Constructing a Map of an Anonymous Graph: Applications of Universal Sequences

Jérémie Chalopin[1,⋆], Shantanu Das[1], and Adrian Kosowski[2]

[1] LIF, CNRS & Aix-Marseille University, France
jeremie.chalopin@lif.univ-mrs.fr, shantanu.das@acm.org
[2] Gdańsk University of Technology, Poland
and LaBRI, INRIA Bordeaux Sud-Ouest, France
adrian@kaims.pl

Abstract. We study the problem of mapping an unknown environment represented as an unlabelled undirected graph. A robot (or automaton) starting at a single vertex of the graph G has to traverse the graph and return to its starting point building a map of the graph in the process. We are interested in the cost of achieving this task (whenever possible) in terms of the number of edge traversal made by the robot. Another optimization criteria is to minimize the amount of information that the robot has to carry when moving from node to node in the graph.

We present efficient algorithms for solving map construction using a robot that is not allowed to mark any vertex of the graph, assuming the knowledge of only an upper bound on the size of the graph. We also give universal algorithms (independent of the size of the graph) for map construction when only the starting location of the robot is marked. Our solutions apply the technique of universal exploration sequences to solve the map construction problem under various constraints. We also show how the solution can be adapted to solve other problems such as the gathering of two identical robots dispersed in an unknown graph.

Keywords: Graph Exploration, Map Construction, Anonymous Networks, Mobile Robot, Universal Exploration Sequences.

1 Introduction

We consider the problem of exploration and mapping of an unknown unlabelled environment by a mobile entity which we call the agent. The environment is usually modelled as a graph where the agent is initially located at any arbitrary node of the graph. The objective of the agent is to build a map of the graph. The graph is anonymous i.e. the nodes of the graph do not have any identifying labels and thus, all nodes of the same degree look identical to the agent. However, the edges incident to a node are locally ordered with a *port numbering* that allows the agent to deterministically choose an edge and traverse along it. Note that if the agent is allowed to somehow mark the nodes that it visits (such

⋆ This author was partially supported by ANR Projects SHAMAN and ECSPER.

that it can recognize them on future visits), then a simple depth-first search suffices to solve the problem. When the agents do not have the capability to mark nodes it is sometimes difficult to solve the map construction problem. A known technique for traversing unlabelled graphs is to use the so-called universal traversal sequences [18]. A universal traversal sequence is a sequence of port numbers such that if the agent traverses the edges of any graph G according to this sequence it is guaranteed to visit all nodes of G irrespective of the topology of G and the port-numbering on G. However, such sequences tend to be very large and thus it is perhaps not the most efficient method of traversing a graph. Moreover, traversing the graph does not necessarily imply that the agent can build a map. In certain cases, when the graph has enough symmetry, it may not be possible to build a map of the complete graph. In this paper, we concerned about the time complexity (or number of moves made by the agent) for building a map in those cases when it is possible to do so. An efficient method for map construction is useful as a basic step for an autonomous agent in solving other tasks in unknown unlabelled environments.

One application of the map construction problem is the task of gathering together two autonomous agents that are dispersed in a unknown environment. This is called the *rendezvous* problem. When the two dispersed entities can not communicate from a distance, solving rendezvous is essential for an exchange of information or for achieving even the simplest form of coordination between the mobile entities. The rendezvous problem belongs to the class of symmetry-breaking problems (e.g. leader election is another such problem) that are central to study of computability in distributed systems. The importance of the problem is evident from the large volume of literature [5,10,13,16,19,11,26] dedicated to solving the problem under various conditions and restrictions.

Even if the agents succeed in building a map of the graph, it may not always be possible to rendezvous. For instance, if the agents are in a ring of even size and they start from diametrically opposite nodes in the ring, then no deterministic algorithm is guaranteed to solve rendezvous in this case. However, if the agents start from any other location (except being opposite to each other) then it is possible to solve rendezvous, as soon as we allow the agents to mark their starting locations [19]. In this paper, we solve rendezvous in anonymous graphs assuming that the starting locations of the agents are marked. However, the agents are not allowed to mark any other vertices during their traversal. Further, the agents may not have any prior information about the graph not even the size of the graph.

Related Work: Previous studies on graph exploration have mostly concentrated on *labelled* graphs (or digraphs), with an emphasis on minimizing the cost of exploration in terms of either the number of moves (edge traversals) or the amount of memory used by the agent. Panaite and Pelc [21] gave an algorithm for exploring labelled undirected graphs that uses $m+O(n)$ moves, improving on the standard *Depth-First Search* algorithm that takes $2m$ moves. On the other hand, Deng and Papadimitrou [12] as well as Albers and Henzinger [1] studied the exploration of strongly connected directed graphs under the same conditions.

There have also been some studies on the efficiency of exploration when some prior information about the graph is available with the agent—for instance, when the agent possesses an unlabelled isomorphic map of the graph [22].

Given an unknown, unlabelled (sometime called anonymous) graph, it is not always possible to construct an exact map of the graph (due to the presence of symmetries). There exists characterizations of anonymous graphs where it is possible to solve the problem [25].

For exploring arbitrary anonymous graphs, various methods of marking nodes have been used by different authors. Bender *et al.* [7] proposed the method of dropping a pebble on a node to mark it and showed that any strongly connected directed graph can be explored using just one pebble, if the size of the graph is known and using $O(\log\log n)$ pebbles, otherwise. Dudek *et al.* [14] used a set of distinct markers to explore unlabeled undirected graphs. In [15] the authors focus on minimizing the amount of memory used by the agents for exploration (however, they do not require the agents to construct a map of the graph). Others have studied the exploration of mazes or labyrinths, which have been shown [8] to be easier to explore than graphs, due to the availability of orientation information.

