route followed by an agent in a basic grid corresponds to a path in the grid (*i.e.,* a sequence of edges $e_1, e_2, e_3, e_4, \ldots$) that are consecutively traversed by the agent until rendezvous is done. For any integer k, we define the *reverse path* to the path $e_1, \ldots, e_k$ as the path $e_k, e_{k-1}, \ldots, e_1 = \overline{e_1, \ldots, e_{k-1}, e_k}$. We denote by $C(p)$ the number of edge traversals performed by an agent during the execution of a procedure p.

Consider two distinct nodes u and v. We define a specific path from u to v, denoted $P(u,v)$, as follows. If there exists a unique shortest path from u to v, this shorthest path is $P(u,v)$. Otherwise, consider the smallest rectangle $R_{(u,v)}$ such that u and v are two of its corners. $P(u,v)$ is the unique path among the shortest path from u to v that traverses all the edges on the northern side of $R_{(u,v)}$. Note that $P(u,v) = \overline{P(v,u)}$.

An illustration of $P(u,v)$ is given in Figure 1.

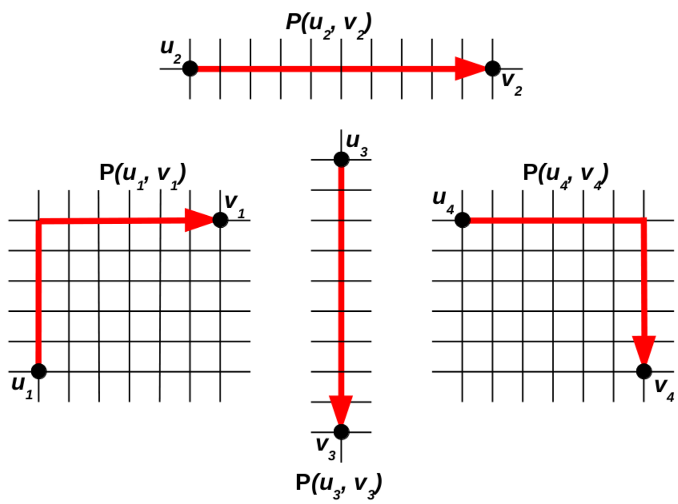

Figure 1: Some different cases for $P(u,v)$

3 Idea of the algorithm

3.1 Informal Description in a Nutshell...

We aim at achieving rendezvous of two asynchronous mobile agents in an infinite grid and in a deterministic way. It is well known that solving rendezvous deterministically is impossible in some symmetric graphs (like a basic grid) unless both agents are given distinct identifiers called labels. We use them to break the symmetry, *i.e.,* in our context, to make the agents follow different routes. The idea is to make each agent "read" its label binary representation, a bit after another from the most to the least significant bits, and for each bit it reads, follow a route depending on the read bit. Our algorithm ensures rendezvous during some of the periods when they follow different routes *i.e.,* when the two agents process two different bits.

Furthermore, to design the routes that both agents will follow, our approach would require to know an upper bound on two parameters, namely the initial distance between the agents and the length (of the binary representation) of the shortest label. As we suppose that the agents have no knowledge of these parameters, they both perform successive "assumptions", in the sequel called *phases*, in order to find out such an upper bound. Roughly speaking, each agent attempts to estimate such an upper bound by successive tests, and for each of these tests acting as if the upper bound estimation was correct. Both agents first perform Phase 0. When Phase i does not lead to rendezvous, they perform Phase $i+1$, and so on. More precisely, within Phase i, the route of each agent is built in such a way that it ensures rendezvous if 2^i is a good upper bound on the parameters of the problem. Hence, in our approach two requirements are needed: both agents are assumed (1) to process two different bits (*i.e.,* 0 and 1) almost concurrently and (2) to perform Phase $i = \alpha$ almost at the same time—where α is the smallest integer such that the two aforementioned parameters are upper bounded by 2^α.

However, to meet these requirements, we have to face two major issues. First, since the adversary can vary both agent speeds, the idea described above does not prevent the adversary from making the agents always process the same type of bit at the same time. Besides, the route cost depends on the phase number, and thus, if an agent were performing some Phase i with i exponential in the initial distance and in the length of the binary representation of the smallest label, then our algorithm would not be polynomial. To tackle these two issues, we use a mechanism that prevents the adversary from making an agent execute the algorithm arbitrarily faster than the other without meeting. Each of both these issues is circumvent via a specific "synchronization mechanism". Roughly speaking, the first one makes the agents read and process the bits of the binary representation of their labels at quite the same speed, while the second ensures that they start Phase α at almost the same time. This is particularly where our feat of strength is: orchestrating in a subtle manner these synchronizations in a fully asynchronous context while ensuring a polynomial cost. Now that we have described the very high level idea of our algorithm, let us give more details.

3.2 Under the hood

The approach described above allows us to solve rendezvous when there exists an index for which the binary representations of both labels differ. However, this is not always the case especially when a binary representation is a prefix of the other one (e.g., 100 and 1000). Hence, instead of considering its own label, each agent will consider a transformed label: The transformation borrowed from [17] will guarantee the existence of the desired difference over the new labels. In the rest of this description, we assume for convenience that the initial Manhattan distance D separating the agents is at least the length of the shortest binary representation of the two transformed labels (the complementary case adds an unnecessary level of complexity to understand the intuition).

As mentioned previously, our solution (`cf. Algorithm 5 in Section 5`) works in phases numbered $0, 1, 2, 3, 4, \ldots$ During Phase i (`cf. Procedure` *Assumption* `called at line 11 in Algorithm 5`), the agent supposes that the initial distance D is at most 2^i and processes one by one the first 2^i bits of its transformed label: In the case where 2^i is greater than the binary representation of its transformed label, the agent will consider that each of the last "missing" bits is 0. When processing a bit, the agent executes a particular route which depends on the bit value and the phase number. The route related to bit 0 (`cf. Procedure` *Berry* `called at line 8 in Algorithm 6`) and the route related to bit 1 (`cf. Procedure` *Cloudberry* `called at line 10 in Algorithm 6`) are obviously different and designed in such a way that if both these routes are executed almost simultaneously by two agents within a phase corresponding to a correct upper bound, then rendezvous occurs by the time any of them has been completed. In the light of this, if we denote by α the smallest integer such that $2^{\alpha} \geq D$, it turns out that an ideal situation would be that the agents concurrently start phase α and process the bits at quite the same rate within this phase. Indeed, we would then obtain the occurrence of rendezvous by the time the agents complete the process of the j-th bit of their transformed label in phase α, where j is the smallest index for which the binary representations of their transformed labels differ. However, getting such an ideal situation in presence of a fully asynchronous adversary appears to be really challenging. This is where the two synchronization mechanisms briefly mentioned above come into the picture.

If the agents start Phase α approximately at the same time, the first synchronization mechanism (`cf. Procedure` *RepeatSeed* `called at line 13 in Algorithm 6`) permits to force the adversary to make the agents process their respective bits at similar speed within Phase α, as otherwise rendezvous would occur prematurely during this phase before the process by any agent of the j-th bit. This constraint is imposed on the adversary by ensuring that after the process of the k-th bit, for any $k \leq 2^{\alpha}$, an agent follows a specific route that forces the other agent to complete the process of its k-th bit. This route, on which the first synchronization is based, is constructed by relying on the following simple principle: If an agent performs a given route X included in a given area $\mathcal{S}$ of the basic grid, then the other agent can "*push* it" over X. In other words, unless rendezvous occurs, the agent forces the other to complete its route X by covering $\mathcal{S}$ a number of times at least equal to the number of edge traversals involved in route X (each covering of $\mathcal{S}$ allows to traverse all the edges of $\mathcal{S}$ at least once). Hence, one of the major difficulties we have to face lies in the setting up of the second synchronization mechanism guaranteeing that the agents start Phase α around the same time. At first glance, it might be tempting to use an analogous principle to the one used for dealing with the first synchronization. Indeed, if an agent a_1 follows a route covering r times an area $\mathcal{Y}$ of the grid, such that $\mathcal{Y}$ is where the first $\alpha - 1$ phases of an agent a_2 take place and r is the maximal number of edge traversals an agent can make during these phases, then agent a_1 pushes agent a_2 to complete its first $\alpha - 1$ phases and to start Phase α. Nevertheless, a strict application of this principle to the case of the second synchronization directly leads to an algorithm having a cost that is superpolynomial in D and the length of the smallest label, due to a cumulative effect that does not appear for the case of the first synchronization. As a consequence, to force an agent to start its Phase α, the second synchronization mechanism does not depend on the kind of route described above, but on a much more complicated route that permits an agent to "*push*" the second one. This works by considering the "pattern" that is drawn on the grid by the second agent rather than just the number of edges that are traversed (`cf. Procedure` *Harvest* `called at line 1 in Algorithm 6`). This is the most tricky part of our algorithm, the main idea of which relies on the fact that some routes made of an arbitrarily large sequence of edge traversals can be pushed at a relative low cost by some other routes that are of comparatively small length, provided they are judiciously chosen. Let us illustrate this point through the following example. Consider an agent a_1 following from a node v_1 an arbitrarily large sequence of X_i, in which each X_i corresponds either to $A\overline{A}$ or $B\overline{B}$ where A and B are any routes ($\overline{A}$ and $\overline{B}$ corresponding to their respective backtrack *i.e.,* the sequence of edge traversals followed in the reverse order). An agent a_2 starting from an initial node v_2 located at a distance at most d from v_1 can force agent a_1 to finish its sequence of X_i (or otherwise rendezvous occurs), regardless of the number of X_i, simply by executing $A\overline{A}B\overline{B}$ from each node at distance at most d from v_2. To support this claim, let us suppose by contradiction that it does not hold. At some point, agent a_2 necessarily follows $A\overline{A}B\overline{B}$ from v_1. However, note that if either agent starts following $A\overline{A}$ (resp. $B\overline{B}$) from node v_1 while the other is following $A\overline{A}$ (resp. $B\overline{B}$) from node v_1, then the agents meet. Indeed, this implies that the more ahead agent eventually follows $\overline{A}$ (resp. $\overline{B}$) from a node v_3 to v_1 while the other is following A (resp. B) from v_1 to v_3, which leads to rendezvous. Hence, when agent a_2 starts following $B\overline{B}$ from node v_1, agent a_1 is following $A\overline{A}$, and is not in v_1, so that it has at least started the first edge traversal of $A\overline{A}$. This means that when agent a_2 finishes following $A\overline{A}$ from v_1, a_1 is following $A\overline{A}$, which implies, using the same arguments as before, that they meet before either of them completes this route. Hence, in this example, agent a_2 can force a_1 to complete an arbitrarily large sequence of edge traversals with a single and simple route. Actually, our second synchronization mechanism uses this idea. Roughly speaking, to make them pushed by the second synchronization mechanism at low cost, the $\alpha - 1$ first phases are designed in such a way that large parts of them can be pushed at low cost in a similar manner as the route followed by agent a_1 in the above example. This was way the most complicated to set up, as each part of each route in every phase had to be orchestrated very carefully to permit this synchronization while still ensuring rendezvous. However, it is through this original and novel way of moving that we finally get the polynomial cost.

