HAL
open science

Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids

Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas, Arnaud Labourel

To cite this version:

Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas, Arnaud Labourel. Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids. DISC 2010, Sep 2010, United States. pp.297–311, 10.1007/978-3-642-15763-9_28 . hal-00534263

HAL Id: hal-00534263

https://hal.science/hal-00534263

Submitted on 9 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Almost Optimal Asynchronous Rendezvous in Infinite Multidimensional Grids

Evangelos Bampas[1,⋆], Jurek Czyzowicz[2], Leszek Gąsieniec[3], David Ilcinkas[1,⋆], and Arnaud Labourel[1,⋆,⋆⋆]

[1] LaBRI, CNRS / INRIA / Université de Bordeaux
{bampas,ilcinkas,labourel}@labri.fr
[2] Université du Québec
Jurek.Czyzowicz@uqo.ca
[3] University of Liverpool
L.A.Gasieniec@liverpool.ac.uk

Abstract. Two anonymous mobile agents (robots) moving in an asynchronous manner have to meet in an infinite grid of dimension $\delta > 0$, starting from two arbitrary positions at distance at most d. Since the problem is clearly infeasible in such general setting, we assume that the grid is embedded in a δ-dimensional Euclidean space and that each agent knows the Cartesian coordinates of its own initial position (but not the one of the other agent). We design an algorithm permitting the agents to meet after traversing a trajectory of length $O(d^\delta \text{polylog } d)$. This bound for the case of 2D -grids subsumes the main result of [12]. The algorithm is almost optimal, since the $\Omega(d^\delta)$ lower bound is straightforward.
Further, we apply our rendezvous method to the following network design problem. The ports of the δ-dimensional grid have to be set such that two anonymous agents starting at distance at most d from each other will always meet, moving in an asynchronous manner, after traversing a $O(d^\delta \text{polylog } d)$ length trajectory.
We can also apply our method to a version of the geometric rendezvous problem. Two anonymous agents move asynchronously in the δ-dimensional Euclidean space. The agents have the radii of visibility of r_1 and r_2, respectively. Each agent knows only its own initial position and its own radius of visibility. The agents meet when one agent is visible to the other one. We propose an algorithm designing the trajectory of each agent, so that they always meet after traveling a total distance of $O((\frac{d}{r})^\delta \text{polylog}(\frac{d}{r}))$, where $r = \min(r_1, r_2)$ and for $r \geq 1$.

1 Introduction

1.1 The problem and the model

Consider a Euclidean δ-dimensional space $\mathcal{F}$. We construct an infinite grid G_δ of dimension δ as follows. The set of nodes of G_δ are the points of $\mathcal{F}$ with

⋆ Partially supported by the ANR project ALADDIN, the INRIA project CEPAGE and by a France-Israel cooperation grant (Multi-Computing project).

⋆⋆ Corresponding author: LaBRI (bât A30), Université Bordeaux 1, 351 cours de la Libération, F-33405 Talence cedex, France. Tph. +33 (0)5 40 00 66 69.

integer coordinates. We link by an edge two nodes $u = (u_1, u_2, \dots, u_\delta)$ and $v = (v_1, v_2, \dots, v_\delta)$ iff there is $i \in \{1, 2, \dots, \delta\}$, s.t., $\forall j \neq i, u_j = v_j$ and $u_i = v_i \pm 1$. At each node, the endpoints of edges incident to it are labeled by unique integers from the set $\{1, 2, \dots, 2\delta\}$, called *port numbers*.

The *route* of each agent is a sequence of adjacent edges which are subsequently traversed during its movement. The actual timing of the *walk* of each agent along its route is *asynchronous*, i.e., it is controlled by an adversary. The adversary initially places both agents at any two nodes in the grid. Given the coordinates of its initial location v_0, the route chosen by an agent is a sequence of edges in the grid $(e_1, e_2, \dots)$, s.t., in stage i the agent traverses the edge $e_i = [v_{i-1}, v_i]$, starting at v_{i-1} and finishing at v_i. Stages are repeated indefinitely until rendezvous is accomplished. We assume that each agent starts its walk at any time, but both agents are placed by the adversary at their respective initial positions at the same time, and since that moment any moving agent may encounter the other agent, even if the other agent didn't start its walk yet.

We describe the walk f of an agent on its route following definitions from [12, 15, 16]. Let $R = (e_1, e_2, \dots)$ be the route of an agent. Let $(t_1, t_2, \dots)$, where $t_1 = 0$, be an increasing sequence of reals, chosen by the adversary, that represent points in time. Let $f_i : [t_i, t_{i+1}] \to [v_i, v_{i+1}]$ be any monotonous, continuous function, chosen by the adversary, s.t., $f_i(t_i) = v_i$ and $f_i(t_{i+1}) = v_{i+1}$. For any $t \in [t_i, t_{i+1}]$, we define $f(t) = f_i(t)$. The interpretation of the walk f is as follows. At time t the agent is at point $f(t)$ of its route. The adversary may arbitrarily vary the speed of the agent for as long as the walk of the agent in each segment is continuous and monotonous.

Agents with routes R_1 and R_2 and with walks $f^{(1)}$ and $f^{(2)}$ meet at time t, if points $f^{(1)}(t)$ and $f^{(2)}(t)$ are identical. A rendezvous is guaranteed for routes R_1 and R_2, if the agents using these routes meet, regardless of the walks chosen by the adversary. The *cost* of the rendezvous algorithm is measured by the sum of the lengths of the trajectories of both agents from their starting locations until the time t of the rendezvous. Since the actual portions of these trajectories may vary depending on the adversary, we consider the maximum of this sum, i.e., the worst-case over all possible walks chosen for both agents by the adversary. In this paper we are looking for a rendezvous algorithm of the smallest possible cost with respect to the unknown original distance d between the agents.

