Christian Seitz · Michael Berger · Bernhard Bauer

Towards a general approach to mobile profile based distributed grouping

Abstract In this paper, we present a new kind of mobile ad hoc application, which we call *mobile profile based distributed grouping* (MoPiDiG), which is a combination of mobile clustering and data clustering. In MoPiDiG, each mobile host is endowed with a user profile and, while the users move around, hosts with similar profiles are found and a robust mobile group is formed. The members of a group are able to cooperate with each other or attain a goal together. In this article, MoPiDiG is defined and compared with related approaches. Furthermore, a modular architecture and algorithms are presented to build arbitrary MoPiDiG applications.

1 Introduction

Tomorrow's world will be intrinsically ubiquitous and mobile. Ubiquitous computing is a new trend in computation and communication. It is an intersection of several technologies, including embedded devices, service discovery, wireless networking, and personal computing technologies. In an ad hoc network, mobile devices can detach themselves completely from the fixed infrastructure and establish transient and opportunistic connections with other devices that are within communication range. The structure of an ad hoc mobile network could be highly dynamic. The absence of a fixed network infrastructure, frequent and unpredictable disconnections, and power considerations render the development of ad hoc mobile applications a very challenging task.

We present a new mobile ad hoc network application area, which we call *mobile profile based distributed grouping* (MoPiDiG). In MoPiDiG, each mobile host is endowed with a *user profile*. A user profile (short: *profile*) is a comprehensive data collection belonging to a specific object (e.g., a person). A profile consists of a set of parameters defining the configuration of a user-specific application. While the users move around, mobile hosts with similar profiles are found and a mobile group is formed. The participants of a group are able to cooperate or attain a goal together.

The MoPiDiG framework can be applied in a variety of domains. In the intelligent manufacturing area, often, optimal teams must be found to create specialized products. In order to determine these groups, each worker is equipped with a profile with his/her skills as profile entries, and MoPiDiG uses these profile entries to determine the best team for a specific task.

Another possible application of the MoPiDiG framework is to offer personalized guided tours in museums or expositions. In museums or other cultural facilities, guided tours are offered in which an expert tells some background information about the exhibits. Unfortunately, such guided tours are not customized to the participants. With mobile grouping, the visitors can be grouped according to their interests and preferences, which allows the guide to make a more personalized tour.

In the public transport area, MoPiDiG could be applied in a taxi sharing scenario. If a train arrives at a railway station or an aeroplane at an airport, the different passengers may have the same destination, e.g., a hotel. This destination address is part of the users' profiles and are stored on their mobile devices. While people are waiting at the baggage terminal, the mobile devices exchange profiles and try to find small groups of people with similar destinations. This scenario can even help people to get group rates in public transportation.

C. Seitz (✉) · M. Berger
Corporate Technology,
Information and Communication,
Siemens AG, 81730 München, Germany
E-mail: Christian.Seitz@mchp.siemens.de

B. Bauer
Institute of Computer Science,
University of Augsburg,
86135 Augsburg, Germany

The rest of the paper is organized as follows. Section 2 gives an overview of related work of other clustering or grouping problems. Section 3 formally defines a MoPiDiG problem. The next section presents the architecture of a MoPiDiG application. Section 5 describes the used algorithms in our approach to MoPiDiG and presents simulation results. Finally, Sect. 6 concludes the paper with a summary.

2 Problem classification and related work

In this section, we classify with which problems a MoPiDiG application is confronted. Furthermore, we show which other research areas are related and how this new problem has already been discussed in the literature.

2.1 Problem classification

In MoPiDiG, mobile hosts are equipped with wireless transmitters, receivers, and a user profile. They are moving in a geographical area and are forming an ad hoc network. In this environment, hosts with similar profiles have to be found. MoPiDiG in ad hoc environments comprises three main problems that have to be solved to accomplish a MoPiDiG application. The first problem is the dynamic behavior of an ad hoc network, where the number of mobile hosts and communication links constantly changes. Secondly, a data structure for the user profile has to be defined and a mechanism must be created to compare the profile instances. Finally, similar profiles have to be found in the ad hoc network and the corresponding host needs to form a group, in spite of the dynamic behavior of the ad hoc network.

2.2 Related research areas

Grouping algorithms and their applications appear very often in the literature. There are mainly two different research areas associated with it; namely, mobile networks and databases and data mining. Grouping in mobile networks describes the partitioning of a mobile network in several, mostly disjoint, clusters [2, 14]. The clustering process comprises the determination of a cluster head in a set of hosts. A cluster is a group of hosts, able to communicate with the cluster head. This clustering takes place at the network layer and is used for routing purposes.

In the following, we will have a closer look at these two research areas. Clustering is also known in the database or data mining area. A huge amount of data is scanned with the goal to find similar data sets. This research is also known as unsupervised learning. In the surveys of Fasulo [8] and Fraley and Raftery [3], an overview of many algorithms for that domain can be found. Maitra [11] and Kolatch [6] examine data clusters in distributed databases.