4 Basic patterns

In this section we define some sequences of moving instructions, *i.e.,* patterns of moves, that will serve in turn as building blocks in the construction of our rendezvous algorithm.

4.1 Pattern *Seed*

Pattern *Seed* is involved as a subpattern in the design of all the other patterns presented in this section. The description of Pattern *Seed* is given in Algorithm 1. It is made of two periods. For a given non-negative integer x, the first period of Pattern $Seed(x)$ corresponds to the execution of x phases, while the second period is a complete backtrack of the path travelled during the first period. Pattern *Seed* is designed in such a way that it offers some properties that are shown in Subsubsection 6.1.2 and that are necessary to conduct the proof of correctness. In particular, starting from a node v, Pattern $Seed(x)$ allows to visit all nodes of the grid at distance at most x from v and to traverse all edges of the grid linking two nodes at distance at most x from v.

Algorithm 1 Pattern $Seed(x)$

```
/* First period */
for i ← 1; i ≤ x; i ← i + 1 do
    /* Phase i */
    Perform (N(SE)^i(WS)^i(NW)^i(EN)^i)
end for
/* Second period */
L ← the path followed by the agent during the first period
Backtrack by following the reverse path L̄
```

4.2 Pattern *RepeatSeed*

Following the high level description of our solution (Section 3), Pattern *RepeatSeed* is the basic primitive procedure that implements the first synchronizations mechanism (between two consecutive bit processes). An agent a_1 executing pattern $RepeatSeed(x, n)$ from a node u processes n times pattern $Seed(x)$ from node u. All along this execution, a_1 stays at distance at most x from u. Besides, once the execution is over, the agent is back at u.

The description of pattern *RepeatSeed* is given in Algorithm 2.

Algorithm 2 Pattern $RepeatSeed(x, n)$

```
Execute n times Pattern Seed(x)
```

4.3 Pattern *Berry*

According to Section 3, Pattern *Berry* is used in particular to design the specific route that an agent follows when processing bit 0. The description of Pattern *Berry* is given in Algorithm 3. It is made of two periods, the second of which is a backtrack of the first one. Pattern *Berry* offers several properties that are proved in Subsubsection 6.1.4 and used in the proof of correctness. Among those properties, we can mention the following. Pattern $Berry(x, y)$ executed from a node u for any two integers x and y allows an agent to perform Pattern $Seed(x)$ from each node at distance at most y from u.

Algorithm 3 Pattern $Berry(x, y)$

```
/* First period */
Let u be the current node
for i ← 1; i ≤ x + y; i ← i + 1 do
   for j ← 0; j ≤ i; j ← j + 1 do
      for k ← 0; k ≤ j; k ← k + 1 do
         for each node v at distance k from u ordered in the clockwise direction from the North
         do
            Follow P(u, v)
            Execute Seed(i − j)
            Follow P(v, u)
         end for
      end for
   end for
end for
/* Second period */
L ← the path followed by the agent during the first period
Backtrack by following the reverse path L̄
```

4.4 Pattern *Cloudberry*

Algorithm 4 describes Pattern *Cloudberry*. According to Section 3, Pattern *Cloudberry* is used to design the specific route that an agent follows when processing bit 1. The description of Pattern *Cloudberry* is given in Algorithm 4. As for Patterns *Seed* and *Berry*, the pattern is made of two periods, the second of which corresponds to a backtrack of the first one. Properties related to this pattern are given in Subsubsection 6.1.5. In particular, we can mention the following. Pattern

$Cloudberry(x, y, z, h)$ executed from a node u for any integers x, y, z and h allows an agent to perform Pattern $Berry(x, y)$ from each node at distance at most z from u.

Algorithm 4 Pattern $Cloudberry(x, y, z, h)$

/* First period */
Let u be the current node
Let U be the list of nodes at distance at most z from u ordered in the order of the first visit when applying $Seed(z)$ from node u
for $i \leftarrow 0$; $i \leq 2z(z+1)$; $i \leftarrow i+1$ **do**
  Let v be the node with index $h + i \pmod{2z(z+1)+1}$ in U
  Follow $P(u, v)$
  Execute $Seed(x)$
  Execute $Berry(x, y)$
  Follow $P(v, u)$
end for
/* Second period */
$L \leftarrow$ the path followed by the agent during the first period
Backtrack by following the reverse path $\overline{L}$

5 Main Algorithm

In this section, we give the formal description of Algorithm RV (refer to Algorithm 5) allowing to solve rendezvous in a basic grid. As mentioned in Subsection 3.2, we use the label of an agent only when it has been transformed. Let us describe this transformation that is borrowed from [17]. Let $(b_0 b_1 \dots b_{n-1})$ be the binary representation of the label of an agent. We define its transformed label as the binary sequence $(b_0 b_0 b_1 b_1 \dots b_{n-1} b_{n-1} 01)$. This transformation permits to obtain the feature that is highlighted by the following remark.

Remark 2. *Given two distinct labels l_a and l_b, their transformed labels are never prefixes of each other. In other words, there exists an index j such that the j-th bit of the transformed label of l_a is different from the j-th bit of the transformed label of l_b.*

As explained in Section 3, we need such a feature because our solution requires that at some point both agents follow different routes by processing different bit values.

Algorithm RV makes use of a subroutine, *i.e.,* procedure *Assumption*, which in turn also makes use of several other subroutines relying on the basic patterns presented in the previous section. The purpose of the rest of this section is to give the formal description of these subroutines.

The codes of Procedures *Assumption*, *Harvest*, and *PushPattern* are respectively given by Algorithm 6, Algorithm 7, and Algorithm 8. According to Section 3, Algorithm *Assumption* called with some parameter 2^i corresponds to phase i in which an agent supposes that $2^i \geq D$ and acts as if 2^i was a correct upper bound on D. Algorithm *Harvest* corresponds to the second synchronization mechanism mentioned in Subsection 3.2, while Algorithm *PushPattern* is a subroutine of the former one allowing to push an agent under some conditions (or otherwise rendezvous occurs).

To introduce the formal descriptions of Algorithms 6 and 7, we need to define two sequences that will be used in the instructions of both these algorithms:

Algorithm 5 RV

Let *Label* be the label of the agent represented as an array of bits indexed from 0, and n its length
Let $TransformedLabel$ be an array of length $2n+2$ indexed from 0
for each bit b_i of *Label* **do**
  $TransformedLabel[2i] = b_i$
  $TransformedLabel[2i+1] = b_i$
end for
$TransformedLabel[2n] = 0$
$TransformedLabel[2n+1] = 1$
$d \leftarrow 1$
while agents have not met yet **do**
  Execute $Assumption(d)$
  $d \leftarrow 2d$
end while

$$\rho(1) = 1 \text{ and } \forall \text{ power of two } i \geq 2,\ \rho(i) = r(\frac{i}{2}) + \frac{3i}{2}(\frac{i}{2}(i(\frac{i}{2}+1)+1)+1)$$
$$\forall \text{ power of two } i,\ r(i) = \rho(i) + 3i$$

Algorithm 6 $Assumption(d)$

Execute $Harvest(d)$
$radius \leftarrow r(d)$
$i \leftarrow 0$
while $i < d$ **do**
  $j \leftarrow 0$
  while $j \leq 2d(d+1)$ **do**
    if $TransformedLabel[i] = 0$ or i is at least the length of $TransformedLabel$ **then**
      Execute $Berry(radius, d)$
    else
      Execute $Cloudberry(radius, d, d, j)$
    end if
    $radius \leftarrow radius + 3d$
    Execute $RepeatSeed(radius, C(Cloudberry(radius - 3d, d, d, j)))$
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while

To introduce Algorithm 8, we need the following definitions of *basic decomposition* and *perfect decomposition*.

Definition 3 (Basic decomposition & Perfect decomposition)**.** *Given a call P to an algorithm, we say that the basic decomposition of P, denoted $\mathcal{BD}(P)$, is P itself if P corresponds to a basic pattern,*

Algorithm 7 *Harvest*(d)

for $i \leftarrow 1$; $i < d$; $i \leftarrow 2i$ **do**
  Execute $PushPattern(i, d)$
end for
Execute $Cloudberry(\rho(d), d, d, 0)$
Execute $RepeatSeed(r(d), C(Cloudberry(\rho(d), d, d, 0)))$

the type of which belongs to $\{RepeatSeed; Berry; Cloudberry\}$*. Otherwise, if during its execution* P *makes no call then* $\mathcal{BD}(P) = \perp$*, else* $\mathcal{BD}(P) = \mathcal{BD}(x_1), \mathcal{BD}(x_2), \ldots, \mathcal{BD}(x_n)$ *where* $x_1, x_2, \ldots, x_n$ *is the sequence (in the order of execution) of all the calls in* P *that are children of* P*. We say that* $\mathcal{BD}(P)$ *is a perfect decomposition if it does not contain any* $\perp$*.*

Remark 4. *The basic decomposition of every call to procedure Assumption is perfect.*

Algorithm 8 *PushPattern*(i, d)

for each p in $\mathcal{BD}(Assumption(i))$ **do**
  if p is a call to pattern *RepeatSeed* with value x as first parameter **then**
    Execute $Berry(x, d)$
  else
    /* pattern p is either a call to pattern *Berry* or a call to pattern *Cloudberry* (in view of Remark 4) and has at least two parameters */
    Let x (resp. y) be the first (resp. the second) parameter of p
    Execute $RepeatSeed(d + x + 2y, C(Cloudberry(x, y, y, 0)))$
  end if
end for

6 Proof of correctness and cost analysis

The purpose of this section is to prove that Algorithm RV ensures rendezvous in the basic grid at cost polynomial in D (the initial distance between the agents), and l, the length of the shortest label. To this end, the section is made of four subsections. The first two subsections are dedicated to technical results about the basic patterns presented in Section 4 and synchronization properties of Algorithm RV, which are used in turn to carry out the proof of correctness and the cost analysis of Algorithm RV. The last two subsections are devoted to the proof of correctness and polynomial complexity of Algorithm RV.