1.2 Related work

The rendezvous problem was first described in [38]. A detailed discussion of the large literature on rendezvous can be found in the excellent book [3]. Most of the results in this domain can be divided into two classes: those considering the geometric scenario (rendezvous in the line, see, e.g., [7, 8, 23], or in the plane, see, e.g., [5, 6]), and those discussing rendezvous in graphs, e.g., [2, 4]. A generalization of the rendezvous problem is that of gathering [21, 27–29, 35, 41], when more than two agents have to meet in one location.

If graphs are unlabeled, deterministic rendezvous requires breaking symmetry, which can be accomplished either by allowing marking nodes or by labeling

the agents. Deterministic rendezvous with anonymous agents working in unlabeled graphs but equipped with tokens used to mark nodes was considered e.g., in [33]. In [43] the authors studied gathering of many agents with unique labels. In [17, 31, 40] deterministic rendezvous in graphs with labeled agents was considered. However, in all the above papers, the synchronous setting was assumed. Asynchronous gathering under geometric scenarios has been studied, e.g., in [13, 21, 37] in different models than ours: agents could not remember past events, but they were assumed to have at least partial visibility of the scene. The first paper to consider deterministic asynchronous rendezvous in graphs was [16]. The authors concentrated on complexity of rendezvous in simple graphs, such as the ring and the infinite line. They also showed feasibility of deterministic asynchronous rendezvous in arbitrary finite connected graphs with *known* upper bound on the size. Further improvements of the above results for the infinite line were proposed in [39]. Gathering many robots in a graph, under a different asynchronous model and assuming that the whole graph is seen by each robot, has been studied in [28, 29].

The assumption that the distributed mobile entities know their initial location in the geometric environment was considered in the past, e.g., in the context of geometric routing, see, e.g., [1, 9, 32, 34], where it is typically assumed that the source node knows the position of the destination as well as its own position, or broadcasting [19, 20], where the position awareness of only the broadcasting node is admitted. Such assumption, partly fueled by the availability and the expansion of the Global Positioning System (GPS), is sometimes called *location awareness* of agents or nodes of the network, and it often leads to better bounds of the proposed solutions.

An alternative approach to location awareness is adopted in *network design* where the nodes of the network are preprocessed in order to enable or to speed up a certain distributed process. For example, in the context of graph exploration it was shown in [11] that a suitable *a priori* coloring of the nodes with 3 colors provides a graph environment that can be explored by an agent equipped with a constant size memory. In contrast, if the preprocessing is not allowed it is known [22] that an agent requires $\Omega(\log n)$ bits of memory to be able to explore all graphs of order n. Another approach to network design consists in graph preprocessing by setting the port numbers, so that the graph exploration is easy, i.e., with constant memory, e.g. see [24, 26] or memoryless agents, [14, 18, 30].

The efficient rendezvous solutions proposed in this paper rely directly on the use of *space-covering sequences* introduced recently in [12]. The space-covering sequences are close relatives to *space-filling curves* studied extensively in the literature, see, e.g., [10, 25, 36, 42]. The space-filling curves visit every point in an infinite grid exactly once. Gotsman and Lindenbaum pointed out in [25] that for any space-filling curve there always exist some close points in the grid that are arbitrarily far apart on the space-filling curve, i.e., these type of curves fail in preserving locality in the worst case. The deficiency of space-filling curves comes from the assumption that they must visit each point in a discrete 2D-space exactly once. The authors of [12] propose a novel structure of *space-covering*

sequences that traverse points in a discrete 2D-space repeatedly. They show that for any two points located at an arbitrary distance d in the 2D-space there are well defined and efficiently computable instances of these two points in the sequence at distance $O(d^{2+\varepsilon})$ apart, for any positive constant ε.

1.3 Our results

In this paper, we design efficient rendezvous algorithms for anonymous agents asynchronously moving in multidimensional infinite grids. In section 2, we consider the location aware agents, i.e., agents knowing the Cartesian coordinates of their own initial positions in the integer grid. We give an almost optimal $O(d^{\delta}\text{polylog } d)$ rendezvous algorithm where δ is the space dimension of the grid. Our approach, applied to the 2-dimensional grid, results in an $O(d^2\text{polylog } d)$ algorithm, improving the recent main result from [12], i.e., a $O(d^{2+\varepsilon})$ upper bound. This may be seen as an exponential improvement on the deficiency factor that leads to an almost optimal solution since a straightforward lower bound on the worst case distance is $\Omega(d^2)$. The consequence of our approach is the design of the more efficient and simpler space-covering sequences introduced in [12]. In section 3 we show how this approach may be applied to the case of rendezvous of two agents with positive visibility radii, moving in a Euclidean space. Again each agent is aware of its original position (but not the position of the other agent). Finally, in section 4, we show that it is possible to set the port numbers of the integer grid in the δ-dimensional space to achieve an efficient rendezvous. In this case, the anonymous agents are not location aware.

2 Rendezvous algorithm for location aware agents in the grid

In this section we assume that the grid belongs to the Euclidean δ-dimensional space $\mathcal{F}$ and each agent knows the integer Cartesian coordinates of its initial position in the space. We assume that the agents have coherent compasses and a common unit of length permitting to refer to the same system of Cartesian coordinates.