MoPiDiG combines the two aforementioned clustering approaches and, in order to accomplish its objective, the mentioned problems are to be solved. The problems, arising by means of the motion of the hosts, could be solved by methods used in the mobile network area. Searching for similar profiles is based on algorithms of data clustering. Both methods must be adapted to MoPiDiG, e.g., while in the database area millions of data sets must be scanned, in the MoPiDiG application, at the utmost, 100 other hosts are present. In contrast to data sets in databases, ad hoc hosts move around and are active, i.e., they can publish their profile by themselves.

2.3 Related work

There are other works that analyze ad hoc clustering or grouping algorithms. Roman et al. [9] deal with consistent group membership. They assume that the position of each host is known by other hosts, and two hosts only communicate with each other if it is guaranteed that, during the message exchange, the transmission range will not be exceeded. In our environment, obtaining position information is not possible because such data is not always available, e.g., inside of buildings. Hatzis et al. [10] describe algorithms for mobile ad hoc networks, but they assume that the number of hosts does not change during protocol execution. This assumption appears as too strict because vanishing and appearing hosts are an ad hoc networks characteristic.

Another related field is agreement or consensus algorithms, where all hosts must agree on a *binary value*, based on the votes of each host. They must all agree on the same value, and that value must be the vote of at least one process. This is almost what we want to achieve, but in MoPiDiG, we have to agree on complex profiles. Furthermore, not all hosts must agree in our case—it is enough when a subset agrees. Badache et al. [1] adapt agreement algorithms to a mobile environment, but they use fixed base stations and no real ad hoc network without an infrastructure.

3 MoPiDiG definitions and assumptions

This section gives a formal introduction to MoPiDiG and defines the used ad hoc network model. Finally, some assumptions to the application are made.

3.1 The ad hoc network model

An ad hoc network is generally modeled as an undirected graph $G=(V, E)$.

Definition 1. Graph $G(V, E)$ Let $G=(V, E)$ be a graph with the vertex set $V=\{v_1, v_2,..., v_n\}$ and edges

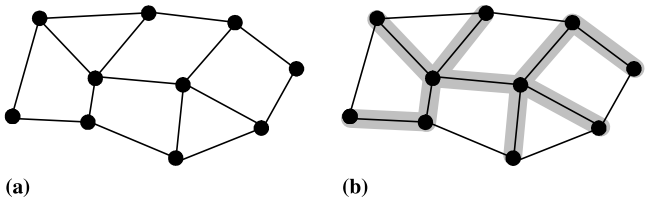

Fig. 1 **a** A graph and **b** a possible spanning tree

$E \subseteq V \times V$ $((v_i, v_j) \in E$ if the Euclidian distance $d(v_i, v_j) \leq r_t$[1].

A graph is depicted in Fig. 1a. The vertices v_i of G_0 represent mobile hosts. Due to the motion of the vertices, a graph G_0 as shown in Fig. 1a is only a snapshot of an ad hoc network, because, as a consequence of the mobility of the hosts, G will constantly be changing.

Assumptions on the mobile nodes and network are:

- We do not rely on any central component.
- There is no location information available, e.g., GPS data or cell information.
- Each mobile device has a permanent, constant unique ID.
- The transmission range of all hosts is r_t. This also guarantees a symmetrical communication behavior, i.e., if host A can send messages to host B, then host B can send messages to host A as well, even though, physically, it is easy to envision circumstances in which some hosts may be able to reach much further than others.
- Each host knows all its neighbors[2] and its associated ID. This service is accomplished by the ad hoc middleware.

3.2 Formal definition of MoPiDiG

In a MoPiDiG application, each host has a user profile. A profile is a point in the profile space Π.

Definition 2. Profile space Let $\Pi = \Pi_1 \times ... \times \Pi_m$ be a profile space with finite dimension m. A point $P \in \Pi$ with $P = (p_1, ..., p_m)$ corresponds to a profile. The p_i are referred as profile entries.

Furthermore, let $\phi : V \rightarrow \mathcal{P}$ be a function that maps to each node $v_i \in V$ to a profile P_j, i.e., $\phi(v_i) = P_j$.

The algorithms in Sect. 5 distinguish between a local group and a decentralized group, which are defined in the following.

Definition 3. Communication relation $a\ \mathcal{C}_n\ b$: The relation $a\ \mathcal{C}_n\ b$ indicates that a node a can communicate with a node b over a maximum of n vertices, i.e., a path $\{e_1, ..., e_m\}$ $(m \leq n)$ from node a to node b with $|\{e_1, ..., e_m\}| \leq n$ exists.

Definition 4. Neighbor set N_v^k The neighbor set N_v^k is defined as $N_v^k = \{x | v\ \mathcal{C}_k\ x\}$. Thus, N_v^k contains the set of nodes that can communicate over k edges with the node v.

