Asynchronous approach in the plane: A deterministic polynomial algorithm

Sébastien Bouchard[1], Marjorie Bournat[1], Yoann Dieudonné[2], Swan Dubois[1], and Franck Petit[1]

1 **Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA, LIP6 UMR 7606, Paris, France**
`<firstname>.<lastname>@lip6.fr`

2 **Laboratoire MIS, Université de Picardie Jules Verne**
33 rue Saint-Leu, 80039 Amiens, France
`<firstname>.<lastname>@u-picardie.fr`

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.

Type of submission: Regular paper
Eligible for the best student paper award

1998 ACM Subject Classification G.2.2 Graph Theory; C.2.4 Distributed Systems

Keywords and phrases 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. 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 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 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, more formally described in Sections 4—details on the algorithm, the proofs of correction, and cost analysis are given in appendix. Finally, we make some concluding remarks in Section 5.

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] (and outlined in the preliminaries section of the full version given in appendix), 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.*

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 $P(u, v) = \overline{P(v, u)}$.

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, acts 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 1 in Section 4) works in phases numbered $0, 1, 2, 3, 4, \ldots$ During Phase i (cf. Procedure *Assumption* called at line 3 in Algorithm 1), 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 (relying in particular on Procedure *Berry* called at line 9 in Algorithm 2) and the route related to bit 1 (relying in particular on on Procedure *Cloudberry* called at line 11 in Algorithm 2) 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 jth 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 15 in Algorithm 2) 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 jth bit. This constraint is imposed on the adversary by dividing each bit process into some predefined steps and by ensuring that after each step s of the kth bit process, for any $k \leq 2^\alpha$, an agent follows a specific route that forces the other agent to complete the step s of its kth bit process. 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 2). This is the most tricky part of our algorithm, one of the main idea of which relies in particular 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 implements this idea (this point is refined in the penultimate paragraph above Definition 2 in Section 4). 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 *in fine* this low cost synchronization while still ensuring rendezvous. However, it is through this original and novel way of moving that we finally get the polynomial cost.

4 Formal description of our algorithm and its analysis

The purpose of this section is to give the formal description of our solution and the involved subroutines along with their main objectives and how they work at a high level. The main algorithm that solves the rendezvous in a basic grid is Algorithm RV (shown in Algorithm 1). As mentioned in Section 3, 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 following feature: Given two distinct labels, their transformed labels are never prefixes of each other. As explained in the previous section, we need such a feature because our solution requires that at some point both agents follow different routes by processing different bit values.

Algorithm 1 RV

```
1: $d \leftarrow 1$
2: **while** agents have not met yet **do**
3:     Execute $Assumption(d)$
4:     $d \leftarrow 2d$
5: **end while**
```

Algorithm RV makes use of a subroutine, *i.e.,* Procedure *Assumption*. When an agent executes this procedure with a parameter α that is a "good" assumption *i.e.,* that upperbounds the initial distance D and the value j of the smallest bit position for which both transformed labels differ, we have the guarantee that rendezvous occurs by the end of this execution. In the rest of this section, we assume that α is the smallest good assumption that upperbounds D and j. The code of Procedure *Assumption* is given in Algorithm 2. It makes use, for technical reasons, of the sequence r that is defined below.

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

$$\forall \text{ power of two } i,\ r(i) = \rho(i) + 3i$$

Procedure *Assumption* can be divided into two parts. The first part consists of the execution of Procedure *Harvest* (line 1 of Algorithm 2) and corresponds to the second synchronization mechanism mentioned in Section 3. The main feature of this procedure is the following: when the earlier agent finishes the execution of $Harvest(\alpha)$ within the execution of $Assumption(\alpha)$, we have the guarantee that the later agent has at least started to execute *Assumption* with parameter α (actually, as explained below, we have even the guarantee that most of $Harvest(\alpha)$ has been executed by the later agent). Procedure *Harvest* is presented below. The second part of Procedure *Assumption* (cf. lines $2-19$ of Algorithm 2) consists in processing the bits of the transformed label one by one. More precisely when processing a given bit in a call to Procedure $Assumption(d)$, the agent acts in steps $0, 1, \ldots, 2d(d+1)$: After each of these steps, the agent executes Pattern *RepeatSeed* whose role is described below. In each of these steps, the agent executes *Berry* (resp. *Cloudberry*) if the bit it is processing is 0 (resp. 1). These patterns of moves (cf. Algorithms 5 and 6) are made in such a way that rendezvous occurs by the time any agent finishes the process of its jth bit in $Assumption(\alpha)$ if we have the following synchronization property. Each time any of both agents starts executing a step s during the process of its jth bit in $Assumption(\alpha)$, the other agent has finished the execution of either step $s-1$ in the jth bit process of $Assumption(\alpha)$ if $s>0$, or the last step of the $(j-1)$th bit process of $Assumption(\alpha)$ if $s=0$ ($j>0$ in view of the label transformation given above). To obtain such a synchronization, an agent executes what we called the first synchronization mechanism in the previous section (cf. line 15 in Algorithm 2) after each step of a bit process. Actually, this mechanism relies on procedure *RepeatSeed*, the code of which is given in Algorithm 8. Note that the total number of steps, and thus of executions of *RepeatSeed*, in $Assumption(\alpha)$ is $2\alpha^2(\alpha+1)+\alpha$. For every $0 \leq i \leq 2\alpha^2(\alpha+1)+\alpha$, the ith execution of *RepeatSeed* in $Assumption(\alpha)$ by an agent permits to force the other agent to finish the execution of its ith step in $Assumption(\alpha)$ by repeating a pattern *Seed* (its main purpose is described just above its code given by Algorithm 7): With the appropriate parameters, this pattern *Seed* covers any pattern (*Berry* or *Cloudberry*) made in the ith step of $Assumption(\alpha)$ and the number of times it is repeated is at least the maximal number of edge traversals we can make in the ith step of $Assumption(\alpha)$.