In the absence of any device for marking nodes, unknown anonymous graphs can still be explored using universal traversal/exploration sequences [18]. Aleliunas et al. [2] showed that there exists universal traversal sequences of polynomial size for all connected graphs of a given size n. A recent result by Reingold [23] showed that universal exploration sequences can be constructed in logarithmic space. Such sequences have been used for solving the rendezvous problem [10,24] though only in the synchronous setting.

The problems of rendezvous and leader election has been extensively studied as symmetry-breaking problems in unknown anonymous graphs, starting from the work of Angluin [4]. Characterizations of the solvable instances for leader election in *message passing* networks of processors, have been provided by Boldi *et al.* [9] and by Yamashita and Kameda [25] among others. Recently, Fusco and Pelc [17] have shown that leader election can be solved if each process has a memory of $O(\log n)$ bits, matching the lower bound given by Ando et al. [3]. The rendezvous problem has been solved under various different assumptions such as distinct labels for the agents, sense of direction information, or prior knowledge of topology (e.g.[5,13,16,19,26]). In the most general setting of unknown anonymous graph with identical agents, the problem was recently solved in [10], though only for synchronous agents. In the asynchronous case, an almost complete solution using distinct labels has been provided in [11]. The idea of solving rendezvous by marking the starting locations with tokens was first proposed by Baston and Gal [6].

Our Results: We study the complexity of map construction in anonymous graphs by a mobile agent that is not allowed to write on the nodes of the graph. We present several polynomial time deterministic algorithms for map construction.

In the model where no vertices of the graph are marked, for the task of map construction to be feasible, the agents must know some bound n on the number of nodes of the graph and some bound $d \leq n$ on its degree. The folklore algorithm based on view construction [25] requires $O(d^n)$ moves by the agent. The recent paper [10] provides more efficient map construction algorithms: a polynomial-time approach (with very high exponent) using small memory, and an $O(n^{10}d^5 \log^2 n)$-time algorithm using $O(n^9 d^4 \log^2 n)$ memory. Herein we put forward two improved algorithms which offer different time/memory tradeoffs:

- a simple algorithm running in $O(n^6 d^2 \log n)$ time, using $O(n^6 d^2 \log n)$ memory (Prop. 6),
- a more advanced algorithm running in $O(n^6 d^3 \log n)$ time, using $O(n^3 d^2 \log n \log d)$ memory (Prop. 7).

In the model in which the agent has no prior knowledge of graph parameters (such as n or d), in order to make the problem feasible, we assume that the starting location of the agent is specially marked. In this case, we show how to guess the value of n and thus solve map construction in polynomial time using an optimal memory ($\Theta(\log n)$) algorithm (Prop. 8). We also present another algorithm which requires slightly more agent memory ($O(nd \log n)$) but is much more efficient in terms of time steps, requiring only $O(n^3 d)$ steps (Prop. 9). Finally, in this model we also show how our algorithms can be extended to solve the rendezvous of two mobile agents in anonymous graphs with marked homebases even in the asynchronous case (Prop. 11).

2 Model, Definitions and Known Results

2.1 Our Model

The environment is represented by a simple undirected connected graph $G = (V(G), E(G))$. The agent starts from a single node of the graph, called the *home-base*. The agent can traverse any edge of the graph incident to its current location. At each node $v \in V(G)$, the edges incident to v are distinguishable to any agent arriving at v. There is a bijective function

$$\lambda_v : \{(v, u) \in E(G) : u \in V(G)\} \to \{0, 1, 2, \ldots d(v) - 1\}$$

which assigns unique labels (port-numbers) to the edges incident at node v (where $d(v)$ is the degree of v). An agent at a node u can choose to leave through any incident edge $e = (u, v)$ simply by specifying the port number $\lambda_u(u, v)$ of the edge. On reaching the node v, the agent knows the port number $\lambda_v(v, u)$ of the edge through which it arrived. The ith successor of a node u, denoted by $succ(u, i)$ is the node v reached by taking port number i from node u (where $0 \leq i < deg(u)$). For any edge (u, v), we use $\lambda(u, v)$ to denote the ordered pair of labels $(\lambda_u(u, v)$,$\lambda_v(u, v))$. A path in G is a sequence of nodes $P = (u_0, u_1, \ldots, u_k)$ such that $(u_j, u_{j+1}) \in E(G)$, $\forall j, 0 \leq j < k$ and the label sequence of path P is $\Lambda(P) = (\lambda(u_0, u_1), \ldots \lambda(u_{k-1}, u_k))$.

The nodes of G do not have visible identities by which a visiting agent can identify them. In other words, nodes having the same degree look identical to the agents. The agents have computing and storage capabilities. When an agent moves from one node to another, it carries with its own local memory which consists of two parts. One part is a write-only stable storage which is used to write the output (we assume it is large enough to store a map of G). The other part is the agent's private memory which is used for remembering the information obtained in previous moves. Our objective is to minimize the private memory of the agent i.e. the amount of information it needs to remember while moving along the graph. When the agent is located at any node of the graph, it has access to a read-write memory which can be used for local computation (but not for storing information). We are not concerned about the cost of performing local computations at node. We are interested in minimizing the total number of edge traversals (steps) made by the agent in achieving its tasks.