Definition 5. Similarity operator σ Let $\mathcal{K}$ be an r-tuple of nodes V of the graph G and let v be an arbitrary node of G. Thus, σ is defined as σ: $V^r \times V \rightarrow \{\text{true, false}\}$.

Definition 6. Local group $\mathcal{G}$ Let N_v^1 be the set of communication partners of an arbitrary node v_i over 1 edge and let $P(N_v^1)$ be the power set of N_v^1 and let $K = \{M \in P(N_v^1) | \sigma(v_i, M) = \text{true}\}$. Then, a local group $\mathcal{G} \subseteq N_{v_i}^1$ is defined as $\mathcal{G} = \max_{\forall x \in K} |X|$. The set of all n local groups $\mathcal{G}$ is denoted with G.

The local group $\mathcal{G}$ of a vertex v consists of all the direct neighbor hosts whose profiles are similar to the profile of v.

In order to achieve a decentralized group, local groups must be combined to get a new group. This is done by a combination operator.

Definition 7. Combination operator γ Let $\mathcal{G}_i$ and $\mathcal{G}_j$ be two local groups and v be an arbitrary node in G. For the combination $\gamma(\mathcal{G}_i, \mathcal{G}_j) = \mathcal{G}_i \cup \mathcal{G}_j \backslash M \subset \mathcal{G}_i \cup \mathcal{G}_j$, the following conditions must hold:

$$|\gamma(\mathcal{G}_i, \mathcal{G}_j)| > |\mathcal{G}_i| \wedge |\gamma(\mathcal{G}_i, \mathcal{G}_j)| > |\mathcal{G}_j|$$
$$\sigma(v, \gamma(\mathcal{G}_i, \mathcal{G}_j)) = \text{true}$$

Definition 8. Decentralized group Let N_v^k be all the neighbors of a node v. A decentralized group is obtained if the local group of the node v is combined with the local group of each element in N_v^k.

4 MoPiDiG application architecture

In this section, the architecture of a MoPiDiG application is presented, which is depicted in Fig. 2. A MoPiDiG application consists of three essential parts: the

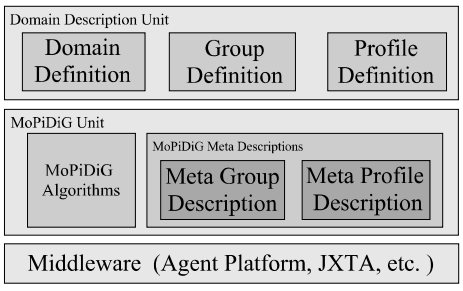

Fig. 2 Components of a MoPiDiG application

[1] $r_t \triangleq$ transmission radius.

[2] The neighbors of a host h are all other hosts that reside within h's transmission range r_t.

middleware, the MoPiDiG unit and the domain description unit.

The middleware establishes the basis for a MoPiDiG application. It is in charge of detecting other hosts in the mobile environment and provides a mechanism for sending and receiving messages to other hosts which are within transmission range. The central element of a MoPiDiG application is the MoPiDiG unit. It is made up of a MoPiDiG algorithm entity and a MoPiDiG description entity.

The MoPiDiG algorithm entity distinguishes between *local grouping* and *decentralized grouping*. In order to obtain a decentralized group, a two-tier process is started. At first, each host selects from its neighbor hosts the subset of hosts which ought to be in the group. This set of hosts is called a *local group*. After each host has determined its local group, these groups are exchanged in a second step and a unique decentralized group is acquired.

As the MoPiDiG unit is totally domain independent, the structure of a profile or a group definition must be defined. This is done by the *meta profile description* and the *meta group description*, which are elements of the *MoPiDiG description entity*. The meta group and meta profile description define what a group or profile definition consists of and specify optional and mandatory elements of a group or profile definition. In both meta descriptions, there is a mandatory general part defining the name of the profile or group. The meta profile description encompasses a set of tags for profile entry definitions and specifies the structure of rules that can be declared in the profile definition in the domain description Unit. The meta group description comprises abstract tag definitions for the size of a group, how a group is defined and the properties to become a group member.

On top of the MoPiDiG unit, the *domain description unit* is located. This unit adjusts the MoPiDiG unit to a specific application domain. In the *domain definition* entity, domain-dependent knowledge is described and in the profile and group definition entities the domain-specific profiles and group properties are defined. The *profile definition* specifies the structure of the profile for the domain, in accordance with the meta profile description.

The profile is specified in XML and the structure is defined by an XML schema. In Fig. 3, an example of a profile definition is given for the taxi sharing scenario, as explained in Sect. 1. After specifying personal data (e.g., name) the application-dependent entries are defined. The PROFILE part consists of a concatenation of these entries in a name–value pair manner. For the taxi sharing scenario, the x and y value in the profile specify the destination of the user. The <ENTRYNAME>maxTime simply specifies the time that the user is willing to spend until his/her destination is reached.