6.1 Properties of the basic patterns

This subsection is dedicated to the presentation of some technical materials about the basic patterns described in Section 4, which will be used in the proof of correctness of Algorithm 5 solving rendezvous in a basic grid.

6.1.1 Vocabulary

Before going any further, we need to introduce some extra vocabulary in order to facilitate the presentation of the next properties and lemmas.

Definition 5. *A pattern execution A precedes another pattern execution B if the beginning of A occurs before the beginning of B.*

Definition 6. *Two pattern executions A and B are concurrent iff:*

- *pattern execution A does not finish before pattern execution B starts*
- *pattern execution B does not finish before pattern execution A starts*

By misuse of langage, in the rest of this paper we will sometimes say "a pattern" instead of "a pattern execution".

Hereafter we say that a pattern A *concurrently precedes* a pattern B, iff A and B are concurrent, and A precedes B.

Definition 7. *A pattern A pushes a pattern B in a set of executions E, if for every execution of E in which B concurrently precedes A, agents meet before the end of the execution of B, or B finishes before A.*

In the sequel, given two sequences of moving instructions X and Y, we will say that X is a prefix of Y if Y can be viewed as the execution of the sequence X followed by another sequence possibly empty.

6.1.2 Pattern *Seed*

In this subsubsection, we show some properties related to Pattern *Seed*.

Proposition 8 follows by induction on the input parameter of Pattern *Seed* and Proposition 9 follows from the description of Algorithm 1.

Proposition 8. *Let x be an integer. Starting from a node v, Pattern Seed(x) guarantees the following properties:*

1. *it allows to visit all nodes of the grid at distance at most x from v*
2. *it allows to traverse all edges of the grid linking two nodes at distance at most x from v*

Proposition 9. *Given two integers $x_1 \leq x_2$, the first period of Pattern Seed(x_1) is a prefix of the first period of Pattern Seed(x_2).*

Lemma 10. *Let x_1 and x_2 be two integers such that $x_1 \leq x_2$. Let a_1 and a_2 be two agents executing respectively Patterns Seed(x_1) and Seed(x_2) both from the same node such that the execution of Pattern Seed(x_1) concurrently precedes the execution of Pattern Seed(x_2). Let t_1 (resp. t_2) be the time when agent a_1 (resp. a_2) completes the execution of Pattern Seed(x_1) (resp. Seed(x_2)). Agents a_1 and a_2 meet by time $min(t_1, t_2)$.*

Proof. Consider a node u and a first agent a_1 executing Pattern $\mathit{Seed}(x_1)$ from u with x_1 any integer. Suppose that the execution of $\mathit{Seed}(x_1)$ by a_1 concurrently precedes the execution of Pattern $\mathit{Seed}(x_2)$ by another agent a_2 still from node u with $x_1 \leq x_2$.

According to Proposition 9, the first period of $\mathit{Seed}(x_1)$ is a prefix of the first period of Pattern $\mathit{Seed}(x_2)$. If the path followed by agent a_1 during its execution of $\mathit{Seed}(x_1)$ is $e_1, e_2, \ldots, e_n$, $\overline{e_1, e_2, \ldots, e_n}$ (the overlined part of the path corresponds to the backtrack), then the path followed by agent a_2 during the execution of Pattern $\mathit{Seed}(x_2)$ is $e_1, e_2, \ldots, e_n, s, \overline{e_1, e_2, \ldots, e_n, s}$ where s corresponds to the edges traversed at a distance $\in \{x_1 + 1; \ldots; x_2\}$. When a_2 starts executing the path $e_1, e_2, \ldots, e_n$, a_1 is on the path $e_1, e_2, \ldots, e_n, \overline{e_1, e_2, \ldots, e_n}$. Thus, either a_2 catches a_1 when the latter is following $e_1, e_2, \ldots, e_n$, or they meet while a_1 follows $\overline{e_1, e_2, \ldots, e_n}$.

Thus, if the execution of $\mathit{Seed}(x_1)$ by a_1 concurrently precedes the execution of $\mathit{Seed}(x_2)$ by agent a_2 both executed from the same node, agents meet by the end of these executions. □

6.1.3 Pattern *RepeatSeed*

This subsubsection is dedicated to some properties of Pattern *RepeatSeed*. Informally speaking, Lemmas 11 and 12 describe the fact that Pattern *RepeatSeed* pushes respectively Pattern *Berry* and *Cloudberry* when it is given appropriate parameters.

Lemma 11. *Consider two nodes u and v separated by a distance δ. If Pattern $\mathit{Berry}(x_1, y)$ is executed from node v and Pattern $\mathit{RepeatSeed}(x_2, n)$ is executed from node u with x_1, x_2, y and n integers such that $x_2 \geq x_1 + y + \delta$ and $n \geq C(\mathit{Berry}(x_1, y))$ then Pattern $\mathit{RepeatSeed}(x_2, x)$ pushes Pattern $\mathit{Berry}(x_1, y)$.*

Proof. Assume that, in the grid, there are two agents a_1 and a_2. Denote by u and v their respective initial positions. Suppose that u and v are separated by a distance δ. Assume that agent a_1 starts executing Pattern $\mathit{RepeatSeed}(x_2, n)$ from node u and agent a_2 performs Pattern $\mathit{Berry}(x_1, y)$ on node v (with $n \geq C(\mathit{Berry}(x_1, y))$ and $x_2 \geq x_1 + y + \delta$). Also suppose that Pattern $\mathit{Berry}(x_1, y)$ concurrently precedes Pattern $\mathit{RepeatSeed}(x_2, n)$. Let us suppose by contradiction, that $\mathit{RepeatSeed}(x_2, n)$ does not push $\mathit{Berry}(x_1, y)$, which means, by Definition 7 that at the end of the execution of $\mathit{RepeatSeed}(x_2, n)$ by a_1, agents have not met and a_2 has not finished executing its $\mathit{Berry}(x_1, y)$.

When executing its $\mathit{Berry}(x_1, y)$ agent a_2 can not be at a distance greater than $x_1 + y$ from its initial position, and can not be at a distance greater than $\delta + x_1 + y$ from node u. Besides, in view of Proposition 8, each Pattern $\mathit{Seed}(x_2)$ (which composes Pattern $\mathit{RepeatSeed}(x_2, n)$) from node u allows to visit all nodes and to traverse all edges at distance at most x_2 from node u. Thus, each Pattern $\mathit{Seed}(x_2)$ executed from node u allows to visit all nodes and to traverse all edges (although not necessarily in the same order) that are traversed during the execution of Pattern $\mathit{Berry}(x_1, y)$ from node v.

Consider the position of agent a_2 when a_1 starts executing any of the $\mathit{Seed}(x_2)$ which compose Pattern $\mathit{RepeatSeed}(x_2, n)$, and when a_1 has completed it. If a_2 has not completed a single edge traversal, then whether it was in a node or traversing an edge, it has met a_1 which traverses every edge a_2 traverses during its execution of Pattern $\mathit{Berry}(x_1, y)$. As this contradicts our hypothesis, each time a_1 completes one of its executions of Pattern $\mathit{Seed}(x_2)$, a_2 has completed at least an edge traversal. As agent a_1 executes $n \geq C(\mathit{Berry}(x_1, y))$ times Pattern $\mathit{Seed}(x_2)$ then a_2 traverses at least $C(\mathit{Berry}(x_1, y))$ edges before a_1 finishes executing its $\mathit{RepeatSeed}(x_2, n)$. As $C(\mathit{Berry}(x_1, y))$ is the number of edge traversals in $\mathit{Berry}(x_1, y)$, when a_1 finishes executing Pattern $\mathit{RepeatSeed}(x_2, n)$,

a_2 has finished executing its Pattern $Berry(x_1, y)$, which contradicts our assumption and proves the lemma. □

Lemma 12. *Consider two nodes u and v separated by a distance δ. If Pattern Cloudberry(x_1, y, z, h) is executed from node v (with x_1, y, z and h integers) and Pattern RepeatSeed(x_2, n) is executed from u such that $x_2 \geq x_1 + y + z + \delta$ and $n \geq C(Cloudberry(x_1, y, z, h))$ then Pattern RepeatSeed(x_2, n) pushes Pattern Cloudberry(x_1, y, z, h).*

Proof. Using similar arguments to those used in the proof of Lemma 11, we can prove Lemma 12. □

6.1.4 Pattern *Berry*

This subsubsection is dedicated to the properties of Pattern *Berry*. Informally speaking, Lemma 14 describes the fact that Pattern *Berry* permits to push Pattern *RepeatSeed* when it is given appropriate parameters. Proposition 13 and Lemma 15 are respectively analogous to Proposition 9 and Lemma 10.

According to Algorithm 3, we have the following proposition.

Proposition 13. *Given four integers $x_1 + y_1 \leq x_2 + y_2$, the first period of Pattern Berry(x_1, y_1) is a prefix of the first period of Pattern Berry(x_2, y_2).*

Lemma 14. *Consider two nodes u and v separated by a distance δ. Let RepeatSeed(x_1, n) and Berry(x_2, y) be two patterns respectively executed from nodes u and v with x_1, x_2, y and n integers. If $y \geq \delta$ and $x_1 \leq x_2$ then Pattern Berry(x_2, y) pushes Pattern RepeatSeed(x_1, n).*

Proof. Assume that there are two agents a_1 and a_2 initially separated by a distance δ. Assume that their respective initial positions are node u and node v. Agent a_2 executes Pattern $RepeatSeed(x_1, n)$ centered on v with x_1 and n any integers. This execution of Pattern $RepeatSeed(x_1, n)$ concurrently precedes the execution of Pattern $Berry(x_2, y)$ by a_1 with $y \geq \delta$ and $x_1 \leq x_2$. When executing this Pattern, agent a_1 performs Pattern $Seed(x_2)$ from each node at distance at most y from u with $y \geq \delta$. So, at some point a_1 executes Pattern $Seed(x_2)$ centered on node v. Since $x_2 \geq x_1$, by Lemma 10, if a_2 has not finished executing its $RepeatSeed(x_1, n)$ when a_1 starts executing Pattern $Seed(x_2)$ from v, then agents meet by the end of the latter.