2.2 Universal Exploration Sequences

In this paper, we will use the notion of a *Universal Exploration Sequence* (UXS) [18]. Let $(a_1, a_2, \ldots, a_k)$ be a sequence of integers. An *application* of this sequence to a graph G at node u is the sequence of nodes $(u_0, \ldots, u_{k+1})$ obtained as follows: $u_0 = u$, $u_1 = succ(u_0, 0)$; for any $1 \leq i \leq k$, $u_{i+1} = succ(u_i, (p + a_i) \mod d(u_i))$, where p is the port number at u_i corresponding to the edge $\{u_{i-1}, u_i\}$. A sequence $(a_1, a_2, \ldots, a_k)$ whose application to a graph G at any node u contains all nodes of this graph is called a UXS for this graph. A UXS for a class $\mathcal{G}$ of graphs is a UXS for all graphs in this class.

For all feasible pairs of N and D, let $U(N, D)$ be a UXS for the class $\mathcal{G}_{N,D}$ of all graphs with at most N nodes and maximum degree at most D. The following important result, based on a reduction from Kouckỳ [18], is due to Reingold [23].

Proposition 1 ([23]). *For any positive integer n, there exists a UXS $Y(n) = (a_1, a_2, \ldots, a_M)$ for the class $\mathcal{G}_n$ of all graphs with at most n nodes, such that*

- *M is polynomial in n,*
- *for any $i \leq M$, the integer a_i can be constructed using $O(\log n)$ bits of memory.*

The above result implies that a (usually non-simple) path $(u_0, \ldots, u_{M+1})$ traversing all nodes can be computed (node by node) in memory $O(\log n)$, for any graph with at most n nodes. Moreover, logarithmic memory suffices to walk back and forth on this path: to walk forward at node u_i, port $(p + a_i) \bmod d(u_i)$ should be computed when coming by port p, to walk backward, port $(p - a_i) \bmod d(u_i)$ should be computed.

Proposition 2 ([2]). *For any positive integers n, d, $d < n$, there exists a universal exploration sequence of length $O(n^3 d^2 \log n)$ for the family of all graphs with at most n nodes and maximum degree at most d.*

Note that the exploration sequences in the proposition above are not constructible in logarithmic memory, while the log-space constructible sequences from Proposition 1 are much longer (though still polynomial in n).

2.3 The Map Construction Problem

As mentioned before, the problem of reconstructing the topology of a network of processors has been studied before, notably in [25]. That paper introduced the concept of the *view* of a node in a graph, which we restate below:

Definition 1 ([25]). *The view $\mathcal{V}_{G,\lambda}(v)$ of node v, in a graph G with port-numbering λ, is an infinite edge-labelled rooted tree T, whose root represents the node v and for each neighboring node u_i of v, there is a vertex x_i in T and an edge from the root to x_i with the same labels as the edge from v to u_i in G. The subtree of T rooted at x_i is again the view $\mathcal{V}_{G,\lambda}(u_i)$ of the node u_i.*

We shall drop the subscript λ when it is obvious from the context.

Proposition 3 ([20]). *Given any simple graph G with n nodes and a port-numbering λ, two vertices $u, u' \in V(G)$ have the same view (i.e. $\mathcal{V}_G(u) = \mathcal{V}_G(u')$) if and only if the views truncated to a depth of n are equal (i.e. $\mathcal{V}^n_G(u) = \mathcal{V}^n_G(u')$).*

If two nodes of a graph have identical views then these nodes are said to be equivalent to each other. If the nodes of the graph are grouped into classes such that two nodes are put in the same class if and only if they have the same view, then such a classification is an equivalence partition of $V(G)$, where all classes have the same size. Based on this partitioning, the *quotient graph* of G is defined as follows.

Definition 2 ([25]). *Given an undirected connected graph G with port-numbering λ, the* quotient graph *H is an edge-labelled multigraph such that there exists a homomorphism φ from G to H satisfying the following: (i) For any two nodes u and v, $\varphi(u) = \varphi(v)$ if and only if $\mathcal{V}_G(u) = \mathcal{V}_G(v)$, (ii) For each edge (u, v) of G, there is an edge $(\varphi(u), \varphi(v))$ in H labelled with $\lambda(u, v)$ and (iii) H has no other edges.*

If two graphs G_1 and G_2 have identical quotient graph then it is not possible to distinguish between them by just traversing them (without making any marks on the graph). Any deterministic algorithm executed on G_1 would produce the same output as the same deterministic algorithm executed on G_2. Thus, for such graphs, it is not possible to reconstruct an exact copy of the graph. In fact the maximum information that can be obtained by an agent traversing the graph, is represented by the quotient graph.

Definition 3. *We define the Map Construction problem as follows. Given an undirected connected graph G with port-numbering λ, an agent starting at any node of G has to build the (edge-labelled) multigraph H such that H is the quotient graph of (G, λ).*

Note that if G has no symmetry (i.e. when all nodes have distinct views) then the quotient graph of G is G itself. Thus for these cases, the maps constructed by our algorithms would be the exact copy of G.

Finally we present a well known impossibility result for the rendezvous problem.