The *group* term varies from application to application. A group can be a few people with similar properties (the same hobby, profession, age, etc.), but it also can be a set of machines with totally different capabilities. For that reason, the *group definition* that comprises the characteristics of the group ought to be found. Figure 4 shows an example of a profile description, also for the taxi sharing scenario. In the <ENTRIES> section, the tags are defined, which are the basis for the grouping. In

Fig. 3 Profile definition in the taxi sharing scenario

```
<?xml version="1.0" standalone="yes" ?>
<DOCUMENT>
    <USERPROFILE>
       <NAME>John Smith</NAME>
       ...
       <PROFILE>
        <PROFILEENTRY>
           <ENTRYNAME>xValue</ENTRYNAME>
           <ENTRYVALUE>190</ENTRYVALUE>
        </PROFILEENTRY>
        <PROFILEENTRY>
           <ENTRYNAME>yValue</ENTRYNAME>
           <ENTRYVALUE>251</ENTRYVALUE>
        </PROFILEENTRY>
        <PROFILEENTRY>
           <ENTRYNAME>maxTime</ENTRYNAME>
           <ENTRYVALUE>25</ENTRYVALUE>
        </PROFILEENTRY>
       </PROFILE>
    </USERPROFILE >
</DOCUMENT>
```

```
<?xml version="1.0" standalone="yes" ?>
<DOCUMENT>
    <GROUPDESCRIPTION>
       <ENTRIES>
        <ENTRYNAME>xValue</ENTRYNAME>
          <ENTRYNAME>yValue</ENTRYNAME>
       </ENTRIES>
       <GROUPSIZE>
        <LOWERLIMIT>0</LOWERLIMIT>
        <UPPERLIMIT>3</UPPERLIMIT>
      </GROUPSIZE>
    </GROUPDESCRIPTION>
</DOCUMENT>
```

Fig. 4 Group description in the taxi sharing scenario

this application, there is the xValue and the yValue, which is the destination. Furthermore, the group size is defined, which is in the range from 0 to the upper value of 3. The user can add further constraints to that description, e.g., whether the other passengers should be a smoker or non-smoker, male or female etc.

5 MoPiDiG algorithms

In this section, the architecture of the algorithm entity is presented, the used network model is defined and the algorithms for each layer are shown. The algorithm entity has a layered architecture and encompasses algorithms for initiator determination, virtual topology creation, local grouping and decentralized grouping. Finally, we show some simulation results in order to indicate how stable the generated groups are.

5.1 MoPiDiG algorithm entity

The most important part of a MoPiDiG application is the algorithm entity (AE). The design of this essential entity is shown in Fig. 5. The basis for the algorithm entity is an *ad hoc middleware*. The middleware is needed by the AE in order to send and receive messages in the dynamic environment. Furthermore, the middleware has to provide a lookup service to find new communication partners.

The lowest layer of the AE is the *initiator detection layer*, which assigns the initiator role to some hosts. An initiator is needed in order to guarantee that the algorithm of the next layer is not started by each host of the network. This layer does not determine one single initiator for the whole ad hoc network. It is sufficient if the number of initiator nodes is only reduced.

The *virtual topology layer* is responsible for covering the graph G with another topology, e.g., a tree or a logical ring. This virtual topology is necessary to reduce the number of messages that are sent by the mobile hosts. First experiences show that a tree is the most suitable virtual topology and, therefore, we will only address the tree approach in this paper.

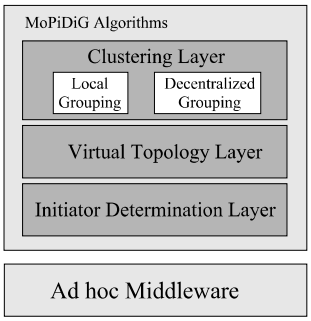

Fig. 5 Architecture of the MoPiDiG algorithm entity

The next layer is the most important one, the *grouping layer*, which accomplishes both the local grouping and the decentralized grouping. Local grouping comprises the selection of hosts that are taken into account for global grouping. Decentralized grouping encompasses the exchange of the local groups, with the goal to achieve a well defined global group (see the definitions in Sect. 4).

5.2 Initiator determination

Before the spanning tree is created, the initiators must be determined, i.e., who is allowed to send the first creation messages. Without initiators, all hosts would start randomly sending messages with the result that a tree will never be created. We are not in search of one single initiator; we only want to guarantee that not all hosts will start the initiation.

There are two ways to determine the initiator; an active and a passive one. The active approach starts an election algorithm (see Malpani et al. [12]). These algorithms are rather complex, i.e., a lot of messages are sent, which is very time consuming. They guarantee that only one initiator is elected and, in case of link failures, that another host takes on the initiator role. Such a procedure is not appropriate and not necessary for MoPiDiG because the initiator is only needed once and it matters little if more than one initiator is present. Therefore, we decided to use the passive determination method, which is similar to Gafni and Bertsekas [5]. By applying the passive method, no message is sent in the beginning to determine an initiator. Since each host has an ID and knows all its neighbor IDs, we only allow a host being an initiator if its ID is larger than all the IDs of its neighbors. The initiator is in charge of starting the virtual topology algorithm, which is described in the next section.