Hence, to avoid rendezvous the adversary must choose an execution in which the speed of agent a_2 is such that it completes all executions of Patterns $Seed(x_1)$ inside $RepeatSeed(x_1, n)$ before a_1 starts the execution of Pattern $Seed(x_2)$ centered on v.

Let t_1 (resp. t_2) be the time when agent a_1 (resp. a_2) completes its execution of Pattern $Berry(x_2, y)$ (resp. $RepeatSeed(x_1, n)$). Thus, if the execution of Pattern $RepeatSeed(x_1, n)$ by a_2 concurrently precedes the execution of Pattern $Berry(x_2, y)$ by agent a_1, either $t_2 \leq t_1$ or the agents meet by time $min(t_1, t_2)$. □

Lemma 15. *Consider two agents a_1 and a_2 executing respectively Patterns Berry(x_1, y_1) and Berry(x_2, y_2) both from node u with x_1, x_2, y_1 and y_2 integers such that $x_2 + y_2 \geq x_1 + y_1$. Suppose that the execution of Berry(x_1, y_1) by a_1 concurrently precedes the execution of Berry(x_2, y_2) by a_2. Let t_1 (resp. t_2) be the time when agent a_1 (resp. a_2) completes its execution of Pattern Berry(x_1, y_1) (resp. Berry(x_2, y_2)). Agents a_1 and a_2 meet by time $min(t_1, t_2)$.*

Proof. Consider a node u and a first agent a_1 executing Pattern $Berry(x_1, y_1)$ from u with x_1 and y_1 two integers. Suppose that the execution of Pattern $Berry(x_1, y_1)$ by a_1 concurrently precedes an execution of Pattern $Berry(x_2, y_2)$ by another agent a_2 still from node u with $x_2 + y_2 \geq x_1 + y_1$.

This proof is similar to the proof of Lemma 10. According to Proposition 13, if the path followed by agent a_1 during its execution of $Berry(x_1, y_1)$ is $e_1, e_2, \ldots, e_n, \overline{e_1, e_2, \ldots, e_n}$ (the overlined part of the path corresponds to the backtrack), then the path followed by agent a_2 during the execution of Pattern $Berry(x_2, y_2)$ is $e_1, e_2, \ldots, e_n, s, \overline{e_1, e_2, \ldots, e_n, s}$ where s corresponds to the edges traversed from the $x_1 + y_1 + 1$-th iteration of the main loop of Pattern $Berry$ to its $x_2 + y_2$-th iteration. Thus, either a_2 catches a_1 when the latter is following $e_1, e_2, \ldots, e_n$, or they meet while a_2 follows $\overline{e_1, e_2, \ldots, e_n}$.

Let t_1 (resp. t_2) be the time when agent a_1 (resp. a_2) completes its execution of Pattern $Berry(x_1, y_1)$ (resp. $Berry(x_2, y_2)$). In the same way as in the proof of Lemma 10, if the execution of $Berry(x_1, y_1)$ by a_1 concurrently precedes the execution of $Berry(x_2, y_2)$ by agent a_1 both executed from the same node, the agents meet by time $min(t_1, t_2)$. □

6.1.5 Pattern *Cloudberry*

Informally speaking, the following lemma highlights the fact that Pattern *Cloudberry* can push "a lot of basic patterns" under some conditions. In other words, we can force an agent to make a lot of edge traversals "at relative low cost".

Lemma 16. *Consider two nodes u and v separated by a distance δ. Consider a sequence S of Patterns RepeatSeed and Berry executed from u, and a Pattern Cloudberry(x, y, z, h) executed from v (with x, y, z and h four integers) such that $z \geq \delta$ and the execution of S concurrently precedes the execution of Pattern Cloudberry(x, y, z, h). If for each Pattern RepeatSeed R and Pattern Berry B belonging to S, $x + y$ is greater than or equal to the sum of the parameters of B, and x is greater than or equal to the first parameter of R, then the execution of Pattern Cloudberry(x_1, y_1, z, h) from v pushes S.*

Proof. Let a_2 be an agent executing a sequence S of Patterns *RepeatSeed* and *Berry* from a node u. Suppose that there exist two integers x_1 and y_1 such that each Pattern *Berry* B inside the sequence is assigned parameters the sum of which is at most $x_1 + y_1$, and such that each Pattern *RepeatSeed* R of the sequence is assigned a first parameter which is at most x_1. Let v be another node separated from u by a distance δ. Suppose that another agent a_1 executes Pattern $Cloudberry(x_1, y_1, z, h)$ from v with $z \geq \delta$ and h two integers.

In order to prove that the execution of Pattern $Cloudberry(x_1, y_1, z, h)$ by a_1 pushes the sequence of Patterns S, let us suppose by contradiction that there exists an execution in which S concurrently precedes $Cloudberry(x_1, y_1, z, h)$, and that by the end of the execution of $Cloudberry(x_1, y_2, z, h)$ by a_1, a_2 neither has met a_1 nor has finished executing its whole sequence of patterns.

According to Algorithm *Cloudberry*, when executing $Cloudberry(x_1, y_1, z, h)$, a_1 executes Pattern $Seed(x)$ followed by Pattern $Berry(x, y)$ on each node at distance at most z from v. As $z \geq \delta$, during its execution of $Cloudberry(x_1, y_1, z, h)$, a_1 follows $P(v, u)$, executes Pattern $Seed(x_1)$ (denoted by p_1) and then Pattern $Berry(x_1, y_1)$ (denoted by p_2) both from node u. In order to prove that the execution of $Cloudberry(x_1, y_1, z, h)$ by a_1 pushes the execution of S by a_2, we are going to prove that if a_2 has not finished executing S when a_1 starts executing p_1 and p_2, agents meet. This will imply that the adversary has to make a_2 complete S before a_1 starts executing p_1 and p_2 in order to prevent the agents from meeting, and will thus prove the lemma.

By assumption, a_2 has not finished executing S when a_1 arrives on u to execute p_1 and p_2. Let us consider what it can be executing at this moment. If it is executing Pattern $Seed(x_2)$ with x_2 any integer, then by assumption, $x_2 \leq x_1$ and by Lemma 10, agents meet by the end of the execution of p_1, which contradicts the assumption that agents do not meet by the end of $Cloudberry(x_1, y_1, z, h)$.

It means that when a_1 starts executing p_1, a_2 is executing Pattern $Berry(x_2, y_2)$ for any integers x_2 and y_2 such that $x_2 + y_2 \leq x_1 + y_1$. After p_1, a_1 executes p_2. By Lemma 15, if a_2 is still executing Pattern $Berry(x_2, y_2)$ for any integers x_2 and y_2 such that $x_2 + y_2 \leq x_1 + y_1$ (the same as above, or another) then the agents meet by the end of the execution of p_2 which contradicts our assumption once again. As a consequence, when a_1 starts executing p_2, a_2 is executing Pattern $Seed(x_3)$ for an integer $x_3 \leq x_1$. Denote by p_3 this pattern, and remember that a_2 can not have started it before a_1 starts executing p_1. Moreover, when a_1 starts executing p_2, a_2 can not be in u as it is the node where a_1 starts p_2, thus it has at least started traversing the first edge of p_3. Hence, p_1 concurrently precedes p_3, and p_1 ends up before p_3.

By Algorithm $Seed$, like in the proof of Lemma 10, we can denote by $e_1, \ldots, e_n, \overline{e_1, \ldots, e_n}$ the route followed by a_2 when executing p_3 and by $e_1, \ldots, e_n, s, \overline{e_1, \ldots, e_n, s}$ the route followed by a_1 when executing p_1 where s corresponds to edges traversed at a distance belonging to $\{x_2 + y_2 + 1; \ldots; x_1 + y_1\}$. Remark that according to the definition of a backtrack, $\overline{e_1, \ldots, e_n, s} = \overline{s}, \overline{e_1, \ldots, e_n}$. Consider the moment t_1 when a_2 finishes the first period of p_3 and begins the second one. It has just traversed $e_1, \ldots, e_n$, and is about to execute $\overline{e_1, \ldots, e_n}$. At this moment, a_1 can not have traversed the edges $\overline{e_1, \ldots, e_n}$, or else agents have met by t_1, which would contradict our assumption. However, as p_1 is completed before p_3, a_1 must finish executing $\overline{s}, \overline{e_1, \ldots, e_n}$ before a_2 finishes executing $\overline{e_1, \ldots, e_n}$ which implies that agents meet by the end of the execution of p_1 and contradicts once again the hypothesis that they do not meet by the end of p_2.

So, in every case, it contradicts the assumption that by the end of the execution of Pattern $Cloudberry(x_1, y_1, z, h)$, a_2 neither has met a_1 nor has finished executing S. Hence, the execution of Pattern $Cloudberry(x_1, y_1, z, h)$ by a_1 pushes the execution of S by a_2, and the lemma holds. □

6.2 Agents synchronizations

We recall the reader that D is the initial distance separating the two agents in the basic grid.

The aim of this subsection is to introduce and prove several synchronization properties our algorithms offer (cf., Lemmas 20 and 21). By "synchronization" we mean that if one agent has completed some part of its rendezvous algorithm, then either it must have met the other agent or this other agent has also completed some part (not necessarily the same one) of its algorithm *i.e.,* it must have made progress.

To prove Lemmas 20 and 21, we first need to show some more technical results—Lemmas 17, 18, and 19.

Lemma 17. *Let u and v be the two nodes initially occupied by the agents a_1 and a_2. Let d_1 and $d_2 \geq D$ be two powers of two not necessarily different from each other. If agent a_2 executes Procedure $Assumption(d_1)$ from node u and agent a_1 executes Procedure $PushPattern(d_1, d_2)$ from node v, then Procedure $PushPattern(d_1, d_2)$ pushes Procedure $Assumption(d_1)$.*

Proof. Consider two agents a_1 and a_2. Their respective initial nodes are u and v, which are separated by a distance D. Assume that a_2 executes Procedure $Assumption(d_1)$ with d_1 any power of two, and that a_1 executes $PushPattern(d_1, d_2)$ with $d_2 \geq D$ any other power of two. Assume by contradiction that the execution of $Assumption(d_1)$ by agent a_2 concurrently precedes the execution of Procedure $PushPattern(d_1, d_2)$ by a_1, and that by the end of the execution of the latter, neither agents have met, nor the execution of Procedure $Assumption(d_1)$ is completed.