Algorithm 2 $Assumption(d)$

```
1: Execute $Harvest(d)$
2: $radius \leftarrow r(d)$
3: $i \leftarrow 1$
4: while $i \leq d$ do
5:     $j \leftarrow 0$
6:     while $j \leq 2d(d+1)$ do
7:         // Begin of step $j$
8:         if the length of the transformed label is strictly greater than $i$, or its $i$th bit is 0 then
9:             Execute $Berry(radius, d)$
10:        else
11:            Execute $Cloudberry(radius, d, d, j)$
12:        end if
13:        // End of step $j$
14:        $radius \leftarrow radius + 3d$
15:        Execute $RepeatSeed(radius, C(Cloudberry(radius - 3d, d, d, j)))$
16:        $j \leftarrow j + 1$
17:    end while
18:    $i \leftarrow i + 1$
19: end while
```

Algorithm 3 gives the code of Procedure *Harvest*. As in Procedure *Assumption*, it makes use, for technical reasons, of two sequences ρ and r that are defined above. Procedure *Harvest* is made of two parts: the executions of Procedure *PushPattern* (lines $1-3$ of Algorithm 3), and the calls to the patterns *Cloudberry* and *RepeatSeed* (lines $4-5$ of Algorithm 3). When $Harvest$ is executed with parameter α (which is a good assumption), the first part ensures that the later agent has at least completed every execution of *Assumption* with a parameter that is smaller than α, while the second part ensures that the later agent has completed almost the entire execution of $Harvest(\alpha)$ (more precisely, when the earlier agent finishes the second part, we have the guarantee that it remains for the later agent to execute at most the last line before completing its own execution of $Harvest(\alpha)$).

To give further details on Procedure $Harvest$, let us first describe Procedure *PushPattern* (its code is given in Algorithm 4). When the earlier agent completes the execution of *Push-Pattern*$(2i, d)$ with i some power of two, assuming that the later agent had already completed *Assumption*(i), we have the guarantee that the later agent has completed its execution of *Assumption*$(2i)$. To ensure this, we regard the execution of *Assumption*$(2i)$ as a sequence of calls to basic patterns (namely *RepeatSeed*, *Berry* and *Cloudberry*), which is formally defined in Definition 2. This sequence is what we meant when talking about "the pattern drawn on the grid" in Section 3. For each basic pattern p_1 in the sequence, the earlier agent executes another pattern p_2 at the end of which we ensure that the later agent has completed p_1. If p_1 is either Pattern *Berry* or Pattern *Cloudberry*, then p_2 is Pattern *RepeatSeed*: we use the same idea here as for the first synchronization mechanism. If p_1 is Pattern *RepeatSeed*, then p_2 is Pattern *Berry*, relying on a property of the route $X\overline{X}$ (with X any non-empty route) introduced in the last paragraph of Subsection 3.2: if both agents follow this route concurrenly from the same node, then they meet. Pattern *Seed* can be seen as such a route, and Procedure *Berry* (whose code is shown in Algorithm 5) consists in executing Pattern *Seed* from each node at distance at most α. Hence, unless they meet, the later agent completes its execution of Pattern *RepeatSeed* before the earlier one starts executing *Seed* from the same node. Note that *PushPattern* uses as many patterns as the number of basic patterns in the sequence it is supposed to push: this and the fact of doubling the value of the input parameter of Procedure *Assumption* in Algorithm 1 contribute in particular to keep the polynomiality of our solution.