Unfortunately, fairness of the initiator determination process is not guaranteed. Thus, it could happen that a mobile device never takes the initiator position. At this time, it cannot be said whether it is an advantage being the initiator or not.

5.3 Virtual topology creation

Having confined the number of initiators, graph G_0 can be covered with a virtual topology (VT). Simulations showed that a *spanning tree* is a promising approach for a VT and, therefore, we will only describe the spanning tree VT in this paper.

A spanning tree spT(G) is a connected, acyclic subgraph containing all the vertices of the graph G. Graph theory (e.g., Diestel [13]) guarantees that, for every G, a spT(G) exists. Figure 1b shows a graph together with one possible spanning tree.

5.3.1 The algorithm

Each host keeps a spanning tree sender list (STSL). The STSL contains the subset of a host's neighbors belonging to the spanning tree. The initiator, as determined in the previous section, sends a create message furnished with its ID to all its neighbors. If a neighbor receives a create message for the first time, this message is forwarded to all the other neighbors except for the sender of the create message. The host adds each receiver to the STSL. If a host receives a message from a host that is already in the STSL, it is removed from the list. The pseudocode notation of this algorithm is shown in Fig. 6.

To identify a tree, the ID of the initiator is always added to each message. It may occur that a host already belongs to another tree. Under these circumstances, the message is not forwarded any more and the corresponding host belongs to two (or more) trees.

In order to limit the tree size, a hop counter c_h is enclosed in each message and is decremented each time the message is forwarded. If the counter is equal to zero, the forwarding process stops. Note that, with an increasing c_h, the time for building a group also increases because c_h is equivalent to the half-diameter d_G of the graph G.

By using a hop counter, it may occur that a single host does not belong to any spanning tree because all the surrounding trees are large enough, i.e., c_h is reached. The affiliation of that host is not possible because tree nodes do not send messages in case the hop counter's value is zero. When time elapses and a node does notice that it still does not belong to a tree, an initiator determination is started by this host. Two cases must be distinguished. In the first case, the host is surrounded only by tree nodes; in the other case, a group of isolated hosts are existing. In both cases, the isolated host contacts all its neighbors by sending an init message, and, if a neighbor node already belongs to a tree, it answers with a join message. If no non-tree node is around, the single node chooses arbitrarily one of the neighbors and joins the tree by sending a join-agree message; to the other hosts, a join-refuse message is sent. If another isolated host gets the init message, a init-agree message is returned and the host sending the init message becomes the initiator and starts creating a new tree.

5.3.2 Evaluation

The main reason for creating a virtual spanning tree upon the given topology is the reduction of messages needed to reach an agreement. Let n be the number of vertices and e be the number of edges in a graph G. Then, there are $2e-n+1$ messages necessary to create the spanning tree. If no tree would be built and a host receives a message, this message must be forwarded to all its neighbors, which again results in $2e-n+1$ messages for distributing the message through the graph. Overlaying a graph with a virtual spanning tree, the number of forwarded messages is reduced to $n-1$. Determining the factor when a tree becomes more profitable leads us to $A = \frac{2e-n+1}{2(e-n+1)}$. If, on average, $e = 2n$, the amortization A results in $\frac{3n+1}{2n+2}$, which converges to 1.5 for the amortization factor A with increasing n.

In the above equation, the tree maintenance costs are not taken into account. If a new host comes into the transmission range or another host leaves, this is recognized by the ad hoc agent platform and the host is added to or removed from the neighbor list. If the neighbor list has changed, the tree has to be updated. A vanishing host is worse than an appearing one, and could have more negative effects on the tree. If a host is added to the tree, attention should be paid to cycles, which could appear. Therefore, the host is added only once into the STSL to guarantee that no cycle is formed. If a host leaves the ad hoc network, all its neighbors are affected and the vanished host must be deleted from each neighbor host's STSL.

```
Initialization for a node:
  STSL = null;
  initiator = false;
  sent = false;
  root = null;
  visitedHops = 0;

Start (initiator node):
 initiator := true;
 send CREATEMESSAGE to NEIGHBORS;
```

```
Receipt of a CREATEMESSAGE from host p:
  if( not sent )
    root = p;
    STSL += p;
    if( ++visitedHops < HOPS )
     send CREATEMESSAGE to NEIGHBORS;
     sent = true;
    fi;
  fi;
  else STSL -= p;
```

5.4 Local grouping—optimizing the local view

In this section, algorithms are presented that determine the subset of neighbor hosts which initially belong to a host's group, called a *local group*. In order to guarantee that groups are not formed arbitrarily but bring a benefit to its members, a group profit function is defined.