Proposition 4 ([9,25]). *Given a graph G with a port-numbering λ, the deterministic rendezvous of two agents is impossible if the starting location of the two agents have the same view.*

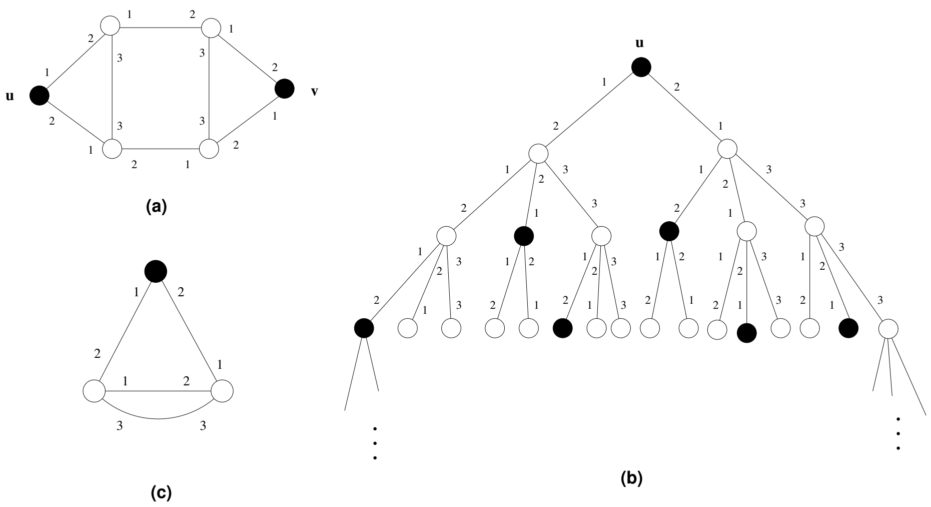

Fig. 1. (a) An example graph containing two agents initially at the marked nodes. The view of each agent is shown in (b) while the quotient graph is shown in (c).

3 Map Construction with Knowledge of Upper Bound

In this section, we assume that the agent has prior knowledge of n, the size of the graph. In fact, in all our algorithms, the value of n can be replaced by any upper bound $N \geq n$.

A polynomial-time approach for solving the map construction problem can be obtained by applying a subroutine from [10], which, for any given starting node, computes an integer in the range $[1, n]$ which is a unique identifier of the node in the quotient graph. In this way, the map exploration problem can be solved by performing a DFS exploration of the graph and computing the identifiers of the endpoints of all the traversed edges. The claim below follows.

Proposition 5 ([10]). *The map construction problem can be solved in time $O(nd \cdot T(n, d))$, where $T(n, d)$ is the (polynomial) time complexity of computing the identifier of a node in a graph of order n and degree d.*

Using the routines from [10], the operation of computing the identifier is extremely time consuming. When the agent is equipped with only $O(\log n)$ memory, we have $T(n, d) = O(|U(n^2, d)|^2 |U(n, d)|^2)$, where the used exploration sequences need to be logarithmically constructible. It is possible to implement the signature detection routines in $T(n,d) = O(|U(n^2,d)||U(n,d)|)$ steps, but using memory of the same order as the number of steps. Thus, for the best known upper bounds on the length of exploration sequences, this means that the map is constructed in $O(n^{10}d^5 \log^2 n)$ time and $O(n^9 d^4 \log^2 n)$ memory.

In this section we put forward two algorithms which solve the map construction problem more efficiently. The first relies on the intriguing property that the traversal of a sufficiently long exploration sequence is sufficient to identify the graph. The second uses UXS-s in a completely different way.

3.1 Using a UXS as a Sequence for Graph Identification

Suppose that a fixed sequence $Y = (a_1, a_2, \ldots, a_M)$ applied at a node $u = u_0$ of graph G results in the traversal of G visiting the nodes $(u_0, u_1, \ldots, u_{M+1})$. The *signature* of node u is the sequence of edge labels which are traversed by an application of the sequence in graph G starting at node u: $S_{(Y,G)}(u) = (\lambda(u_0,u_1), \ldots, \lambda(u_M, u_{M+1}))$.

The results in [10] provide a constructive criterion for distinguishing the views of two vertices of a graph G based on the signatures of vertices. In fact, by a minor modification of their proof, we obtain a method for distinguishing the views of vertices in any two (not necessarily identical) graphs, and we have the following result.

Lemma 1. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs with at most n nodes. Then, for any nodes $u_1 \in V_1$, $u_2 \in V_2$, we have $\mathscr{V}_{G_1}(u_1) \neq \mathscr{V}_{G_2}(u_2)$, if and only if $S_{(Y,G_1)}(u_1) \neq S_{(Y,G_2)}(u_2)$, where $Y = U(2n^2, d)$.*

Proposition 6. *There exists an algorithm for map construction which runs in $|U(2n^2,d)|$ steps and requires $O(|U(2n^2,d)|)$ memory.*

Proof. The algorithm proceeds by performing a traversal of the sequence $Y = U(2n^2,d)$, starting from the agent's homebase u. The agent records successive labels encountered during its traversal, thus computing the signature $S_{(Y,G)}(u)$. Based on this, the agent computes its quotient graph (using local computations only), as the smallest graph $G' = (V', E')$ with distinguished node $u' \in V'$ such that $S_{(Y,G')}(u') = S_{(Y,G)}(u)$. Such a graph must exist, since the tested property is by Lemma 1 equivalent to the condition $\mathscr{V}_G(u) = \mathscr{V}_{G'}(u')$, which is satisfied by a non-empty family of graphs, having a unique element which is smallest in terms of the number of nodes. This element is precisely the quotient graph of G, which completes the proof. $\square$

3.2 An Algorithm with Efficient Identification of Nodes

We now present an algorithm to solve map construction more efficiently. Our algorithm uses ideas that are usually used to minimize a deterministic automaton.