The general idea of our approach is the following. The rendezvous algorithm constructs the trajectory of the agents following the integer grid lines of the Euclidean space. Hence each such line contains points $(x_1, x_2, \ldots, x_\delta)$, where x_i, for some fixed $1 \leq i \leq \delta$ is any real value, and each x_j, for $j \neq i$, $1 \leq j \leq \delta$ is some integer. We will define an infinite sequence of grids, each grid inducing a partition of the space into δ-dimensional hypercubes. The size of these hypercubes increases when following this sequence of partitions. We then define a directed acyclic graph whose nodes are the hypercubes of all such partitions, with arcs between intersecting hypercubes belonging to consecutive partitions (the arcs are directed from the larger to the smaller hypercubes). For any initial position p of the agent, we consider an infinite tree $T(p)$ containing all hypercubes of the family containing point p, such that for each node in $T(p)$ its successors

are also in $T(p)$ and if the same hypercube is obtained this way more than once it is duplicated so that $T(p)$ remains a tree. The route produced for each agent will correspond to the upward movement along $T(p)$ interleaved with some depth-first-search type traversals of the siblings of the newly visited node. We show that the two agents will eventually visit some hypercube, belonging to both their respective trees, which contains the two initial positions of the agents. The route corresponding to the visit of such a hypercube will result in rendezvous.

2.1 Euclidean space partitions induced by multidimensional grids

We consider an infinite sequence of partitions $\Pi = \pi_1, \pi_2, \ldots$ of the grid G_δ into hypercubes of dimension δ in $\mathcal{F}$. For the sake of simplicity, in the rest of the paper, we will use the term *hypercube* for a hypercube of dimension δ. The corners of hypercubes in π_i are points $u = (u_1, u_2, \ldots, u_\delta)$ such that $\forall j \in \{1, 2, \ldots, \delta\}, u_j = 2^{i-1} + k2^i$ for some integer k. Each grid π_i partitions space $\mathcal{F}$ into hypercubes of side length 2^i. To assure that each π_i forms an exact partition, we assume that each hypercube H contains, besides its interior points, the corner v having maximum coordinates, as well as all open f-faces containing v, for $f = 1, 2, \ldots, \delta - 1$. Such corner v is called the *reference point* of the hypercube containing it. For example, a 3-dimensional hypercube, having points $(0, 0, 0)$ and $(1, 1, 1)$ as its corners, is a union of its reference point $v = (1, 1, 1)$, the three open edges of the cube incident to v, the three open square faces incident to v, and the interior of the cube.

Lemma 1. *For positive integers i and k, such that $i \geq k + 1$, any hypercube located in partition π_i intersects $(2^k + 1)^\delta$ hypercubes belonging to partition π_{i-k}.*

Proof. Let $\mathcal{C}$ be a hypercube drawn from partition π_i. Consider any edge e of $\mathcal{C}$ along dimension $l \in \{1, \ldots, \delta\}$. This edge is of length 2^i. Due to the definition of Π the edge e cannot be aligned with edges in partition π_{i-k} and each endpoint of e is located in the centre of some hypercube in this partition. Since hypercubes in partition π_{i-k} are of size 2^{i-k}, the edge e must penetrate (including its endpoints) exactly $\frac{2^i}{2^{i-k}} + 1 = 2^k + 1$ hypercubes in π_{i-k}. Finally, since this phenomenon applies to every dimension, the number of hypercubes in partition π_{i-k} intersected by a hypercube in partition π_i is $(2^k + 1)^\delta$. □

By Γ we denote the subsequence $\gamma_1 = \pi_{i_1}, \gamma_2 = \pi_{i_2}, \ldots$, of the above sequence of partitions, such that the indices i_j are defined by the recurrence:

$$\begin{aligned} i_1 &= 1 \\ i_{j+1} &= i_j + \max\{1, \lceil \log i_j \rceil\} \end{aligned}$$

We consider the infinite, directed acyclic graph T, whose nodes are the hypercubes of the partitions $\gamma_1, \gamma_2, \ldots$ and there is a directed edge in T from hypercube P to Q, if

- P and Q are from two consecutive grids γ_k and γ_{k-1}, respectively, and

– P and Q have a nonempty intersection

For any point p of the Euclidean space, we call the *ascending path* $A_k(p)$ the path of T formed of the hypercubes $S_1(p), S_2(p), \ldots, S_k(p)$, such that each hypercube $S_i(p)$ belongs to grid γ_i and $p \in S_i(p)$. By $T_k(p)$ we denote the tree containing the ascending path $A_k(p)$, rooted at $S_k(p)$, obtained from T in such a way that, each time a hypercube has more than one predecessor in T, it is split (duplicated together with all its succession class) so eventually a tree is obtained. By $T(p)$ we denote the infinite tree obtained in the similar way.

Lemma 2. *For a given point p of the Euclidean space and for $s \geq 4$, any hypercube of size $s \cdot 2^{\lceil \log \log s \rceil}$ belonging to $T(p)$ has $(2^{\lceil \log \log s \rceil} + 1)^{\delta}$ children.*

Proof. The proof follows directly from the definition of $T(p)$ and from Lemma 1. □

Lemma 3. *For any pair of points p_1, p_2 at distance d in the Euclidean space and any $\delta + 1$ partitions $\gamma_{j_1}, \gamma_{j_2}, \ldots, \gamma_{j_{\delta+1}}, \in \Pi$, each one composed of hypercubes of size at least $4d$, there exists a hypercube of one of the partitions which contains both points p_1, p_2.*

Proof. First, we have to show the following claim.

Claim. Let H_1 and H_2 be two hyperplanes (of dimension $\delta - 1$) in $\mathcal{F}$ separating hypercubes in distinct partitions of $\gamma_{j_1}, \gamma_{j_2}, \ldots, \gamma_{j_{\delta+1}}$. If H_1 and H_2 are parallel then they are at distance at least $2d$ of each other.