Definition 9. Group profit function f_{GP} Let $P(V)$ be the power set of the nodes of a graph $G(V, E)$ and $\mathcal{G}$ be a local group. The group profit function $f_{\mathrm{GP}}(\mathcal{G}) : P(V) \rightarrow \mathbb{R}$ assigns a value to a group $\mathcal{G} \in P(V)$. This value reflects the benefit that emerges from group formation.

If a new node v_i is added to the group $\mathcal{G}$, $\mathrm{f_{GP}}$ must increase: $f_{\mathrm{GP}}(\mathcal{G} \cup v_i) > f_{\mathrm{GP}}(\mathcal{G})$ in order to justify the addition of v_i.

The algorithm adds in each step exactly one new local group member. Initially, a host scans all the known profiles and adds the one with the smallest distance to him. If the group with two points will bring a greater benefit, the points are added to the point. The group now has two members. It may be that only one point is added in one step because otherwise the shape of a group gets beyond control. A host A can add another host B in exactly the opposite direction to host D being added by host C. If more than one point should be added, coordination is needed.

The points already belonging to the group form a line because, at all times, only one point is added. In order to sustain this kind of line, we only allow the two endpoints to add new points. To coordinate these two points, the endpoints of the line may add new points alternately. If the right end has added a new point in step n, in step $(n+1)$, it is the left side's turn to add a point. The alternating procedure stops when one side is not able to find a new point. In such a case, only the other side continues to add points until no new point is found. If a host is allowed to add a point and there is also one to add, it is not added automatically. The new point must bring a benefit, according to definition 9.

Figure 7 illustrates the process of finding a local group and Fig. 8 contains the pseudocode of the algorithm.

In Fig. 7, a two-dimensional profile space can be seen. The points are the profiles of some users. The dark black point is the point that the local view is to be obtained for. In image (a), the grouping starts. In (b) and (d) on the right side of the line and in (c) and (e) on the left side of the line, new points are added. The last picture (f) represents the complete group. In (f), last point was added on the left side, although it would have been the right side's turn, but there are no points within range, so the left side has to find one.

5.5 Decentralized grouping—achieving the global view

In the previous section, each host has identified its neighbor hosts that belong to its local group g_i. These local groups must be exchanged in order to achieve a global group.

The algorithm presupposes no special initiator role. Each host may start the algorithm and it can even be initiated by more than one host contemporaneously. The core of the used algorithm is an echo algorithm, see [7].

Initially, an arbitrary host sends an EXPLORER message with its local group information enclosed to its neighbors, which are element of the spanning tree (the STSL, see Sect. 5.3). If a message arrives, the enclosed local group is taken and is merged with its current local view of the host to get a new local view. The merging

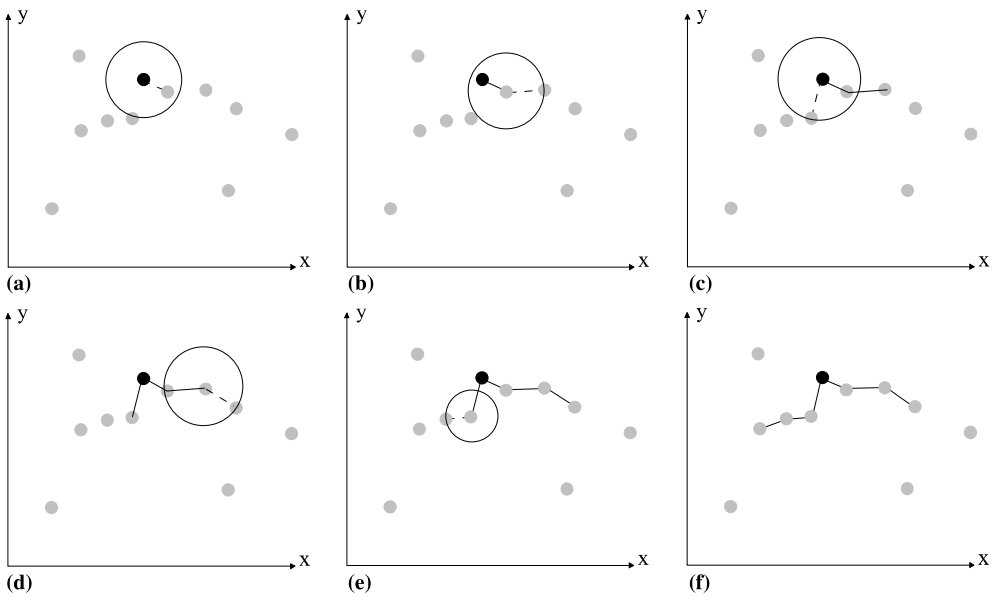

Fig. 7 The local grouping algorithm

Fig. 8 Pseudocode of the local grouping algorithm

```
firstReferenceNode := currentPoint;
secondReferenceNode := currentPoint;
nextPoint := null;
localGroup := currentPoint;

currentProfit = profit_function( localGroup );

while((nextPoint:=getNearestProfilePoint(firstReferenceNode))
                !=null)
  futureProfit := profit_function( localGroup + nextPoint );
  if( futureProfit > currentProfit ) then
    localGroup += nextPoint;
    neighbors -= nextPoint;
    currentProfit = futureProfit;
    firstReferenceNode := secondReferenceNode;
    secondReferenceNode := nextPoint;
  fi;
elihw;
```

function tries to maximize the group profit function, i.e., if two groups are merged, from each group, these members become a member of the new group which together draw more profit than each single group.