Given a graph G and node u of G and a sequence of edge-labels $Y = ((p_1,q_1),(p_2,q_2),\ldots,(p_j,q_j))$, we say that Y is *accepted* from u if there exists a path $P = (u = u_0, u_1, \ldots, u_j)$ in G such that $\Lambda(P) = Y$, i.e. for each i, $1 \leq i \leq j$, $(p_i, q_i) = \lambda(u_{i-1}, u_i)$. For any $k > 0$, two vertices u, v that have the same view up to depth k are said to be k-equivalent; we denote it by $u \sim_k v$. The k-class of u is the set of all vertices that are k-equivalent to u and this set is denoted by $[u]_k$. Given any two distinct k-classes C, C', a (C, C')-*distinguishing path* is a sequence of edge-labels $Y_{C,C'} = ((p_1, q_1), (p_2, q_2), \ldots, (p_j, q_j))$ of length at most k such that $Y_{C,C'}$ is accepted from each node $u \in C$ and it is not accepted from any node $v \in C'$. For any two distinct k-classes, there always exists either a (C, C')-distinguishing path or a (C', C)-distinguishing path.

To compute the quotient graph of G, it suffices to visit every node v of G and identify the n-class of v and each of its neighbors. Recall from Proposition 3 that $[u]_n = [u]_\infty = [u]$ for any node $u \in G$. Once these equivalence classes are known, one can construct the quotient graph H as follows. The vertices of H are the equivalence classes, and there is an edge labelled by (p, q) from $[u]$ to $[v]$ in H if and only if, u has a neighbor $v' \in [v]$ such that $\lambda_u(u, v') = p$ and $\lambda_v(v', u) = q$.

We present an algorithm (See Algorithm 1) that iterates over k, and for each k, explores the graph and identifies the k-classes of the visited nodes and their neighborhoods. We use a UXS $U(n, d)$ of size $O(n^3 d^2 \log n)$ for the traversal.

For $k = 1$, it is easy to determine the k-class of any node v by traversing each edge incident to v and noting the labels. From this information, one can find the distinguishing paths for any pair of 1-classes. For $k \geq 2$, it is possible to identify the k-classes and the corresponding distinguishing paths (from knowledge of the $k - 1$ classes) using the properties below.

Lemma 2. *For $k \geq 2$, two nodes u and v belong to the same k-class, if and only if (i) u and v belong to the same 1-class and (ii) for each i, $0 \leq i \leq deg_G(u) = deg_G(v)$, the ith neighbor u_i of u and the ith neighbor v_i of v belong to the same $(k-1)$-class and $\lambda(u, u_i) = \lambda(v, v_i) = (i, j)$, for some $j \geq 0$.*

Proposition 7. *Algorithm 1 solves map construction for any graph of size n in $O(|U(n, d)| \cdot n^3 d)$ moves and requires $O(n^3 \log n + |U(n, d)| \log d)$ memory.*

Proof. Let n_k be the number of k-classes. During the kth iteration, on each node v reached by the UXS, for each neighbor w of v, the agent computes the $k - 1$ class of w. To do so, it needs to check at most n_k different paths of length $k - 1$. Consequently, for each node v, it needs $O(\deg(v) \cdot n_k \cdot k)$ moves to compute the k-class of v. Thus, during the kth iteration of the algorithm, the agent performs $O(d \cdot n_k \cdot k \cdot |U(n, d)|)$ moves, where d is the maximum degree of the graph. Due to Proposition 3 there are at most n iterations, and $n_k \leq n$; so the total number of moves made by the agent is $O(|U(n, d)| \cdot n^3 d)$.

At the end of the kth iteration, the agent needs to remember the number n_k of k-classes and $n_k(n_k - 1)/2$ distinguishing paths, each of length at most k. This can be stored using $O(n^3 \log n)$ bits. During the kth iteration, the agent needs to remember for each v and for each neighbor w of v, the label of the edge (v, w) and the index of the $(k-1)$-class of w. For each v, it needs $O(\deg(v) \cdot \log n)$ bits. However, the agent does not need to remember the k-class of each v_i, but it is sufficient to identify the distinct k-classes that exist in the graph. Thus, since there are at most n different k-classes, the agent needs $O(n \cdot d \cdot \log n)$ bits of memory to compute the number of k-classes, and to compute the corresponding distinguishing paths using the distinguishing paths for the $(k-1)$-classes. Since the agent can store the UXS using $O(|U(n,d)| \log d)$ bits, the agent can execute this algorithm using $O(n^3 \log n + |U(n,d)| \log d)$ memory. □

Algorithm 1. Class-Refinement(n)

```
Let $v_1, v_2, \dots v_t$ be the sequence of nodes visited by $U(n,d)$, possibly containing
duplicate nodes ;
Apply $U(n,d)$ and **for** *each node* $v_i$ **do**
    Store the labels of each edge incident to $v_i$;
Compute the number of 1-classes and store a distinguishing path for each pair
of distinct classes ;
k := 2;
**repeat**
    Apply $U(n,d)$ and **for** *each node* $v_i$ **do**
        **for** *each edge* $(v_i, w)$ *incident to* $v_i$ **do**
            Compute the $(k-1)$-class of $w$ (using the distinguishing paths);
            Store the label of $(v_i, w)$ and the index of the $(k-1)$-class of $w$ ;
    Compute the number of $k$-classes and store a distinguishing path for each
    pair of distinct $k$-classes ;
    Increment k;
**until** *the number of* $k$*-classes is equal to the number of* $(k-1)$*-classes* ;
Compute the quotient graph ;
```

4 Universal Algorithms for Map Construction

In this section, we assume that the agents do not know the size of the graph G and we are interested in designing universal algorithms that work for graphs of any size. Note that it is not possible to perform exploration with stop in unlabelled graphs of arbitrary size and topology. No terminating algorithm can guarantee to visit all the nodes of an arbitrary connected graph with unmarked nodes. To get around this problem, we assume that the starting location of an agent is specially marked, so that it can be distinguished from the other nodes. This is a much weaker assumption compared to allowing the agent to have a pebble which it can drop at any node and later retrieve it. However this weak assumption is sufficient for obtaining universal algorithms for the map construction problem.