Proof of the claim. Let H_1 and H_2 be hyperplanes separating hypercubes in γ_i and γ_j respectively. H_1 is defined by $x_l = 2^{i-1} + k2^i$ for some $l \in \{1, 2, \ldots, \delta\}$ and $k \in \mathbb{Z}$. Similarly, H_2 is defined by $x_{l'} = 2^{j-1} + k'2^j$ for some $l' \in \{1, 2, \ldots, \delta\}$ and $k' \in \mathbb{Z}$). Notice that the normal vector of H_1 is parallel to the x_l-axis and that the normal vector of H_2 is parallel to the $x_{l'}$-axis. H_1 and H_2 are parallel if and only if $l = l'$. Assume without loss of generality that $i < j$. The distance between the hyperplanes is equal to :

$$\begin{aligned} dist(H_1, H_2) &= |(2^{i-1} + k2^i) - (2^{j-1} + k'2^j)| \\ &= 2^{i-1}|(1 + 2k) - 2^{j-i}(1 + 2k')| \end{aligned}$$

Notice that $|(1 + 2k) - 2^{j-i}(1 + k'2k')|$ is an integer since k, k' and 2^{j-i} are integers. Moreover, $|(1+2k) - 2^{j-i}(1+k'2k')|$ is odd and so it is greater than zero, since $(1+2k)$ is odd and $2^{j-i}(1+2k')$ is even. It follows that $dist(H_1, H_2) \geq 2^{i-1}$. Finally, since the size of the hypercubes in γ_i is $2^i \geq 4d$, the hyperplanes are at distance at least $2d$. This ends the proof of the claim.

Assume, by contradiction, that there are two points p_1, p_2 at distance d that are not in the same hypercube in any partition in $\gamma_{j_1}, \gamma_{j_2}, \ldots, \gamma_{j_{\delta+1}}$. Since p_1 and p_2 are not in the same hypercube in each γ_{j_i}, there is at least one hyperplane H_i

separating p_1 and p_2, i.e., the segment joining p_1 and p_2 intersects H_i. Let $\mathcal{H}$ be the set $\{H_1, H_2, \ldots, H_{\delta+1}\}$. There are at least two parallel hyperplanes H and H' in $\mathcal{H}$ since the normal vector of each H_i is parallel to an axis of the space $\mathcal{F}$. The segment s joining p_1 and p_2 intersects H in u and H' in v. The distance between p_1 and p_2 is at least the distance between u and v, hence, at least the distance between H and H'. However, by the Claim, H and H' are at distance at least $2d$, a contradiction. It follows, that there exists a hypercube in one of the partitions containing both points p_1 and p_2. □

2.2 The algorithm

In the beginning the agent is at its original position p. The agent calls procedure `Initialize` (line 2) to move to the reference point of the hypercube H_1 from γ_1 (the lowest level grid of the construction) containing p. H_1 is the starting point in the ascending path $A(p)$ of the infinite tree $T(p)$, represented also by the single-node tree $T_1(p)$. The execution of the algorithm corresponds to the traversal of the ascending path $A(p)$, constructing iteratively the sequence of trees $T_1(p), T_2(p), \ldots$. We say that, when an agent advances on the ascending path to a new hypercube H_{i+1}, which is the root of the tree $T_{i+1}(p)$, the agent *explores* the new hypercube H_{i+1}. The exploration of a new hypercube is an attempt to meet the other agent, which, at this time, may be exploring the same hypercube or one of its ancestors.

The exploration of the hypercube H_{i+1} is made during the i-th iteration of the main loop of the `RV` algorithm. At the beginning of the i-th iteration the agent is placed in the reference point of hypercube H_i – the root of $T_i(p)$. The first phase of the loop consists in moving to the reference point of H_{i+1}. Iteratively, each preceding sibling of H_i is traversed in the backward sense by `Traverse` procedure and the previous sibling is reached by the corresponding connector (lines 7-9). Eventually, when the smallest sibling is reached and traversed, the connector to its parent is traversed (line 11), leading to the reference point of H_{i+1}. Finally, H_{i+1} is traversed in the forward sense by `Traverse` procedure (line 13).

The entire route of the agent is the concatenation of a sequence of connectors between different hypercubes of the construction (most of them repeated many times). Each such connector either joins consecutive siblings or it joins the first child (the smallest among all siblings) and its parent. In order to use efficient (i.e., short) connectors, the children of each node are arranged in such a way that the first child contains the reference point of its parent, and the consecutive siblings are adjacent hypercubes, i.e., they share a $(\delta - 1)$-face, cf. Lemma 4 below. Then, when children are of diameter r, all connectors between them or joining them to parents are of length $O(r)$.

The algorithm uses a variable *height* which is the height of the hypercube in the tree $T(p)$, which is currently being explored. The variable H corresponds to the hypercube at the reference point of which the agent currently is. The operation `Connect`(H, C) generates a connector from the reference point of the current hypercube H to the reference point of the hypercube C. Notice

```
Algorithm RV(point p)
height ← 1;
H ← Initialize(v);
Traverse(H, 1, forward);
repeat
    P ← Parent(H, p)
    Traverse(H, height, backward);
    while PrevSib(H, P) exists do
        Connect(H, PrevSib(H, P)); H ← PrevSib(H, P);
        Traverse(H, height, backward);
    height ← height + 1;
    Connect(H, P);
    H ← P;
    Traverse(H, height, forward);
until rendezvous
```

that C must be either the first child, parent, next sibling or previous sibling of H. The next sibling and previous sibling of a node can be obtained by procedure `NextSib`(H, P) and `PrevSib`(H, P) respectively. Observe that these procedures use a parent P of the hypercube H as a parameter since a hypercube can have one of several parents and so one of several potential next or previous siblings. The procedure `Parent`(H, p) returns the parent of H containing point p.