The new local view is forwarded to all neighbors except for the sender of the received message. If a node has no other outgoing edges and the algorithm has not terminated, the message is sent back to the sender. If more than one host initiates the algorithm and a host receives several EXPLORER messages, then only the EXPLORER message from the host that has the high ID (message extinction) is forwarded. The pseudocode of the group distribution is shown in Fig. 9. But if the algorithm in Fig. 9 has terminated, it is still not yet guaranteed that each node has the same global view. In the worst case, only the initiator node has a global view. For that reason, the echo algorithm has to be executed once more. In order to save messages, in the second run, the echo messages need not be sent because no further information gain is achieved.

A critical point is to determine the termination of the grouping process. The algorithm terminates in at most $2d_G = 4c_h$ steps and, because the echo messages in the second run are not sent, this is reduced to $3c_h$. If a host receives this amount of messages, the grouping is finished. But, due to the mobility, nodes come and go. Currently, the algorithms stops if a node gets from all its neighbors the same local group information. This local group information is supposed to be the global group information.

To make sure that all group members have the same global view, the corresponding hosts check this with additional confirmation messages. But currently, this part is considered to be optional.

Fig. 9 Pseudocode of the echo algorithm, including the group merging process

```
Start (only if not ENGAGED):

  initiator := true;
  ENGAGED := true;
  N := 0;
  localGroup = getLocalGroup();
  EXPLORER.add( localGroup );
  send EXPLORER to STSL;

Receipt of an ECHO message:

  N := N+1;
  localGroup := merge( localGroup,
              ECHO.getLocalGroup() );
  if N = |STSL| then
    ENGAGED := false;
    if( initiator ) then finish;
     else
      ECHO.setLocalGroup( localGroup );
      send ECHO to PRED;
    fi;
  fi;
```

```
An EXPLORER message from host p is received:

  if ( not ENGAGED )  then
    ENGAGED := true;
    N := 0;
    PRED := p;
    localGroup := getLocalGroup()
    localGroup := merge(localGroup,
             EXPLORER.getLocalGroup());
    EXPLORER.setLocalGroup(localGroup);
    send EXPLORER to STSL-PRED;
  fi;
  N := N+1;
  if ( N = |STSL| ) then
    ENGAGED := false;
    if( initiator ) then finish;
     else
      localGroup := merge(localGroup,
            EXPLORER.getLocalGroup());
      ECHO.setLocalGroup(localGroup);
      send ECHO to PRED;
    fi;
  fi;
```

5.6 Group stability

In this subsection, the stability of the groups is evaluated. By "stability," we mean the time that a group does not change, i.e., no other host is added and no group member leaves the group. This stability duration is very important for our algorithms because, in this time, the group formation must be finished.

We developed a simulation tool to test for how long the groups are stable. The velocity of the mobile hosts is uniformly distributed in the interval [0, 5.2], the average velocity of the mobile hosts is 2.6 km/h. This speed seems to be the prevailing speed in pedestrian areas. Some people do not walk at all (they look into shop windows, etc.), other people hurry from one shop to the other and, therefore, walk faster. Moreover, we assume a radio transmission radius of 50 m.

The left picture in Fig. 10 shows this dependency. The picture shows that the time in which a group is stable decreases rapidly. A group with two people exists for, on average, 30 s, whereas a group with five people is only stable for 9 s. Nevertheless, a group that is stable for 9 s is still sufficient for our algorithms.

The stated times for groups are the times for the worst case scenario, i.e., no group member leaves the group and no other joins the group. But, for the algorithms, it does not matter if a group member leaves during the execution of the algorithm. The only problem is that this person cannot be informed about its potential group membership. In case a person joins the group, the profile information of this point must reach every other point in the group, which, of course, must also occur in the time the group is stable. Unfortunately, we do not have simulation results for groups with 10–20 members.

The group stability is, furthermore, affected by the speed of the mobile hosts. The faster the mobile hosts are, the more rapidly they cross the communication range. In our simulation environment, we analyzed the stability of the groups in relation to the speed, which is shown in the right picture of Fig.10. In this simulation, we investigated for how long a group of three people is stable when different velocities are prevailing. For the simulation, again the chain algorithm is used and the transmission radius is 50 m.