4.1 Guessing the Value of n

The universal exploration sequences used in the previous section used the order of the graph as input. If this information is not available, we can try to guess a value of an upper bound N on n. If the assumed value of N is not big enough, we may not be able to explore the entire graph using $U(N,N)$. The idea is to detect this fact and increase the value of N and try again. Eventually, we would reach a correct upper bound on the size of the graph. In this case, any of the algorithms from the previous section can be applied to solve the map construction problem.

The first of the proposed approaches is implementable in logarithmic space.

Proposition 8. *There exists an algorithm for an agent with a marked homebase which computes an upper bound $N \geq n$ on the order of the graph, $N \in \text{poly}(n)$, using $O(\log n)$ memory.*

Proof. Let K be a parameter which is initially set as 1 and doubled in successive iterations of the algorithm. The idea of the proof is to detect in each iteration whether the universal exploration sequence $U(K, K)$, starting from the homebase r of the agent has visited all nodes of the graph G. The considered UXS is obtained through Reingold's log-space construction [23]. For the smallest value of parameter K such that $U(2K, 2K)$ explores G, and $U(K, K)$ does not, we have that $K < n \leq U(2K, 2K)$. Hence, by putting $N = U(2K, 2K)$ we obtain the sought polynomial upper bound on the value of N, since the length of the considered UXS is polynomial in K.

It remains to describe a subroutine which allows the agent to decide if an exploration sequence $U(K, K)$, starting from homebase r, explores the entire graph G. Let $S = (r = u_0, u_1, \ldots, u_M)$ be the sequence of vertices visited during the traversal, and $U = \{u_0, u_1, \ldots, u_M\}$. Observe that since G is a connected graph, the considered traversal does not completely explore G if and only if there exists a node $v \in V \setminus U$ which is a neighbor of some node $u \in U$. The algorithm proceeds by visiting the successive vertices $(u_0, u_1, \ldots, u_M)$ of the exploration sequence. At each node u_i, the agent makes a detour to explore its neighborhood $Nbd(u_i)$. The agent visits successive nodes of this neighborhood, and for each node $v \in Nbd(u_i)$, $v \neq r$, executes a subroutine to decide if $v \in U$. More precisely, when located at v, for successive values of index $j = 1, 2, \ldots, M$, the agent performs a test to decide whether $v = u_j$, and then returns to v. Testing the condition $v = u_j$ is performed by traversing a path starting at v and defined through the sequence of port labels which appear in the traversal $(u_j, u_{j-1}, \ldots, u_0)$. In other words, we follow a reversal of the j-prefix of the exploration sequence $U(K, K)$, starting by leaving node v through the port by which u_j is entered in sequence S. Since each node can be uniquely identified by the sequence of ports appearing on any path leading from the marked homebase r to this node, we have that $v = u_j$ if and only if the traversal of the considered path terminates at the marked node r.

We finally note that since navigating the robot along sequence $U(K, K)$, or any prefix or reversal of $U(K, K)$, only requires $O(\log n)$ memory (cf. [18]), the entire algorithm runs using $O(\log n)$ memory. □

4.2 More Efficient Map Construction

In this section we consider other methods of exploration rather than using an UXS. The fact that the starting node r of the agent is marked and can be distinguished from other nodes, makes it easier to perform an exploration. The agent can perform a breadth-first traversal building a BFS-tree T rooted at r.

During the traversal, whenever the agent explores a new edge and reaches a node v, it checks whether v is same as some node u in its tree. This can be done by successively applying the label-sequences for the back-paths from each node $u \in T$ to the root r, and checking if one of these hits the marked node. Based on this idea, we have an algorithm for building a map of G starting from the single marked homebase in G (See Algorithm 2). The algorithm maintain a BFS-tree T containing the visited nodes and a data structure called ROOT_PATHS that stores the edge-labelled path P in T from any node v to the homebase r. For such a stored path P, Start(P) refers to the node v.

Proposition 9. *There exists an algorithm for map construction for an agent with a marked homebase which runs in $O(n^3 d)$ steps and uses $O(n \cdot d \log n)$ memory.*

Proof. First we show that the Map output by algorithm BFS-Tree-Construction is an exact copy of G and the graph T output by the algorithm is a spanning tree of G. Note that the sequence of labels on the path from the homebase r to each node in T is unique. Thus no node appears more than once in T. Since the algorithm performs a breadth-first search, every node is reached by the algorithm. If the algorithm does not add a reached node u to T then there is path from u to r which is identically labelled as an exisitng path $P \in$ ROOT_PATHS. Hence by the previous argument u already exists in T. It is easy to see that T is connected and every edge in T appears in G. Thus, T is a spanning tree of G. The Map is a super-graph of T and every edge that is traversed by the algorithm is added to Map (either as tree-edge or as a cross-edge). The algorithm traverses each edge incident to any node in T and thus all edges of G are traversed by the algorithm. Thus we conclude that Map is an isomorphic copy of G.