Thus, once the earlier agent completes the first part of $Harvest(\alpha)$, the later one has at least started the execution of *Assumption*(α) (and thus of the first part of $Harvest(\alpha)$). At this point, we might think at first glance that we just shifted the problem. Indeed, the number of edge traversals that has to be made to complete all the executions of *Assumption* prior to *Assumption*(α) is quite the same, if not higher, than the number of edge traversals that has to be made when executing the first part of $Harvest(\alpha)$. Hence the difference between both agents in terms of edge traversals has not been improved here. However, a crucial and decisive progress has nonetheless been done: contrary *a priori* to the series of *Assumption* executed before *Assumption*(d), the first part of $Harvest(\alpha)$ can be pushed at low cost via the execution of Pattern *Cloudberry* (line 4 of Algorithm 3) by the earlier agent. Actually this pattern corresponds to the kind of route, described at the end of Subsection 3.2 for the second synchronization mechanism, which is of small length compared to the sequence of patterns it can push. Indeed, the first part of $Harvest(\alpha)$ can be viewed as a "large" sequence of Patterns *Seed* and *Berry*: however *Seed* and *Berry* can be seen (by analogy with Subsection 3.2) as routes of the form $A\overline{A}$ and $B\overline{B}$ respectively, while Pattern *Cloudberry* executes *Seed* and *Berry* (i.e., $A\overline{A}B\overline{B}$) once from at least each node at distance at most α.

Note that when the earlier agent has completed the execution of Pattern *Cloudberry* in $Harvest(\alpha)$, the later agent has at least started the execution of Pattern *Cloudberry* in $Harvest(\alpha)$. Hence, there is still a difference between both agents, but it has been considerably reduced: it is now relatively small so that we can handle it pretty easily afterwards.

Algorithm 3 $Harvest(d)$

```
1: for $i \leftarrow 1$; $i < d$; $i \leftarrow 2i$ do
2:     Execute $PushPattern(i, d)$
3: end for
4: Execute $Cloudberry(\rho(d), d, d, 0)$
5: Execute $RepeatSeed(r(d), C(Cloudberry(\rho(d), d, d, 0)))$
```

▶ **Definition 2** (Basic and Perfect Decomposition)**.** Given a call P to an algorithm, we say that the basic decomposition of P, denoted by $\mathcal{BD}(P)$, is P itself if P corresponds to a basic pattern, 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. The basic decomposition of every call to Procedure *Assumption* is perfect.

Algorithm 4 $PushPattern(i, d)$

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

Algorithm 5 Pattern $Berry(x, y)$

```
1: /* First period */
2: Let $u$ be the current node
3: for $i \leftarrow 1$; $i \le x + y$; $i \leftarrow i + 1$ do
4:     for $j \leftarrow 0$; $j \le i$; $j \leftarrow j + 1$ do
5:         for $k \leftarrow 0$; $k \le j$; $k \leftarrow k + 1$ do
6:             for each node $v$ at distance $k$ from $u$ ordered clockwise from the North do
7:                 Follow $P(u, v)$
8:                 Execute $Seed(i - j)$
9:                 Follow $P(v, u)$
10:            end for
11:        end for
12:    end for
13: end for
14: /* Second period */
15: $L \leftarrow$ the path followed by the agent during the first period
16: Backtrack by following the reverse path $\overline{L}$
```

Algorithm 6 Pattern $Cloudberry(x, y, z, h)$

```
1: /* First period */
2: Let $u$ be the current node
3: 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 $u$
4: for $i \leftarrow 0$; $i \leq 2z(z+1)$; $i \leftarrow i+1$ do
5:     Let $v$ be the node with index $h + i \pmod{2z(z+1)+1}$ in $U$
6:     Follow $P(u, v)$
7:     Execute $Seed(x)$
8:     Execute $Berry(x, y)$
9:     Follow $P(v, u)$
10: end for
11: /* Second period */
12: $L \leftarrow$ the path followed by the agent during the first period
13: Backtrack by following the reverse path $\overline{L}$
```

Algorithm 7 Pattern $RepeatSeed(x, n)$

```
1: Execute $n$ times Pattern $Seed(x)$
```

Starting from a node v, the main purpose of $Seed(x)$ is 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 (informally, the procedure permits to cover an area of radius x).

Algorithm 8 Pattern $Seed(x)$

```
1: /* First period */
2: for $i \leftarrow 1$; $i \leq x$; $i \leftarrow i+1$ do
3:     /* Phase $i$ */
4:     Perform $(N(SE)^i(WS)^i(NW)^i(EN)^i)$
5: end for
6: /* Second period */
7: $L \leftarrow$ the path followed by the agent during the first period
8: Backtrack by following the reverse path $\overline{L}$
```

According to the proof of correctness of Algorithm RV and its cost analysis that are given in appendix:

▶ **Theorem 3.** *Algorithm RV solves the problem of rendezvous in the basic grid.*

▶ **Theorem 4.** *The cost of Algorithm RV is polynomial in D and l, where D is the initial (Manhattan) distance separating both agents and l is the length of the shortest label.*

5 Conclusion

From Theorems 1, 3 and 4, we obtain the following result concerning the task of approach in the plane.

▶ **Theorem 5.** *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.