According to Algorithm 8, there are as many basic patterns (from $\{RepeatSeed; Berry; Cloudberry\}$) in $\mathcal{BD}(PushPattern(d_1, d_2))$ as in $\mathcal{BD}(Assumption(d_1))$. We denote by n this number of basic patterns. Each basic pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$ and $\mathcal{BD}(Assumption(d_1))$ is given an index between 1 and n according to their order of appearance. According to Remark 4, $\mathcal{BD}(Assumption(d_1))$ is perfect. This means that when agent a_2 starts the execution of $Assumption(d_1)$, this agent starts the execution of the first basic pattern in $\mathcal{BD}(Assumption(d_1))$, that when agent a_2 completes the execution of $Assumption(d_1)$, it completes the execution of the n-th basic pattern in $\mathcal{BD}(Assumption(d_1))$, and that, for any integer i between 1 and $n-1$, agent a_2 does not make any edge traversal between the i-th and the $(i+1)$-th basic pattern in $\mathcal{BD}(Assumption(d_1))$. Every edge traversal agent a_2 makes during the execution of Procedure $Assumption(d_1)$ is performed during one of the basic patterns inside $\mathcal{BD}(Assumption(d_1))$. Remark that $\mathcal{BD}(PushPattern(d_1, d_2))$ is also perfect.

Suppose that for any integer i between 1 and n, by the end of the execution of the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$, agents have met or the execution by a_2 of the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ is over. We get a contradiction, as it means that, by the end of the execution of Procedure $PushPattern(d_1, d_2)$ by a_1 (and thus by the end of the n-th pattern of $\mathcal{BD}(PushPattern(d_1, d_2)))$, agents have met or the execution of the n-th pattern inside $\mathcal{BD}(Assumption(d_1))$ (and thus $Assumption(d_1)$ itself) by a_2 is over. As a consequence, there exists an integer i between 1 and n, such that by the end of the execution of the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$ by a_1, agents have not met, and the execution by a_2 of the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ is not over. Without loss of generality, let us make the assumption that i is the smallest positive integer, such that by the end of the execution of the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$ by a_1, agents have not met, and the execution by a_2 of the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ is not over.

Let us first show that the execution of the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ concurrently precedes the execution of the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$. If $i = 1$, since $Assumption(d_1)$ concurrently precedes $PushPattern(d_1, d_2)$, the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ concurrently precedes the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$. If $i > 1$ and the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ does not concurrently precede the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$, then the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ does not begin before the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$, which implies that the $(i-1)$-th pattern inside $\mathcal{BD}(Assumption(d_1))$ ends after the $(i-1)$-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$, which contradicts the hypothesis that i is the smallest positive integer, such that by the end of the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$, agents have not met, and the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ is not over. This means that the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ concurrently precedes the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$.

According to Lemmas 11, 12 and 14, Algorithm $PushPattern$ and the fact that $d_2 \geq D$, whatever the type of the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ ($Berry$, $Cloudberry$ or $RepeatSeed$), the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$ pushes it. In particular, if the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ is a $Berry$ or a $Cloudberry$ called after the test at Line 7, at Line 8 or 10, the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$ pushes it regardless of which of the two patterns it is. Indeed, for any integers x and h, $Cloudberry(x, d_1, d_1, h)$ is composed of several $Berry(x, d_1)$ so that $C(Cloudberry(x, d_1, d_1, h)) \geq C(Berry(x, d_1))$. As the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ concurrently precedes the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$, this contradicts the fact that by the end of the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$, agents have not met, and the i-th pattern inside $\mathcal{BD}(Assumption(d_1))$ is not over.

We then get a contradiction regardless of the case, which proves the lemma. □

Lemma 18. *Let d_1 be any power of two, and x be any integer such that the first parameter of each* *basic pattern inside $\mathcal{BD}(Assumption(d_1))$ is assigned a value which is at most x. For every power of two $d_2 \geq d_1$, the first parameter of each basic pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$ is lower than or equal to $x + 3d_2$.*

Proof. We prove this lemma by contradiction. Make the assumption that there exists a power of two d_1 and an integer x_1 such that the first parameter of each basic pattern inside $\mathcal{BD}(Assumption(d_1))$ is given a value lower than or equal to x_1. Also suppose that there exists a call to a basic pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$ for some power of two $d_2 \geq d_1$ in which the first parameter is given a value greater than $x_1 + 3d_2$. According to Algorithm *PushPattern*, in $\mathcal{BD}(PushPattern(d_1, d_2))$ there cannot be any call to basic Pattern *Cloudberry*, and each basic pattern inside $\mathcal{BD}(Push$-$Pattern(d_1, d_2))$ and $\mathcal{BD}(Assumption(d_1))$ is given an index between 1 and n according to their order of appearance, with n the number of basic patterns in either of these decompositions. Thus, for any integer i between 1 and n, there is a pair of patterns (p_1, p_2) such that p_1 is the i-th basic pattern inside $\mathcal{BD}(Assumption(d_1))$, and p_2 is the i-th pattern inside $\mathcal{BD}(PushPattern(d_1, d_2))$. We consider any pair (p_1, p_2) such that the first parameter of p_2 is given a value greater than $x_1 + 3d_2$, and we analyse three cases depending on the type of pattern p_1. By assumption, the first parameter of p_1 is x_1.

Let us first consider the case in which p_1 is Pattern $RepeatSeed(x_2, n_1)$ with x_2 and n_1 any two integers. According to Algorithm 8, since p_1 is Pattern $RepeatSeed(x_2, n_1)$, p_2 is $Berry(x_2, d_2)$. By assumption, the first parameter of p_2 is greater than x_1+3d_2, which contradicts our other assumption that the first parameter of p_1 is at most x_1.

Thus, p_1 is either Pattern *Berry* or Pattern *Cloudberry*. In $\mathcal{BD}(Assumption(d_1))$, whether it is called directly by Procedure $Assumption(d_1)$, or inside its call to $Harvest(d_1)$, or inside the call of the latter to $PushPattern(d_3, d_1)$ with a power of two $d_3 < d_1$, the second parameter of Pattern *Berry* is always d_1, and the second and third parameters of Pattern *Cloudberry* are always d_1 as well. Let p_1 be Pattern $Berry(x_2, d_1)$ with any integer $x_2 \leq x_1$. According to Algorithm 8, p_2 is $RepeatSeed(d_1 + d_2 + x_2, C(Berry(x_2, d_1)))$. This implies that the first parameter of Pattern p_2 *i.e.,* $d_1 + d_2 + x_2$ is greater than $x_1 + 3d_2$. This means that $x_2 > x_1 - d_1 + 2d_2 > x_1$ which contradicts the assumption that the first parameter of p_1 is at most x_1.

At last, according to Algorithm 8, if p_1 is Pattern $Cloudberry(x_2, d_1, d_1, h)$ with two integers h and $x_2 \leq x_1$, p_2 is $RepeatSeed(d_2 + 2d_1 + x_2, C(Cloudberry(x_2, d_1, d_1, h)))$. This implies that the first parameter of Pattern p_2 *i.e.,* $d_2 + 2d_1 + x_2$ is greater than $x_1 + 3d_2$. This means that $x_2 > x_1 + 2d_2 - 2d_1 > x_1$ which also contradicts the assumption that the first parameter of p_1 is at most x_1.

Hence, within $\mathcal{BD}(PushPattern(d_1, d_2))$, there cannot be a call to a basic pattern in which the first parameter is assigned a value greater than $x_1 + 3d_2$, which proves the lemma. □

Lemma 19. *Let d_1 be a power of two. The first parameter of each basic pattern inside $\mathcal{BD}(Assumption(d_1))$ is at most $\rho(2d_1) - 3d_1$.*

Proof. We prove this lemma by induction on d_1.

Let us first consider that $d_1 = 1$, and prove that the first parameter of each basic pattern inside $\mathcal{BD}(Assumption(1))$ is at most $\rho(2) - 3$. Let us assume by contradiction that there exists a basic pattern inside $\mathcal{BD}(Assumption(1))$ for which the first parameter is given a value greater than $\rho(2)-3$. Denote by p such a pattern. Procedure $Assumption(1)$ begins with $Harvest(1)$ which is composed of calls to $Cloudberry(\rho(1), 1, 1, 0)$ and $RepeatSeed(r(1), C(Cloudberry(1, 1, \rho(1), 0)))$. As $\rho(1)$ and $r(1)$ are lower than $\rho(2) - 3$, pattern p does not belong to $\mathcal{BD}(Harvest(1))$. As a consequence, pattern p is called after $Harvest(1)$. After $Harvest(1)$, the first parameter that is given to the patterns called in Procedure $Assumption(1)$ is always at most $\rho(2)-3$. Indeed, the first parameter is assigned its maximal value when $j = 2d_1(d_1+1) = 4$ and $i = d_1 = 1$ in the while loop *i.e.,* when $3d_1 = 3$ has been added $i(j+1) = 5$ times to $r(d_1) = r(1)$, which gives a maximal value equal to $r(d_1) + 3d_1^2(2d_1(d_1+1)+1) = r(1)+15 = \rho(2d_1) - 3d_1 = \rho(2)-3$. We then get a contradiction with the existence of p since its first parameter is lower than or equal to $\rho(2)-3$.