Whenever the algorithm traverses an unexplored edge at a node v, it has to check at most n paths in ROOT_PATH, each of length at most n. This takes $O(n^2)$ steps for each edge and thus $O(n^3 d)$ steps in total. The agent requires $O(n \cdot d \cdot \log n)$ memory to store Map and T. The data-structure $ROOT\_PATHS$ does not need to be stored explicitly and can be obtained from T. □

4.3 Solving Rendezvous

We now show the above techniques can be used to solve the rendezvous of two dispersed agents in an unknown graph. Note the algorithm BFS-Tree-Construction from the previous section will fail to build a map if there are more than one agents in the graph. If there are two marked nodes in G and an agent can confuse between these two nodes, as they would look identical to the agent. However, if we execute the algorithm BFS-Tree-Construction in a graph with two marked homebases, the following properties would be satisfied.

Lemma 3. *If two agents starting from marked homebases in a connected graph G execute algorithm BFS-Tree-Construction, then the following holds:*
(i) The graph T constructed by each agent would be an acyclic connected (not necessarily spanning) subgraph of G.
(ii) If the maps constructed by the two agents are identical then the views from the two homebases are identical.

Algorithm 2. BFS-Tree-Construction

```
$Map := T := \{r\}$ ;
Add $r$ to Queue;
ROOT_PATHS := $\emptyset$;
while Queue is not empty do
    Get next node $v$ from Queue and go to $v$ using $Map$;
    while node $v$ has unexplored edges do
        Traverse the next unexplored edge $e = (v, u)$;
        for each path $P \in$ ROOT_PATHS do
            Apply sequence $\Lambda(P)$ at node $u$ ;
            if successfully reached a marked node then
                Add to $Map$ a cross-edge from $v$ to Start($P$);
                Update the number of explored edges at the node Start($P$);
                Return to node $v$ using $T$ and exit Loop;
            else
                Backtrack to node $u$ ;
        if All path sequences failed to reach a marked node then
            Add a new node $u$ to $T$ and $Map$ ;
            Add edge $(v, u)$ to $T$ and $Map$ ;
            Insert $u$ to Queue ;
            ROOT_PATHS := ROOT_PATHS $\cup$ Path$_T(u, r)$ ;
            Backtrack to node $v$ ;
```

Proof. (i) An agent executing Algorithm 2 adds a node u to T only if this node does not exist in T (If the node u already belongs to T the agent can correctly detect this fact). Thus result (i) follows from properties of breadth-first search. (ii) The Map constructed by an agent a consists of a BFS-tree (call it T_a) and some cross-edges. The tree T_a is a subgraph of G rooted at the homebase r_a of the agent. If the maps of the two agents are identical then, for every cross-edge (u, v) in the Map of agent a, there is a cross-edge (u', v') in the Map of the other agent (say, agent b) such that either (u, v) and (u', v') are actual edges in G, or (u, v') and (u', v) are edges in G. It is possible to build the view of the agent a using the information contained in its Map (and the fact that the two Maps are identical). We replace each cross-edge (u, v) in the Map, by an edge (u, u_v) and a new node u_v, and plug in a copy of Map re-rooted at v at the new node u_v. We can repeat this recursively from the top level down to any depth N until there are no cross-edges up to depth N. Finally, for each tree edge (x, y) where x is the parent of y, we can add an edge (y, y_x) and a new node y_x and attach a copy of the current Map re-rooted at x at the new node y_x. Using this process recursively, one can obtain the view of agent a up to any desired depth N. Hence we conclude that the views of the two agents are identical if and only if the Maps obtained by Algorithm 2 are identical. □

Due to the above results and Proposition 4, we know that when the maps obtained by the two agents are identical, then rendezvous is not solvable deterministically. So, we only need to consider the case when the maps are distinct. In this case if we could compare the maps of the agents, we can elect one of the agents and the agents could rendezvous at the homebase of the elected agent.

The map constructed by an agent is a rooted edge-labelled graph, where the edge-labelling is a port-numbering on G. There exists a total ordering on the family of such graphs. In the following we will use a fixed ordering on this family of graphs and we say $M_1 < M_2$, if M_1 is distinct from M_2 and appears earlier than M_2 in this fixed ordering. We now present an algorithm for rendezvous of the two agents using the algorithm BFS-Tree-Construction as a basic step, followed by comparison of the maps (See Algorithm 3).

Algorithm 3. Universal-RDV

```
$(T, Map)$ := BFS-Tree-Construction();
Let ROOTPATHS be the set of paths obtained during the algorithm;
Traverse Map and **for** *each cross-edge* $e = (u, v) \in Map$ **do**
    Apply the sequence $\lambda(u, v)$;
    Apply the sequence for the path $P \in$ ROOTPATHS that starts at $v$;
    // The agent has reached some marked homebase
    $(T_2, Map2)$ := BFS-Tree-Construction();
    **if** $Map2 < Map$ **then**
        Traverse tree edges from current node to reach root of $Map2$;
        **Terminate**;
    **else if** $Map2 > Map$ **then**
        Traverse tree edges from current node to reach node $v$;
        Apply the sequence $\lambda(v, u)$;
        Apply the sequence for the path $P \in$ ROOTPATHS that starts at $u$;
        **Terminate**;
Output: "Rendezvous is not solvable";
```

Proposition 10. *Algorithm Universal-RDV solves rendezvous of two agents in any connected graph G with marked homebases, whenever it is deterministically possible and otherwise detects failure.*

Proof. The algorithm constructs a map using Algorithm 2 as a sub-procedure and then compares it with the map of the other agent. Since the procedure BFS-Tree-Construction is deterministic, the map of the other agent can be obtained by simply executing algorithm 2 from the homebase of the other agent. So we need to show that the algorithm succeeds in reaching the other homebase. Suppose T_a and T_b be the two trees constructed by the two agents a and b and r_a and r_b be the corresponding homebases. Since each node is included in one of the two trees, there exists a node v in T_a that is adjacent to some node w in T_b. When agent a explored the neighborhood of v, the neighbor w was not added to T_a. This implies that there must be a node $u \in T_a$, such the path from u to r_a is identically labelled as the path from w to r_b. In other words there is a cross-edge (v, u) in Map_a that corresponds to an actual edge (v, w) in G. Thus, when the agent a traverses this cross-edge and follows the path to the root, it will reach the homebase of the other agent. Note that the agent does not know which path leads to the other homebase, so it must repeat this process for each cross-edge in its Map.