The procedure `Traverse`$(H, height, sense)$ performs essentially the depth-first-search traversal of the subtree of $T(p)$ rooted at hypercube H at height *height*. The parameter *sense* permits to indicate an orientation for the traversal of $T(p)$, either *backward* or $forward$. The traversal of a hypercube performed with parameter *sense* equal to *backward* is the reverse of the traversal done with *sense* equal to $forward$. The procedure `TraverseUnit` performs the traversal of a hypercube of size 2 in γ_1.

```
procedure Traverse(H, height, sense)
if (height = 1) then
    TraverseUnit(H, 1, sense)
else
    C ← FirstChild(H);
    Connect(H, C);
    while NextSib(C, H) exists do
        if(sense = forward) then
            Traverse(C, height − 1, forward);
        Connect(C, NextSib(C, H)); C ← NextSib(C, H);
    Traverse(C, height − 1, sense);
    while PrevSib(C, H) exists do
        Connect(C, PrevSib(C, H)); C ← PrevSib(C, H);
        if(sense = backward) then
            Traverse(C, height − 1, backward);
    Connect(C, H);
```

Lemma 4. *The total length of the connectors linking the space-covering of the children of a hypercube H of size $s \cdot 2^{\lceil \log \log s \rceil}$ is $O(s \cdot 2^{\delta(\log \log s+1)})$.*

Proof. Note first that the children of H form also a hypercube of size $s(2^{\lceil \log \log s \rceil} + 1)$ composed of smaller hypercubes of size s (this is not a hypercube which is an element of any of the grid partitions considered). In each hypercube of size s, the corner with the largest coordinates is chosen as its *representative*. The representatives can be interpreted as nodes of another hypercube H_r of size $s \cdot 2^{\lceil \log \log s \rceil}$ in which edges are of length s. It is possible to find a Hamiltonian path in H_r connecting the representatives, sharing the $(\delta-1)$-faces, and further use the edges of this tour as connectors. Since the number of representatives is $2^{\delta(\log \log s+1)}$ and the edges of the tour are of length s, the length of the Hamiltonian path and in turn the total length of the connectors is bounded by $O(s \cdot 2^{\delta(\log \log s+1)})$. □

Lemma 5. *The length of the trajectory of the agent corresponding to the exploration of a hypercube of size s is $O(s^\delta \log s)$.*

Proof. A hypercube of size $s \cdot 2^{\lceil \log \log s \rceil}$ has $(2^{\lceil \log \log s \rceil}+1)^\delta$ children by Lemma 2 and the total length of the connectors of its children is $O(s \cdot 2^{\delta(\log \log s+1)})$ by Lemma 4. Hence, the function $\lambda(s)$ defining the length of the portion of the trajectory corresponding to the exploration of a hypercube of size s is given by the following recurrence :

$$\begin{aligned} \lambda(1) &= c_1 \\ \lambda(s \cdot 2^{\lceil \log \log s \rceil}) &= (2^{\lceil \log \log s \rceil} + 1)^\delta \lambda(s) + O(s \cdot 2^{\delta(\log \log s+1)}) \end{aligned}$$

Let $c_2 \geq 1$ be a constant whose exact value will be defined later. We prove by induction that there is a constant c_3, such that for $s \geq c_2$, we have the following property:

$$\mathcal{P}_s : \lambda(s) \leq c_3 s^\delta \log s$$

We assume that the constant c_3 is large enough such that $c_3 \geq \frac{\lambda(c_2)}{c_2^\delta \log c_2}$. Hence, $\mathcal{P}_{c_2}$ is true. Now, we assume by induction that $\mathcal{P}_s$ is true. We have:

$$\begin{aligned} \lambda(s \cdot 2^{\lceil \log \log s \rceil}) &\leq (2^{\lceil \log \log s \rceil} + 1)^\delta \lambda(s) + c_4 s \cdot 2^{\delta(\log \log s+1)} \\ &\leq (2^{\lceil \log \log s \rceil} + 1)^\delta c_3 s^\delta \log s + c_4 s \cdot 2^{\delta(\log \log s+1)} \end{aligned}$$

Hence to prove that $\mathcal{P}_{s2^{\lceil \log \log s \rceil}}$ is true, it is sufficient to prove that:

$$\begin{aligned} (2^{\lceil \log \log s \rceil} + 1)^\delta c_3 s^\delta \log s + c_4 s \cdot 2^{\delta(\log \log s+1)} &\leq c_3 s^\delta 2^{\delta \lceil \log \log s \rceil} \log(s \cdot 2^{\lceil \log \log s \rceil}) \\ \frac{(2^{\lceil \log \log s \rceil} + 1)^\delta}{2^{\delta \lceil \log \log s \rceil}} \log s + \frac{c_4 2^\delta}{c_3 s^{\delta-1}} &\leq \log s + \log \log s \end{aligned}$$

Notice that:

$$\begin{aligned} (2^{\lceil \log \log s \rceil} + 1)^\delta &= 2^{\delta \lceil \log \log s \rceil} + \sum_{k=0}^{\delta-1} \binom{\delta}{k} 2^{k \lceil \log \log s \rceil} \\ &\leq 2^{\delta \lceil \log \log s \rceil} + 2^\delta 2^{(\delta-1)\lceil \log \log s \rceil} \end{aligned}$$

Hence it is sufficient to prove that:

$$\begin{aligned}
\frac{2^{\delta\lceil\log\log s\rceil}+2^{\delta}2^{(\delta-1)\lceil\log\log s\rceil}}{2^{\delta\lceil\log\log s\rceil}}\log s+\frac{c_4 2^{\delta}}{c_3 s^{\delta-1}} &\leq \log s+\log\log s\\
\frac{2^{\delta}}{2^{\lceil\log\log s\rceil}}\log s+\frac{c_4 2^{\delta}}{c_3 s^{\delta-1}} &\leq \log\log s\\
2^{\delta}+\frac{c_4 2^{\delta}}{c_3 s^{\delta-1}} &\leq \log\log s \qquad \text{since } \log s\leq 2^{\lceil\log\log s\rceil}\\
2^{\delta}+\frac{2^{\delta}}{s^{\delta-1}} &\leq \log\log s \qquad \text{assuming that } c_3\geq c_4
\end{aligned}$$