Let us now assume that there exists a power of two d_2 such that for each power of two $d_3 \leq d_2$, the first parameter of each basic pattern inside $\mathcal{BD}(Assumption(d_3))$ is at most $\rho(2d_2) - 3d_2$, and prove that the first parameter of each basic pattern inside $\mathcal{BD}(Assumption(2d_2))$ is at most $\rho(4d_2) - 6d_2$. Let us assume by contradiction that there exists a basic pattern p inside $\mathcal{BD}(Assumption(2d_2))$ which is assigned a first parameter that is greater than $\rho(4d_2) - 6d_2$. Procedure $Assumption(2d_2)$ begins with $Harvest(2d_2)$ which in turn, begins with $PushPattern(1, 2d_2)$, $\ldots$, $PushPattern(d_2, 2d_2)$. According to the definition of a basic decomposition, if p is called by $PushPattern(1, 2d_2)$, $\ldots$, $PushPattern(d_2, 2d_2)$, it belongs to $\mathcal{BD}(PushPattern(1, 2d_2))$, $\ldots$, $\mathcal{BD}(PushPattern(d_2, 2d_2))$. By induction hypothesis, inside $\mathcal{BD}(Assumption(1))$, $\ldots$, $\mathcal{BD}(Assumption(d_2))$, the first parameter of each basic pattern is at most $\rho(d_2) - 3d_2$. According to Lemma 18, inside $\mathcal{BD}(PushPattern(1, 2d_2))$, $\ldots$, $\mathcal{BD}(PushPattern(d_2, 2d_2))$, the first parameter of each basic pattern is at most $\rho(d_2) + 3d_2 = r(d_2) \leq \rho(2d_2) - 6d_2$. Moreover, after $PushPattern(1, 2d_2)$, $\ldots$, $PushPattern(d_2, 2d_2)$, $Harvest(2d_2)$ executes Pattern $Cloudberry(\rho(2d_2), 2d_2, 2d_2, 0)$ followed by Pattern $RepeatSeed(r(2d_2), C(Cloudberry(2d_2, 2d_2, \rho(2d_2), 0)))$. Inside these calls, the first parameter is respectively given the values $\rho(2d_2)$ and $r(2d_2)$ which are both lower than $\rho(4d_2) - 6d_2$. As a consequence, p does not belong to $\mathcal{BD}(Harvest(2d_2))$. This means that this pattern is called after $Harvest(2d_2)$. However, in the same way as when $d_1 = 1$, we can show that the first parameter keeps increasing and reaches a maximal value equal to $r(2d_2) + 12d_2^2(4d_2(2d_2+1)+1) = \rho(4d_2) - 6d_2$ which contradicts the existence of a basic pattern inside $\mathcal{BD}(Assumption(2d_2))$ which is assigned a first parameter that is greater than $\rho(4d_2) - 6d_2$, and then proves the lemma. □

Before presenting the next lemma, we need to introduce the following notions. We say that the first four lines of Algorithm *Harvest* are its first part, and that the last line is the second part. Procedure *Assumption* begins with a call to Procedure *Harvest*: We will consider that the first part of Procedure *Assumption* is the first part of this call, and that the second part of Procedure *Assumption* is the second part of this call. After these two parts, there is a third part in Procedure *Assumption* which consists of calls to basic patterns. Moreover, note that the execution of Algorithm RV can be viewed as a sequence of consecutive calls to Procedure *Assumption* with an increasing parameter. We will say that the $(i+1)$-th call to Procedure *Assumption* (*i.e.,* the call to Procedure $Assumption(2^i)$ by an agent executing Algorithm RV is Phase i.

Lemma 20. *Consider two agents executing Algorithm RV. Let i be an integer such that $2^i \geq D$. If rendezvous has not occurred before, at the end of the execution by any of both agents of the second part of Phase i, the other agent has finished executing the first part of Phase i.*

Proof. Let a_1 and a_2 be two agents executing Algorithm RV. Let i_1 and d_1 be two integers such that $2_1^i = d_1 \geq D$. Assume by contradiction that at the end of the execution of the second part of Phase i_1 by a_1, agents have not met and a_2 has not completed its execution of the first part of Phase i_1.

By assumption, when a_1 finishes executing the second part of Phase i_1, a_2 is either executing Phase i_2 for an integer $i_2 < i_1$, or the first part of Phase i_1.

First of all, let us show that when a_1 finishes executing the sequence $PushPattern(1, d_1)$, $\ldots$, $PushPattern(2^{i_1-1}, d_1)$ (*i.e.,* the loop at the beginning of procedure $Harvest(d_1)$), a_2 cannot be executing Phase i_2 for an integer $i_2 < i_1$. Indeed, in view of Lemma 17 and the fact that $d_1 \geq D$, we know that the sequence $PushPattern(1, d_1)$, $\ldots$, $PushPattern(2^{i_1-1}, d_1)$ pushes the sequence $Assumption(1)$, $\ldots$, $Assumption(2^{i_1-1})$. This means that by the time a_1 finishes $PushPattern(2^{i_1-1}, d_1)$, the agents have met or a_2 has finished executing Procedure $Assumption(2^{i_1-1})$ *i.e.,* Phase $(i_1 - 1)$. Given that by assumption, agents do not meet before a_1 completes its execution of the first part of Phase i_1, when a_1 finishes executing the loop at the beginning of procedure $Harvest(d_1)$, a_2 is executing the first part of Phase i_1.

Let us now show that when a_1 finishes executing $Cloudberry(\rho(d_1), d_1, d_1, 0)$, a_2 has finished executing the loop at the beginning of Procedure $Harvest(d_1)$. According to Lemmas 18 and 19, inside this loop, the first parameter which is assigned to Patterns *RepeatSeed* and *Berry* is at most $\rho(d_1)$. Besides, while executing this loop, a_2 executes a sequence of Patterns *RepeatSeed* and *Berry* called by Procedure *PushPattern*. Since $d_1 \geq D$, according to Lemma 16, the execution of $Cloudberry(\rho(d_1), d_1, d_1, 0)$ by a_1 pushes the execution by a_2 of the loop at the beginning of Procedure $Harvest(d_1)$. By assumption, when a_1 finishes executing $Cloudberry(\rho(d_1), d_1, d_1, 0)$, agents have not met which implies that a_2 has finished executing the loop.

After executing Pattern $Cloudberry(\rho(d_1), d_1, d_1, 0)$ but before completing Procedure $Harvest(d_1)$, a_1 performs $RepeatSeed(r(d_1), C(Cloudberry(\rho(d_1), d_1, d_1, 0)))$. According to Lemma 12, as $r(d_1) = \rho(d_1) + 3d_1$, the execution of $RepeatSeed(r(d_1), C(Cloudberry(\rho(d_1), d_1, d_1, 0)))$ by a_1 pushes the execution of $Cloudberry(\rho(d_1), d_1, d_1, 0)$ by a_2. Still by assumption, when a_1 finishes executing $RepeatSeed(r(d_1), C(Cloudberry(\rho(d_1), d_1, d_1, 0)))$, agents have not met, and thus a_2 has finished executing $Cloudberry(\rho(d_1), d_1, d_1, 0)$. This means that when a_1 finishes executing $Harvest(d_1)$ and thus the second part of Phase i_1, a_2 has completed the execution of the first part of Phase i_1, which proves the lemma. □

In the hereafter lemma, we focus on the calls to Pattern *RepeatSeed* in the second and in the third part of Procedure $Assumption(d_1)$ for any power of two d_1. In the statement and proof of this lemma, they are called "synchronization *RepeatSeed*", and indexed from 1 to $d_1(2d_1(d_1+1)+1)+1)$ in their ascending execution order in these two parts of the procedure. The call to Pattern *RepeatSeed* in the second part of Procedure *Assumption* is the first (indexed by 1) synchronization *RepeatSeed* during an execution of Procedure $Assumption(d_1)$ for any power of two d_1.

Lemma 21. *Let a_1 and a_2 be two agents executing Algorithm RV. Let u and v be their respective initial nodes separated by a distance D. For every power of two $d_1 \geq D$ and every positive integer i, if agents have not met yet, then when one agent finishes executing the i-th synchronization RepeatSeed of $Assumption(d_1)$, the other agent has at least started executing the i-th synchronization RepeatSeed of $Assumption(d_1)$.*

Proof. Consider two nodes u and v separated by a distance D, and two agents a_1 and a_2 respectively located on u and v. Suppose that agent a_1 has just finished executing the i-th synchronization *RepeatSeed* inside Procedure $Assumption(d_1)$ with any power of two $d_1 \geq D$ and any positive integer i. Let us prove by induction on i that if rendezvous has not occurred yet a_2 has at least started executing this i-th synchronization *RepeatSeed*.

Let us first consider the case in which $i = 1$. The synchronization *RepeatSeed* a_1 has just finished executing is called at the end of the execution of Procedure $Harvest(d_1)$ called at Line 1 of Procedure $Assumption(d_1)$. As $d_1 \geq D$, by Lemma 20, when a_1 finishes executing Pattern

RepeatSeed, and thus $Harvest(d_1)$, agents have met or a_2 has completed the execution of the first part of Procedure $Assumption(d_1)$. This means that when a_1 has finished executing the first synchronization *RepeatSeed*, either agents have met or a_2 has at least begun the execution of the first synchronization *RepeatSeed*.

Let us now make the assumption that, for every power of two $d_1 \geq D$, during an execution of Procedure $Assumption(d_1)$, there exists an integer j between 1 and $d_1(2d_1(d_1+1)+1)$ such that when agent a_1 has finished executing the j-th synchronization *RepeatSeed*, either agents have met or a_2 has at least begun the execution of the j-th synchronization *RepeatSeed*, and prove that when a_1 has finished executing the $(j+1)$-th synchronization *RepeatSeed*, either agents have met or a_2 has at least begun the execution of the $(j+1)$-th synchronization *RepeatSeed*. Let us assume by contradiction that when a_1 has finished executing the $(j+1)$-th synchronization *RepeatSeed*, a_2 has neither met a_1 nor started executing the $(j+1)$-th synchronization *RepeatSeed*.

After executing the j-th synchronization *RepeatSeed*, a_1 executes Line 8 or Line 10 of Algorithm $Assumption(d_1)$ and thus either Pattern *Berry* or Pattern *Cloudberry*, depending on the bits of its transformed label. By Lemmas 14 and 16, as $d_1 \geq D$, if a_2 is still executing the j-th synchronization *RepeatSeed*, whichever pattern a_1 executes, it pushes the execution of the j-th synchronization *RepeatSeed* by a_2. By assumption, when a_1 finishes executing Line 8 or Line 10 of Algorithm $Assumption(d_1)$ after the j-th synchronization *RepeatSeed*, agents have not met which implies that a_2 has finished executing the j-th synchronization *RepeatSeed*.