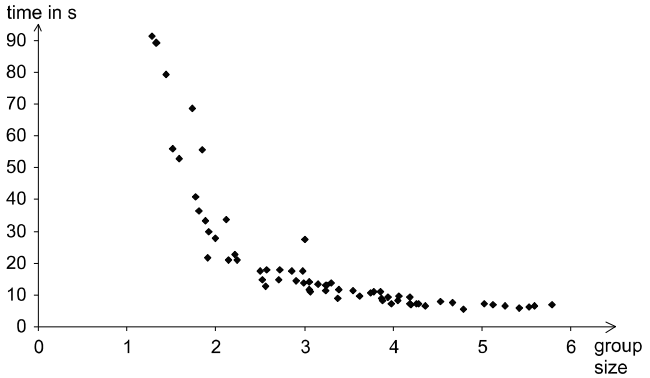

Fig. 10 Graph depicting the time that a group is stable in relation to the group size

The right picture of Fig. 10 shows that the group is stable for more than 40 s when the members have a speed of 1 km/h. The situation changes when the speed increases. If all members walk fast (speed = 5 km/h), the group is only stable for approximately 10 s.

Let r_t be the transmission radius and $\bar{v}$ be the average speed of the mobile peers. Then, the simulations show that:

$$\frac{\pi r}{2\bar{v}} = \text{constant}$$

This expression leads to the following formula for the stability time t_s of a group with n members:

$$t_s = \frac{\pi r}{2\bar{v}} \frac{1}{n-1}$$

6 Conclusion

In this paper, we presented a group of ad hoc applications called mobile profile based distributed groupings (MoPiDiGs). Each mobile host is endowed with its user's profile and, while the user walks around, clusters are found, which are composed of hosts with a similar profile.

The architecture of a MoPiDiG application is shown, which basically is made up of a MoPiDiG description entity that makes the MoPiDiG unit domain independent and an algorithm entity, which is responsible for local grouping and distributed grouping. At first, each host has to find its local group, which consists of all neighbor hosts with similar profiles. Finally, the local groups are exchanged and a global group is achieved. Simulation results show that the groups are stable for long enough to run the algorithms.

We simulated an initial MoPiDiG application, a taxi sharing scenario, where potential passengers with similar destinations form a group [4].

References

1. Badache N, Hurfun M, Macêdo R (1997) Solving the consensus problem in a mobile environment. Technical report no 1146, IRISA, Rennes, France, available at http://citeseer.ist.psu.edu/badache97solving.html
2. Basagni S (1999) Distributed clustering for ad hoc networks. In: Proceedings of the IEEE international symposium on parallel architectures algorithms and networks (ISPAN'99), Fremantle, Australia, June 1999, pp 310–315
3. Fraley C, Raftery AE (1998) How many clusters? Which clustering method? Answers via model-based cluster analysis. Comput J 41(8)578–588
4. Seitz C, Berger M (2003) Towards an approach for mobile profile based distributed clustering. In: Proceedings of the international conference on parallel and distributed computing (Euro-Par 2003), Klagenfurt, Austria, August 2003

5. Gafni EM, Bertsekas DP (1981) Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Trans Commun 29(1):11–18
6. Kolatch E (2001) Clustering algorithms for spatial databases: a survey. Department of Computer Science, University of Maryland, available at http://citeseer.ist.psu.edu/436843.html
7. Chang EJH (1982) Echo algorithms: depth parallel operations on general graphs. IEEE Trans Software Eng 8(4):391–401
8. Fasulo D (1999) An analysis of recent work on clustering algorithms. Technical report no 01- 03-02, Department of Computer Science and Engineering, University of Washington, available at http://citeseer.ist.psu.edu/fasulo99analysi.html
9. Roman G-C, Huang Q, Hazemi A (2001) Consistent group membership in ad hoc networks. In: Proceedings of the international conference on software engineering (ICSE 2001), Toronto, Canada, May 2001, available at http://citeseer.ist.psu.edu/roman01consistent.html
10. Hatzis KP, Pentaris GP, Spirakis PG, Tampakas VT, Tan RB (1999) Fundamental control algorithms in mobile networks. In: Proceedings of the 11th ACM symposium on parallel algorithms and architectures (SPAA'99), Saint-Malo, France, June 1999, pp 251–260, available at http://citeseer.ist.psu.edu/hatzis99fundamental.html
11. Maitra R (1998) Clustering massive datasets. In: Proceedings of the section on statistical computing at the 1998 joint statistical meetings, Dallas, Texas, August 1998, available at http://citeseer.ist.psu.edu/maitra98clustering.html
12. Malpani N, Welch J, Vaidya N (2000) Leader election algorithms for mobile ad hoc networks. In: Proceedings of the 4th international workshop on discrete algorithms and methods for mobile computing and communications (DIALM 2000), Boston, Massachusetts, August 2000. available at http://citeseer.ist.psu.edu/malpani00leader.html
13. Diestel R (2000) Graph Theory. In: Graduate texts in mathematics, vol 173, 2nd edn. Springer, Berlin Heidelberg New York
14. Banerjee S, Khuller S (2000) A clustering scheme for hierarchical control in multi-hop wireless networks. Technical report CS-TR 4103, University of Maryland, available at http://citeseer.ist.psu.edu/banerjee00clustering.html