The left term tends to 2^{δ} for s going to the infinity. Since the right term tends to the infinity and is positive for s going to the infinity, there must exist a constant c_5 such that the inequality is satisfied for all $s \geq c_5$. By taking $c_2 = c_5$ and $c_3 = \max\left\{c_4, \frac{\lambda(c_2)}{c_2^{\delta}\log c_2}\right\}$, all the assumptions made during the proof are fulfilled. Finally, $\mathcal{P}_s$ is true for every s by induction. □

Theorem 1. *Suppose that a pair of agents is originally placed at any two points p_1, p_2 at distance at most d in the δ-dimensional infinite integer grid. Consider the trajectories computed by* `RV`*(p_1) and* `RV`*(p_2). Any asynchronous walk along these trajectories results in the rendezvous of the agents after traversing trajectories of length $O(d^{\delta}\log^{\delta^2+\delta+1} d)$.*

Proof. We show first that all the connectors may be correctly computed by the agent using `RV` algorithm. Consider first the connectors used in algorithm `RV`. Suppose that the agent stores in its memory the global variable *height* - the height of the hypercube actually explored, the coordinates of its initial position p as well as the number of iterations of the loop from lines 7-9 that have already been performed within the current iteration of the main loop from lines 4-14. From this information and from the knowledge how the siblings at each level have been arranged, the agent may compute in which direction (i.e., positive or negative direction of some of the δ axes of the grid) the connector to the next or the previous sibling goes. The length of this connector is a function of *height* solely. Similarly the connector between a parent and its first child is easily computed since the position of the reference point of the first child is easily computed from the reference point of the parent.

In order to compute correctly the connectors computed in the `Traverse` procedure, we suppose that for each active call of `Traverse` we keep on the recursive stack the local variable *height* as well as the number of the sibling in lines 8 and 14 for which the current recursive call from its parent node is made. The data stored on the recursive stack and the knowledge how the siblings at each level are arranged allow to compute the direction and the length of each of the connectors.

In order to consider the moment of termination of the algorithm take $(\delta+1)$ consecutive grids $\gamma_k, \gamma_{k+1}, \ldots, \gamma_{(k+\delta)}$ of Γ, such that γ_k is the first grid of Γ

containing hypercubes of size at least $4d$. By Lemma 3, one of the grid partitions contains some hypercube H containing both points p_1 and p_2. Suppose, by symmetry, that the agent originally placed at point p_1 is the first one to complete the traversal of H. Since the other agent is all the time situated inside the segment of the trajectory corresponding to H the agents must meet.

The largest such hypercube H (the one belonging to $\gamma_{k+\delta}$) has the size in $O(d \log^{\delta+1} d)$. By an application of Lemma 5 for $s = O(d \log^{\delta+1} d)$, we obtain the statement of the theorem. □

To show that the above result is almost optimal observe that there are $\Omega(d^\delta)$ integer grid points in the d-neighborhood of any point. Since the adversary may hold one of the agents for arbitrary long time close to its initial position, the second agent must eventually explore its entire d-neighborhood using a trajectory of length $\Omega(d^\delta)$.

3 Location aware agents in Euclidean space with non-zero visibility

We now consider a model in which an agent at point p in the δ-dimensional Euclidean space $\mathcal{F}$ *sees* all points in the δ-dimensional ball of radius $r \geq 1$ centered at p. We say that the agent has *visibility range* r. In this model, rendezvous occurs when *one* of the agents sees the other. Agents may have different visibility radii, thus it is not necessary that they both see each other at the same time. Each agent is aware of the coordinates of its own location at any time.

The idea is to apply the algorithm developed in the previous section for location-aware agents with zero visibility, but choosing the size of the lowest-level hypercubes in such a way that it is as large as possible and whenever an agent traverses the portion of its trajectory that corresponds to the lowest level hypercubes, it sees all of their interior points.

Each agent, knowing its visibility radius r, determines the maximum integer k (possibly negative) such that $k \leq \log \frac{r}{\sqrt{\delta}}$. Note that this implies $r \geq 2^k \sqrt{\delta}$, therefore the visibility radius of the agent is at least the diameter of a hypercube of size 2^k. This implies that an agent visiting a node of this hypercube sees all the interior points. Having computed the value of k, the agent moves from its original position to the closest reference point of some hypercube of size 2^k of the appropriate layer in the hierarchy Π. From that point, it starts executing the algorithm of the previous section with a unit hypercube of size 2^k.

In a more general setting, the two agents have different visibility radii r_1 and r_2. As explained above, they compute their respective values k_1 and k_2 and they start executing the rendezvous algorithm with hypercubes of respective sizes 2^{k_1} and 2^{k_2}. Essentially, the two agents ascend the same tree-like structure T as before, except that they start at a higher level which is possibly not the same for the two agents. In any case, one of the agents will start exploring first a hypercube that is guaranteed to contain both starting points, according to Lemma 3, and thus it will be able to see the other agent effecting rendezvous.

Due to the modification of the size of the unit hypercubes of the agents, the algorithm behaves in the worst case as if the distance between the starting points of the agents is scaled by $2^{-k} < \frac{2\sqrt{\delta}}{r}$, where $r = \min\{r_1, r_2\}$. This results in a trajectory of length $O((\frac{2d\sqrt{\delta}}{r})^{\delta} \log^{\delta^2+\delta+1}(\frac{2d\sqrt{\delta}}{r})) = O((\frac{d}{r})^{\delta} \log^{\delta^2+\delta+1}(\frac{d}{r}))$.

4 Setting the grid port numbers for efficient rendezvous

We now sketch how to convert the algorithm we developed in Section 2 into an algorithm suitable for anonymous grids with local port numbering. The idea is to put labels on the nodes of the grid, and then using these labels simulate the rendezvous algorithm in the absence of location awareness on the part of the agents. We then discuss how to manipulate the local port numbers at each node so that the agent is able to extract each node's label by performing a short exploration of its neighborhood.