The next pattern that a_1 executes is the $(j+1)$-th synchronization *RepeatSeed*. Given the above assumptions and statements, when a_1 starts executing this synchronization *RepeatSeed*, a_2 has finished executing the j-th synchronization *RepeatSeed* and has started executing Line 8 or Line 10 of Algorithm $Assumption(d_1)$. By Lemmas 11 and 12, as $d_1 \geq D$, whichever pattern a_2 executes, it is pushed by the execution of the $(j+1)$-th synchronization *RepeatSeed* by a_1. Given that, still by assumption, agents do not meet before a_1 finishes executing the $(j+1)$-th synchronization *RepeatSeed*, when this occurs, a_2 has finished the execution of Line 8 or 10 of Algorithm $Assumption(d_1)$, just after the j-th, and just before the $(j+1)$-th synchronization *RepeatSeed*. Hence, when a_1 finishes executing the $(j+1)$-th synchronization *RepeatSeed*, a_2 has at least started executing the $(j+1)$-th synchronization *RepeatSeed*, which contradicts the hypothesis that when a_1 has finished executing the $(j+1)$-th synchronization *RepeatSeed*, a_2 has neither met a_1 nor started executing the $(j+1)$-th synchronization *RepeatSeed*, and proves the lemma. □

6.3 Correctness of Algorithm RV

Theorem 22. *Algorithm RV solves the problem of rendezvous in the basic grid.*

Proof. To prove this theorem, it is enough to prove the following claim.

Claim 23. *Let d_1 be the smallest power of two such that $d_1 \geq max(D, l')$ with l' the index of the first bit which differs in the transformed labels of the agents. Algorithm RV ensures rendezvous by the time one of both agents completes an execution of Procedure $Assumption(d_1)$.*

This proof is made by contradiction. Suppose that the agents a_1 and a_2 executing Algorithm RV never meet. First, in view of Remark 2, l' exists. Respectively denote by u and v, the initial nodes of a_1 and a_2.

Consider an agent that eventually starts executing $Assumption(d_1)$ where d_1 is the smallest power of two such that $d_1 \geq max(D, l')$. As $d_1 \geq D$, by Lemma 20, we know that as soon as this agent finishes executing Procedure $Harvest(d_1)$, both agents have started executing $Assumption(d_1)$.

Otherwise, agents have met which contradicts our assumption. Without loss of generality, suppose that the bits in the transformed labels of agents a_1 and a_2 with the index l' are respectively 1 and 0. We are going to prove that the agents meet before one of them finishes the execution of $Assumption(d_1)$.

To achieve this, we first show that there exists an iteration of the loop at Line 6 of Algorithm 6 during which the two following properties are satisfied:

1. the value of the variable i is equal to l'
2. the value of the variable j is such that when executing Pattern *Cloudberry* at Line 10, the first pair of Patterns *Seed* and *Berry* executed inside this *Cloudberry* by a_1 starts from the initial node of a_2

As, $d_1 \geq l'$, there is an iteration of the loop at Line 4 during which the first property is verified.

We now show that the second property is also satisfied. Let U be a list of all the nodes at distance at most d_1 from u and ordered in the order of the first visit when executing $Seed(d_1)$ from node u. The same list is considered in the algorithm of Pattern $Cloudberry(x, d_1, d_1, h)$ for any integers x and h. First of all, there are $2d_1(d_1 + 1) + 1$ nodes at distance at most d_1 from u, and thus in U. Since the distance between u and v is $D \leq d_1$, v belongs to U. Let j_1 an integer lower than or equal to $2d_1(d_1 + 1)$ be its index in U. According to Procedure *Assumption*, the value of the variable j is incremented at each iteration of the loop at Line 6 and takes one after another each value lower than or equal to $2d_1(d_1 + 1)$. Consider the iteration when it is equal to j_1. According to Algorithm *Cloudberry*, the first node from which a_1 executes *Seed* and *Berry* is the node which has index $j_1 + 0 \pmod{2d_1(d_1 + 1) + 1} = j_1$. This node is v, which proves that there exists an iteration of the loop at Line 6 (and thus of the loop at Line 4) during which the second property is verified too. Let us denote by I the iteration of the loop at Line 4 which satisfies the two aforementioned properties. It is the iteration after the $(1 + (l' - 1)(2d_1(d_1 + 1) + 1) + j_1)$-th synchronization *RepeatSeed*.

According to Lemma 21, we know that when an agent finishes executing the i-th synchronization *RepeatSeed* inside the second and the third part of any execution of Procedure $Assumption(d_1)$ (for any positive integer i lower than or equal to $d_1(2d_1(d_1+1)+1)+1$), the other agent has at least begun the execution of this synchronization *RepeatSeed*. Thus, when an agent is the first one which starts executing I, it has just finished executing the $(1 + (l' - 1)(2d_1(d_1 + 1) + 1) + j_1)$-th synchronization *RepeatSeed* and the other agent is executing (or finishing executing) the same *RepeatSeed*. Let us prove that rendezvous occurs before any of the agents starts the $(2 + (l' - 1)(2d_1(d_1 + 1) + 1) + j_1)$-th synchronization *RepeatSeed*.

Let us consider the patterns both agents execute between the beginning of the $(1+(l'-1)(2d_1(d_1+1)+1)+j_1)$-th synchronization *RepeatSeed*, and the beginning of the next one. Agent a_1 executes Pattern $RepeatSeed(x, n)$ with x an integer and n a positive integer (call this pattern, p_1) and Pattern $Cloudberry(x, d_1, d_1, j_1)$ from node u while a_2 executes $RepeatSeed(x, n)$ (let us call it p_2) and $Berry(x, d_1)$ (p_3) from node v. During its execution of Pattern $Cloudberry(x, d_1, d_1, j_1)$ from node u, a_1 first follows $P(u, v)$, and then executes Pattern $Seed(x)$ followed by Pattern $Berry(x, d_1)$ both from node v (call them respectively p_4 and p_5). Recall that during any execution of Pattern $Berry(x, d_1)$ from node v, there are two periods, the second one consisting in backtracking every edge traversal made during the first one. During the first period, in particular, an agent executes a Pattern $Seed(x)$ from every node at distance at most d_1. Those patterns include an execution of Pattern $Seed(x)$ from node u and another from v. Since backtracking $Seed(x)$ allows to perform exactly the same edge traversals as $Seed(x)$, during the second period of Pattern $Berry(x, d_1)$, there is also an execution of Pattern $Seed(x)$ from node u and another from v.

Let us consider two different cases. In the first one, when a_1 starts executing p_4 from v, inside p_3, a_2 has not yet started following $P(v,u)$ to go executing $Seed(x)$ from u. In the second one, when a_1 starts executing p_4 from v, a_2 has at least started following $P(v,u)$ to go executing $Seed(x)$ from u. In the following, we analyse both these cases.

Concerning the first case, we get a contradiction. Consider what a_2 can be executing when a_1 starts executing p_4 from node v, after following $P(u,v)$. First, it can still be executing the synchronization *RepeatSeed* p_2 from node v. Then, by Lemma 10, rendezvous occurs. The only other pattern that a_2 can be executing at this moment is p_3. However, in this case, we know that a_2 will have finished its execution of p_3 before a_1 starts p_5, just after p_4. Otherwise, by Lemma 15, rendezvous occurs.

We have just reminded the reader that during any execution of Pattern $Berry(x, d_1)$ from v, agent a_2 performs, among the Patterns $Seed(x)$ from every node at distance at most d_1 from v, Patterns $Seed(x)$ from v. If it executes one of these Patterns $Seed(x)$ while a_1 is executing its p_4 from node v after following $P(u,v)$, by Lemma 10, rendezvous occurs. This implies that before a_1 finishes following $P(u,v)$, a_2 has completed each execution of Pattern $Seed(x)$ from v inside its execution of $Berry(x, d_1)$.

It means that, each execution of Pattern $Seed(x)$ from node v during the second period of p_3 has already been completed by a_2 when a_1 starts executing its own $Seed(x)$ from v. Since inside the second period of p_3, a_2 executes Pattern $Seed(x)$ from node v, a_2 has already executed the whole first period of p_3 when a_1 starts executing p_4 from v including Pattern $Seed(x)$ performed from node u, as u is at distance at most d_1 from v. This contradicts the definition of this first case: according to this definition, when a_1 starts executing p_4 from v, inside p_3, a_2 has not followed $P(v,u)$ yet, and thus has not executed $Seed(x)$ from u.

Concerning the second case, we prove that rendezvous occurs, which is also a contradiction. Recall that in this case, when a_1 starts executing p_4 from v, a_2 has at least started following $P(v,u)$ to go executing $Seed(x)$ from u. If a_2 has not finished following $P(v,u)$ when a_1 starts executing $P(u,v)$, then if we denote by t_1 (resp. t_2) the time when a_1 (resp. a_2) finishes following $P(u,v)$ (resp. $P(v,u)$), agents meet by time $min(t_1,t_2)$ as $P(u,v) = \overline{P(v,u)}$. If a_2 has finished following $P(v,u)$ before a_1 starts executing $P(u,v)$, then it has begun executing $Seed(x)$ from u before a_1 finishes executing p_1 (before it executes $Cloudberry(x, d_1, d_1, j_1)$), which means by Lemma 10 that agents achieve rendezvous.

So, whatever the execution chosen by the adversary, rendezvous occurs in the worst case by the time any agent completes $Assumption(d_1)$, which contradicts the assumption that rendezvous never happens. This proves the claim, and by extension the theorem. □

6.4 Cost analysis

Theorem 24. *The cost of Algorithm RV is polynomial in D and l.*

Proof. In order to prove this theorem, we first need to show the following two claims.

Claim 25. *Let d_1 be any power of two. The cost of each basic pattern inside $\mathcal{BD}(Assumption(d_1))$ is polynomial in d_1.*

Let us prove this claim. First, the costs of these basic patterns are polynomial in d_1 if the values of their parameters are polynomial in d_1. Indeed, $C(Seed(x)) \in O(x^2)$, $C(RepeatSeed(x,n)) \in O(n \times C(Seed(x)))$, $C(Berry(x,y)) \in O((x+y)^6)$, and $C(Cloudberry(x,y,z,h)) \in O(z^2 \times (C(Seed(x)) + C(Berry(x,y))))$.

Pattern *Seed* does not belong to $\mathcal{BD}(Assumption(d_1))$. It is called when executing the other basic patterns, which give it a parameter which is polynomial in their own parameters. Hence, we focus on the parameters of Pattern *RepeatSeed*, *Berry*, and *Cloudberry*, and prove that their values are polynomial in d_1.