If the maps from the two homebases are distinct, the agents can always agree on a rendezvous location by comparing the maps. The algorithm fails only if the two maps are identical. In that case, we know that rendezvous is not solvable due to Lemma 3 and Proposition 4. □

Proposition 11. *Any execution of Algorithm Universal-RDV on a graph of size n and maximum degree d by two agents, requires $O(n^4 d^2)$ moves by each agent. Each agent requires a private memory of size $O(nd \log n)$.*

Proof. If there are n nodes in the graph, then the Map of an agent can contain at most n nodes. The map construction process requires $O(n^3 d)$ steps as before. However the process is repeated for each cross-edge in the Map. Each cross-edge corresponds to a distinct edge in G, thus there can be at most $n \cdot d$ cross-edges. Hence the result follows. The agent stores the Map in its memory, which requires $O(nd \log n)$ memory space. □

Note that the algorithm presented here solves rendezvous with detect in the asynchronous case (in contrast to [10]). In case the agents possess only logarithmic memory, we can use the techniques from Section 4.1 to obtain a log-space algorithm for solving rendezvous with detect, in the same setting.

References

1. Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM Journal on Computing 29(4), 1164–1188 (2000)
2. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: 20th Annual Symposium on Foundations of Computer Science (FOCS 1979), pp. 218–223 (1979)
3. Ando, E., Ono, H., Sadakane, K., Yamashita, M.: The space complexity of leader election in anonymous networks. International Journal of Foundations of Computer Science 21(3), 427–440 (2010)
4. Angluin, D.: Local and global properties in networks of processors. In: 12th Symposium on Theory of Computing (STOC 1980), pp. 82–93 (1980)
5. Barrière, L., Flocchini, P., Fraigniaud, P., Santoro, N.: Rendezvous and election of mobile agents: impact of sense of direction. Theory of Computing Systems 40(2), 143–162 (2007)
6. Baston, V., Gal, S.: Rendezvous search when marks are left at the starting points. Naval Research Logistics 48(8), 722–731 (2001)
7. Bender, M., Fernández, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: Exploring and mapping directed graphs. In: 30th ACM Symposium on Theory of Computing (STOC 1998), pp. 269–278 (1998)
8. Blum, M., Kozen, D.: On the power of the compass (or, why mazes are easier to search than graphs). In: 19th Annual Symposium on Foundations of Computer Science (FOCS 1978), pp. 132–142 (1978)

9. Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 33–47. Springer, Heidelberg (2001)
10. Czyzowicz, J., Kosowski, A., Pelc, A.: How to meet when you forget: log-space rendezvous in arbitrary graphs. In: 29th Annual ACM Symposium on Principles of Distributed Computing (PODC 2010), pp. 450–459 (2010)
11. Czyzowicz, J., Labourel, A., Pelc, A.: How to meet asynchronously (almost) everywhere. In: 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 22–30 (2010)
12. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. Journal of Graph Theory 32(3), 265–297 (1999)
13. Dessmark, A., Fraigniaud, P., Kowalski, D.R., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica 46(1), 69–96 (2006)
14. Dudek, G., Jenkin, M., Milios, E., Wilkes, D.: Robotic exploration as graph construction. Transactions on Robotics and Automation 7(6), 859–865 (1991)
15. Fraigniaud, P., Ilcinkas, D.: Digraphs exploration with little memory. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 246–257. Springer, Heidelberg (2004)
16. Fraigniaud, P., Pelc, A.: Deterministic rendezvous in trees with little memory. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 242–256. Springer, Heidelberg (2008)
17. Fusco, E.G., Pelc, A.: How much memory is needed for leader election. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 251–266. Springer, Heidelberg (2010)
18. Koucký, M.: Universal traversal sequences with backtracking. Journal of Computer and System Sciences 65(4), 717–726 (2002)
19. Kranakis, E., Krizanc, D., Santoro, N., Sawchuk, C.: Mobile agent rendezvous in a ring. In: 23rd International Conference on Distributed Computing Systems (ICDCS 2003), pp. 592–599 (2003)
20. Norris, N.: Universal covers of graphs: isomorphism to depth n–1 implies isomorphism to all depths. Discrete Applied Mathematics 56(1), 61–74 (1995)
21. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. Journal of Algorithms 33(2), 281–295 (1999)
22. Panaite, P., Pelc, A.: Impact of topographic information on graph exploration efficiency. Networks 36(2), 96–103 (2000)
23. Reingold, O.: Undirected connectivity in log-space. Journal of the ACM 55(4) (2008)
24. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In: 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 599–608 (2007)
25. Yamashita, M., Kameda, T.: Computing on anonymous networks: Part I - characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems 7(1), 69–89 (1996)
26. Yu, X., Yung, M.: Agent rendezvous: A dynamic symmetry-breaking problem. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 610–621. Springer, Heidelberg (1996)