Omitting the details of the labeling process due to lack of space, we just mention that for dimension δ we need $O(2^{\delta})$ different labels. By choosing carefully the order of the children of each hypercube, we can make sure that the labels provide the agent with enough information to execute the rendezvous algorithm. We now elaborate on the port setting scheme that we use. Let $\mathbf{x} = (x_1, \ldots, x_\delta)$ be an arbitrary node in the δ-dimensional grid. For i in the range $1 \leq i \leq \delta$, we assign port number i of node $\mathbf{x}$ to the edge connecting node $\mathbf{x}$ to node $(x_1, \ldots, x_{i-1}, x_i + 1, x_{i+1}, \ldots, x_\delta)$. The remaining port numbers from $\delta + 1$ to 2δ may be assigned in any of $\delta!$ ways to the remaining edges outgoing from $\mathbf{x}$. This assignment is described by a permutation $\sigma_{\mathbf{x}}$ of $\{1, \ldots, \delta\}$, such that for all i in the range $1 \leq i \leq \delta$, port number $\delta + \sigma_{\mathbf{x}}(i)$ of node $\mathbf{x}$ is assigned to the edge connecting node $\mathbf{x}$ to node $(x_1, \ldots, x_{i-1}, x_i - 1, x_{i+1}, \ldots, x_\delta)$.

Now, suppose that the agent finds itself at node $\mathbf{x}$. It is possible to recover the permutation $\sigma_{\mathbf{x}}^{-1}$, thus also the permutation $\sigma_{\mathbf{x}}$, in the following way: for each i in the range $1 \leq i \leq \delta$, (a) follow port number $\delta + i$ of node $\mathbf{x}$, (b) observe the port number j ($1 \leq j \leq \delta$) that corresponds to the edge just traversed in the local port order of the new node, (c) follow port number j of the new node to go back to the original node $\mathbf{x}$. Then, the agent can deduce that $\sigma_{\mathbf{x}}^{-1}(i) = j$.

Knowing $\sigma_{\mathbf{x}}$, the agent knows exactly which port number corresponds to each direction out of node $\mathbf{x}$ in the δ-dimensional grid. Furthermore, the port setting scheme we just described endows each node with one out of $\delta!$ distinct labels. Therefore, for large enough dimension δ, it is possible to encode the required node labels using the available port number permutations.

We modify the rendezvous algorithm so that whenever an agent arrives at a node $\mathbf{x}$, it performs the following trajectory: it follows port numbers $\delta+1, \ldots, 2\delta$, in that order, and then if follows port numbers $2\delta, \ldots, \delta + 1$, in that order, each time returning to node $\mathbf{x}$. During the first phase of this trajectory (ports $\delta + 1, \ldots, 2\delta$), the agent recovers the permutation $\sigma_{\mathbf{x}}$, and thus the label of $\mathbf{x}$. This modification increases the length of the trajectory of each agent by a factor of $O(\delta)$. Moreover, since the interjected trajectory is a palindrome and both agents perform it at each node they visit, in the worst case they will meet while

exploring the neighborhood of the same node as the one on which they would meet using the original algorithm.

Thus, we arrive at the following theorem:

Theorem 2. *For large enough δ, it is possible to preprocess the local port numbers in the δ-dimensional infinite grid so that two agents originally placed at distance at most d in the grid can achieve rendezvous after traversing trajectories of size $O(d^\delta \log^{\delta^2+\delta+1} d)$.*

For smaller number of dimensions δ, it is still possible to encode all the required labels in the grid, by coarsening the grid on which the algorithm is executed and using more than one node of the original grid to encode the label of a node in the coarsened grid. Details are omitted due to lack of space.

5 Final remarks

Several interesting questions remain unanswered. E.g., is it possible to design an optimal $O(d^\delta)$ rendezvous algorithm in δ-dimensional grids? Is it possible to extend the location aware approach to some classes of graphs other than grids? In Section 4 there seems to be a lot of freedom when choosing possible port arrangements. Is it possible to use a more sophisticated port arrangements so that the mobile agent uses $o(\log d)$ or even a constant number of its memory bits?

References

1. I. Abraham, D. Dolev, and D. Malkhi, LLS: a locality aware location service for mobile ad hoc networks, In Proc. *DIALM-POMC 2004*, pp. 75-84.
2. S. Alpern, The rendezvous search problem, SIAM J. on Control and Optimization 33 (1995), 673-683.
3. S. Alpern and S. Gal, The theory of search games and rendezvous. Int. Series in Operations research and Management Science, number 55, Kluwer Academic Publishers, 2002. Kluwer Academic Publisher, 2002.
4. J. Alpern, V. Baston, and S. Essegaier, Rendezvous search on a graph, Journal of Applied Probability 36 (1999), 223-231.
5. E. Anderson and S. Fekete, Asymmetric rendezvous on the plane, Proc. 14th Annual ACM Symp. on Computational Geometry, 365-373, 1998.
6. E. Anderson and S. Fekete, Two-dimensional rendezvous search, Operations Research 49 (2001), 107-118.
7. V. Baston and S. Gal, Rendezvous on the line when the players' initial distance is given by an unknown probability distribution, SIAM J. on Control and Optimization 36 (1998), 1880-1889.
8. V. Baston and S. Gal, Rendezvous search when marks are left at the starting points, Naval Res. Log. 48 (2001), 722-731.
9. P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, Routing with guaranteed delivery in ad hoc wireless networks, *Wireless Networks* **7(6)**, pp. 609-616, 2001.