For each basic Pattern *Berry* or *Cloudberry* inside $\mathcal{BD}(Assumption(d_1))$, the value given to its second parameter is always d_1. For each basic Pattern *Cloudberry* inside $\mathcal{BD}(Assumption(d_1))$, the value assigned to the third parameter of Pattern *Cloudberry*, is always d_1. The fourth parameter of Pattern *Cloudberry* does not have any impact on its cost since it only modifies the order in which the edge traversals are made, and not their number.

The first parameter of these three basic patterns can take various complicated values, but they are still polynomial in d_1. Indeed, according to Lemma 19, for any power of two d_1, inside $\mathcal{BD}(Assumption(d_1))$, the value of this first parameter is at most $\rho(2d_1) - 3d_1$, which is polynomial in d_1.

At last, the second parameter of Pattern *RepeatSeed*, is always equal to $C(p)$ where p is one of the other patterns, either *Berry* or *Cloudberry*. Besides, since the parameters given to this pattern p are polynomial in d_1, this is also the case for the second parameter of Pattern *RepeatSeed*. Hence the claim is proven.

Claim 26. *Let d_1 be a power of two. The cost of Procedure Assumption(d_1) is polynomial in d_1.*

Let us prove this claim. According to the definition of a basic decomposition, and of Remark 4, for any power of two d_1, each edge traversal performed during an execution of Procedure $Assumption(d_1)$ is performed by one of the basic patterns inside $\mathcal{BD}(Assumption(d_1))$. The cost of Procedure $Assumption(d_1)$ is the same as the sum of the costs of all the basic patterns inside $\mathcal{BD}(Assumption(d_1))$. According to Claim 25, we know that for any power of two d_1, inside $\mathcal{BD}(Assumption(d_1))$, each basic pattern is polynomial in d_1. Thus, to prove this claim it is enough to show that $\mathcal{BD}(Assumption(d_1))$ contains a number of basic patterns which is polynomial in d_1.

For any power of two d_1, Procedure $Assumption(d_1)$ is composed of a call to Procedure $Harvest(d_1)$ and the nested loops. These loops consist in $2d_1(2d_1(d_1+1)+1)$ calls to basic patterns. Half of them are made to *RepeatSeed* and the others either to *Berry* or to *Cloudberry*. In its turn, $Harvest(d_1)$ is composed of two parts: a loop calling Procedure *PushPattern* and two basic patterns. For any power of two d_2, in view of Algorithm 8, and since they are both perfect, the number of basic patterns inside $\mathcal{BD}(PushPattern(d_2, d_1))$ or $\mathcal{BD}(Assumption(d_2))$ is the same. As a consequence, if $d_1 \geq 2$, $\mathcal{BD}(PushPattern(1, d_1))$, $\ldots$, $\mathcal{BD}(PushPattern(\frac{d_1}{2}, d_1))$ is composed of as many basic patterns as there are in $\mathcal{BD}(Assumption(1))$, $\ldots$, $\mathcal{BD}(Assumption(\frac{d_1}{2}))$.

For any power of two i, let us denote by $L_1(i)$ (resp. $L_2(i)$) the number of calls to basic patterns inside $\mathcal{BD}(Assumption(i))$ (resp. $\mathcal{BD}(Harvest(i))$). We then have the following equations:

$$L_1(i) = L_2(i) + 2i(2i(i+1)+1)$$
$$L_2(i) = \sum_{j=0}^{\log_2(i)-1} (L_1(2^j)) + 2$$

They imply the following:

$$L_2(1) = 2 \quad \text{and}$$
$$\text{if} \quad i \geq 2 \quad \text{then} \quad L_2(i) = L_2(\frac{i}{2}) + L_1(\frac{i}{2}) = 2L_2(\frac{i}{2}) + i(i(\frac{i}{2}+1)+1)$$

Hence, $L_2(i) \in \mathrm{O}(i^5)$. Both $L_2(i)$ and $L_1(i)$ are polynomial in i, which means that for any power of two d_1, $\mathcal{BD}(Assumption(d_1))$ is composed of number of basic patterns which is polynomial in d_1. Hence, in view of Claim 25, the cost of $Assumption(d_1)$ is indeed polynomial in d_1, which proves the claim.

Now, it remains to conclude the proof of the theorem. According to Claim 23, rendezvous is achieved by the end of the execution of $Assumption(\delta)$ by any of both agents, where δ is the smallest power of two such that $\delta \geq max(D, l')$ and l' is the index of the first bit which differs in the transformed labels of the agents. So, according to Claim 26, the cost of $Assumption(\delta)$ is polynomial in D and l', and by extension polynomial in D and l as by construction we have $l' \leq 2l+2$. Moreover, before executing $Assumption(\delta)$, all the calls to Procedure *Assumption* use an input parameter lower than δ and thus, each of these calls is also polynomial in D and l. Hence, in view of the fact that the number of calls to procedure *Assumption* before executing $Assumption(\delta)$ belongs to $\Theta(\log \delta)$ (the input parameter of *Assumption* doubles after each call), the theorem follows.

□

7 Conclusion

From Theorems 1, 22 and 24, we obtain the following result concerning the task of approach in the plane.

Theorem 27. *The task of approach can be solved at cost polynomial in the unknown initial distance Δ separating the agents and in the length of (the binary representation) of the shortest of their labels.*

Throughout the paper, we made no attempt at optimizing the cost. Actually, as the acute reader will have noticed, our main concern was only to prove the polynomiality. Hence, a natural open problem is to find out the optimal cost to solve the task of approach. This would be all the more important as in turn we could compare this optimal cost with the cost of solving the same task with agents that can position themselves in a global system of coordinates (the almost optimal cost for this case is given in [10]) in order to determine whether the use of such a system (*e.g.,* GPS) is finally relevant to minimize the travelled distance.

References

[1] Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.*, 36(1):56–82, 2006.

[2] Steve Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002.

[3] Steve Alpern. *The theory of search games and rendezvous.* International Series in Operations Research and Management Science, Kluwer Academic Publishers, 2003.

[4] Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas, and Arnaud Labourel. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, pages 297–311, 2010.

[5] Sébastien Bouchard, Yoann Dieudonné, and Bertrand Ducourthial. Byzantine gathering in networks. *Distributed Computing*, 29(6):435–457, 2016.

[6] Jérémie Chalopin, Yoann Dieudonné, Arnaud Labourel, and Andrzej Pelc. Rendezvous in networks in spite of delay faults. *Distributed Computing*, 29(3):187–205, 2016.

[7] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. Comput.*, 41(4):829–879, 2012.

[8] Reuven Cohen and David Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM J. Comput.*, 34(6):1516–1528, 2005.

[9] Reuven Cohen and David Peleg. Convergence of autonomous mobile robots with inaccurate sensors and movements. *SIAM J. Comput.*, 38(1):276–302, 2008.

[10] Andrew Collins, Jurek Czyzowicz, Leszek Gasieniec, and Arnaud Labourel. Tell me where I am so I can meet you sooner. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, pages 502–514, 2010.

[11] Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Computing*, 25(2):165–178, 2012.

[12] Jurek Czyzowicz, Andrzej Pelc, and Arnaud Labourel. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms*, 8(4):37, 2012.

[13] Gianlorenzo D'Angelo, Gabriele Di Stefano, and Alfredo Navarra. Gathering on rings under the look-compute-move model. *Distributed Computing*, 27(4):255–285, 2014.

[14] Shantanu Das, Dariusz Dereniowski, Adrian Kosowski, and Przemyslaw Uznanski. Rendezvous of distance-aware mobile agents in unknown graphs. In *Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings*, pages 295–310, 2014.

[15] Shantanu Das, Flaminia L. Luccio, and Euripides Markou. Mobile agents rendezvous in spite of a malicious agent. In *Algorithms for Sensor Systems - 11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2015, Patras, Greece, September 17-18, 2015, Revised Selected Papers*, pages 211–224, 2015.

[16] Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *Distributed Computing, 20th International Symposium, DISC 2006, Stockholm, Sweden, September 18-20, 2006, Proceedings*, pages 46–60, 2006.

[17] Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

[18] Yoann Dieudonné and Andrzej Pelc. Deterministic polynomial approach in the plane. *Distributed Computing*, 28(2):111–129, 2015.

[19] Yoann Dieudonné and Andrzej Pelc. Anonymous meeting in networks. *Algorithmica*, 74(2):908–946, 2016.

[20] Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1, 2014.

[21] Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM J. Comput.*, 44(3):844–867, 2015.

[22] Yoann Dieudonné and Franck Petit. Self-stabilizing gathering with strong multiplicity detection. *Theor. Comput. Sci.*, 428:47–57, 2012.

[23] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005.

[24] Pierre Fraigniaud and Andrzej Pelc. Deterministic rendezvous in trees with little memory. In *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*, pages 242–256, 2008.

[25] Pierre Fraigniaud and Andrzej Pelc. Delays induce an exponential memory gap for rendezvous in trees. *ACM Transactions on Algorithms*, 9(2):17, 2013.

[26] Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada, and Masafumi Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *SIAM J. Comput.*, 41(1):26–46, 2012.

[27] Dariusz R. Kowalski and Adam Malinowski. How to meet in anonymous network. *Theor. Comput. Sci.*, 399(1-2):141–156, 2008.

[28] Evangelos Kranakis, Danny Krizanc, and Sergio Rajsbaum. Mobile agent rendezvous: A survey. In *Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO 2006, Chester, UK, July 2-5, 2006, Proceedings*, pages 1–9, 2006.

[29] Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theor. Comput. Sci.*, 355(3):315–326, 2006.

[30] Avery Miller and Andrzej Pelc. Fast rendezvous with advice. In *Algorithms for Sensor Systems - 10th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2014, Wroclaw, Poland, September 12, 2014, Revised Selected Papers*, pages 75–87, 2014.

[31] Avery Miller and Andrzej Pelc. Time versus cost tradeoffs for deterministic rendezvous in networks. *Distributed Computing*, 29(1):51–64, 2016.

[32] Linda Pagli, Giuseppe Prencipe, and Giovanni Viglietta. Getting close without touching: near-gathering for autonomous mobile robots. *Distributed Computing*, 28(5):333–349, 2015.

[33] Thomas Schelling. *The Strategy of Conflict.* Oxford University Press, Oxford, 1960.

[34] Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999.

[35] Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms*, 10(3):12, 2014.