10. K. Buchin, Constructing Delaunay Triangulations along Space-Filling Curves, In Proc. *ESA 2009*, pp. 119-130.
11. R. Cohen, P. Fraigniaud, D. Ilcinkas, A. Korman, and D. Peleg, Label-guided graph exploration by a finite automaton, *ACM Transactions on Algorithms* 4(4), pp. 1-18, 2008.
12. A. Collins, J. Czyzowicz, L. Gasieniec and A. Labourel, "Tell me where I am so I can meet you sooner: Asynchronous rendezvous with location information" In Proc. of *ICALP 2010*.
13. M. Cieliebak, P. Flocchini, G. Prencipe, N. Santoro, Solving the Robots Gathering Problem, Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP 2003), 1181-1196.
14. J. Czyzowicz, S. Dobrev, L. Gasieniec, D. Ilcinkas, J. Jansson, R. Klasing, I. Lignos, R.A. Martin, K. Sadakane, W.-K. Sung, More Efficient Periodic Traversal in Anonymous Undirected Graphs, In Proc *SIROCCO'09*, pp. 167-181.
15. J. Czyzowicz, A. Labourel, and A. Pelc, How to meet asynchronously (almost) everywhere, In Proc. of *SODA 2010*, pp. 22-30.
16. G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, U. Vaccaro, Asynchronous deterministic rendezvous in graphs, Theoretical Computer Science 355 (2006), 315-326.
17. A. Dessmark, P. Fraigniaud, D. Kowalski, and A. Pelc, Deterministic rendezvous in graphs, Algorithmica 46 (2006), 69-96.
18. S. Dobrev, J. Jansson, K. Sadakane, and W.-K. Sung, Finding short right-hand-on-the-wall walks in graphs, In Proc. *SIROCCO'05*, pp. 127-139.
19. Y. Emek, L. Gasieniec, E. Kantor, A. Pelc, D. Peleg, and C. Su, Broadcasting in UDG radio networks with unknown topology, *Distributed Computing* **21(5)**, pp. 331-351, 2009.
20. Y. Emek, E. Kantor and D. Peleg, On the effect of the deployment setting on broadcasting in Euclidean radio networks, In Proc. *PODC 2008*, pp. 223-232.
21. P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous oblivious robots with limited visibility, In Proc. *STACS'01*, pp. 247-258.
22. P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg, Graph exploration by a finite automaton, *Theoretical Computer Science* 345(2-3), pp. 331-344, 2005.
23. S. Gal, Rendezvous search on the line, Operations Research 47 (1999), 974-976.
24. L. Gasieniec, R. Klasing, R.A. Martin, A. Navarra, and X. Zhang, Fast periodic graph exploration with constant memory, *J. on Computer Systems and Sciences* 74(5), pp. 808-822, 2008.
25. C. Gotsman and M. Lindenbaum, On the metric properties of discrete space-filling curves, *IEEE Transactions on Image Processing*, **5(5)**, pp. 794-797, 1996.
26. D. Ilcinkas, Setting Port Numbers for Fast Graph Exploration, Theor. Comput. Sci. 401(1-3): 236-242, 2008.
27. A. Israeli and M. Jalfon, Token management schemes and random walks yield self stabilizing mutual exclusion, In Proc. *PODC'90*, pp. 119-131.
28. R. Klasing, A. Kosowski, A. Navarra, Taking advantage of symmetries: gathering of asynchronous oblivious robots on a ring. Proc. 12th International Conference on Principles of Distributed Systems, (OPODIS 2008), 446-462.
29. R. Klasing, E. Markou, A. Pelc, Gathering asynchronous oblivious mobile robots in a ring, Theoretical Computer Science 390 (2008), 27-39.
30. A. Kosowski, A. Navarra, Graph Decomposition for Improving Memoryless Periodic Exploration, In Proc. *MFCS'09*, pp. 501-512.
31. D. Kowalski, A. Malinowski, How to meet in anonymous network, Theoretical Computer Science 399 (2008), 141-156.

32. G. Kozma, Z. Lotker, M. Sharir and G. Stupp, Geometrically aware communication in random wireless networks, In Proc. *PODC 2004*, pp. 310-319.
33. E. Kranakis, D. Krizanc, N. Santoro and C. Sawchuk, Mobile agent rendezvous in a ring, Proc. 23rd International Conference on Distributed Computing Systems (ICDCS'2003), 592-599.
34. F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, Geometric ad-hoc routing: theory and practice, In Proc. *PODC 2003*, pp. 63-72.
35. W. Lim and S. Alpern, Minimax rendezvous on the line, SIAM J. on Control and Optimization 34 (1996), 1650-1665.
36. B. Moon, H.V. Jagadish, C. Faloutsos, and J.H. Saltz, Analysis of the Clustering Properties of the Hilbert Space-Filling Curve, *IEEE Transactions on Knowledge Data Engineering* **14(1)**, pp. 124-141, 2001.
37. G. Prencipe, Impossibility of gathering by a set of autonomous mobile robots, Theoretical Computer Science 384 (2007), 222-231.
38. T. Schelling, The strategy of conflict, Oxford University Press, Oxford, 1960.
39. G. Stachowiak, Asynchronous Deterministic Rendezvous on the Line, SOFSEM 2009: 497-508.
40. A. Ta-Shma, U. Zwick, Deterministic rendezvous, treasure hunts and strongly universal exploration sequences., Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), 599-608.
41. L. Thomas, Finding your kids when they are lost, Journal on Operational Res. Soc. 43 (1992), 637-639.
42. B. Xu and D.Z. Chen Density-Based Data Clustering Algorithms for Lower Dimensions Using Space-Filling Curves, In Proc. *PAKDD 2007*, pp. 997-1005.
43. X. Yu and M. Yung, Agent rendezvous: a dynamic symmetry-breaking problem, Proc. International Colloquium on Automata, Languages, and Programming (ICALP'1996), LNCS 1099, 610-621.