moPiMine - mobile profile mining

C. Seitz, M. Berger, Bernhard Bauer

Angaben zur Veröffentlichung / Publication details:

Seitz, C., M. Berger, and Bernhard Bauer. 2006. “moPiMine - mobile profile mining.” In *International Workshop on Wireless Ad-Hoc Networks, 2004, 31 May - 3 June 2004, Oulu, Finland, Finland*, 110–14. Los Alamitos, CA: IEEE. https://doi.org/10.1109/iwwan.2004.1525552.

2004 International Workshop on Wireless Ad-Hoc Networks

moPiMine - Mobile Profile Mining

Christian Seitz and Michael Berger
Siemens AG, Corporate Technology
D-81730 Munich, Germany
Email: christian.seitz@mchp.siemens.de
Email: m.berger@siemens.com

Bernhard Bauer
University of Augsburg
Institute of Computer Science
D-86150 Augsburg, Germany
Email: bauer@informatik.uni-augsburg.de

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

I. 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 completely from the fixed infrastructure and establish transient and opportunistic connections with other devices that are in communication range.

We present a new mobile ad hoc network application area, which we call *Mobile Profile Mining* (*moPiMine*). In *moPiMine* 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 to be found and a mobile group is formed. The participants of a group are able to cooperate or attain a goal together.

We apply the *moPiMine* framework in a taxi sharing scenario. If an airplane arrives at an airport some passengers may have the same final destinations e. g. a hotel or destinations are on the route of another person. While people are walking to the exit or waiting at the baggage claim the mobile devices exchange the profiles of their users and try to find small groups with similar destinations. This scenario can even help people to get group rates in public transportation.

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 skills as profile entries, and *moPiMine* uses the profile entries to determine the best team for a specific task.

Another possible application of the *moPiMine* framework would be offering 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. With mobile grouping the visitors can be grouped according to their interests and preferences, which allow the guide to make a more personalized tour.

The rest of the paper is organized as follows. Section II gives an overview of related work of other clustering or grouping problems and the next section describes the grouping mechanisms taken into account. Section IV describes the used algorithms in our approach to *moPiMine* and presents simulation results. Finally, section V concludes the paper with a summary.

II. Problem Classification and Related Work

In this section we classify with which problems a *Data Mining in a Mobile Environment on User Profiles* application is confronted and we show which other research areas are related to this new problem.

A. Problem Classification

Data Mining in a Mobile Environment on User Profiles comprises three main problems which have to be solved to accomplish a *moPiMine* application. The first problem is the dynamic behavior of an ad hoc network, where the number of mobile hosts and communication links permanently change. Secondly, a data structure for the user profile has to be defined and a mechanism must be created to compare profile instances. Finally, similar profiles have to be found in the ad hoc network and the corresponding host form a group, in spite of the dynamic behavior of the ad hoc network.

B. Related Research Areas

Grouping algorithms and their applications appear very often in literature. There are mainly two research areas associated with it, namely mobile networks, databases and data mining. Grouping in mobile networks describes the partitioning of a mobile network in several, mostly disjoint, clusters [1], [2]. This clustering takes place at the network layer and is used for routing purposes.

Clustering is also known in the 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 [4] or Fraley and Raftery [5] an overview of many algorithms for that domain can be found. Maitra [8] and Kolatch [7] examine data-clusters in distributed databases.

moPiMine 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 *moPiMine*, e. g. while in the database area millions of data sets must be scanned, in the *moPiMine* application at the utmost one hundred 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 their own.

III. Grouping Mechanisms

If you look at our daily life there are basically two main reasons why people form groups. The first one is that people cannot solve a problem without a group. Imagine a person wants to play volleyball. A single person is unable to do this kind of sport alone and needs other people. The other reason for group formation is that people are benefiting by the group, although they are also would be capable of solving the problem alone. Examples are group rates in public transportation or cultural facilities. If people join a group they benefit by a lower entrance fee.

The first reason is named *similarity based grouping*, because the resulting group consists of people with similar profiles or at least of similar profile sections. The other approach is called *benefit based grouping* because of the mentioned benefit associated with the group membership.

In the following, for both mechanism it is explained how they are addressed algorithmically.

A. Benefit based Grouping

The heart of the benefit based grouping is the profit function π. This function associates to each group constellation a specific value.

Definition Group Profit Function π:
Let $\mathfrak{P}(V)$ be the power set of the nodes of a graph $G(V, E)$ and $\mathcal{G}$ a local group. The Group Profit function $\pi(\mathcal{G}) : \mathfrak{P}(V) \to \mathbb{R}$ assigns a value to a group $\mathcal{G} \in \mathfrak{P}(V)$. This value reflects the benefit, which emerges from group formation. π is domain dependent and differs form application scenario to application scenario.

Let G be a group and p a potential new group member. The necessary condition for adding p to the group G is

$$\pi(G \cup p) > \pi(G). \tag{1}$$

Let $\mathfrak{P}(G)$ the power set of the group G Then, the sufficient condition for adding p to the group G is

$$\forall G_i \in \mathfrak{P}(G) : \pi(G \cup p) > \pi(G_i)$$

This condition is needed, because it may happen, although condition one holds, partitioning the group in subgroups is even more efficient than adding the new point to the existing group.

Algorithmically, if a large number of new group members are available benefit based grouping is an optimization problem. But, if we assume, there are only a few new members (according to the upper velocity threshold) the grouping consists only of a test of the necessary and sufficient condition.

B. Similarity based Grouping

This approach groups people with *similar* profile entries and is a variation of the benefit based approach. The differences are, that the number of groups k must be predetermined and that there is a special profit function is associated with each group G_i.

Let the population $\mathcal{P}$ be the set of people that want to be grouped in disjoint groups and G_i the set of groups, $\mathcal{P}$ is split up, so that $\bigcup_{i=1}^{k} G_i = \mathcal{P}$. Let k be the number of groups the population $\mathcal{P}$ is split up and $d_X(x, y)$ a distance function to express the similarity of two profile entries. The global criterion for the most similar groups is

$$\min \sum_{i=1}^{k} c(G_i), \quad \text{with } c(G_i) = \sum_{r=1}^{|G_i|} \sum_{s=1}^{|G_i|} \left(d_X(x_r, y_s)\right)^2.$$

As distance functions

$$d_E(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2 \qquad d_M(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

are used. d_E is the Euclidian distance and is the more intuitive one. d_M is the Manhattan distance and is easier to compute and more robust against outliers.

With these changes of the benefit based grouping, the similarity based grouping turns into the k-means algorithm, which is well understood (see [4]).

C. Discussion

Both grouping approaches are rather complex according to complexity. If a global optimum should be reached, both methods are NP-hard. For that reason we concentrate on local optimization in order to reduce the complexity.

The first heuristic can be the analysis of the profit function. If it is known where the profit function has its global or local maximum, the search of new members can be restricted to a certain area. For this heuristic a huge amount of mathematical work is needed, because analyzing functions cannot be generalized.

The next way to reduce the complexity is to choose the group members in an opportunistic way. If more potential grouping partners are available that one is chosen, that increases the profit function most. But this procedure must not result in a global optimum. Simulation results show, that on the average the profit function is 10 to 15 % less than the optimum.

The last heuristic we apply is a kind of hierarchical grouping if it becomes necessary. The grouping process is split up in a *local grouping* and a *decentralized grouping* part. The specifics are explained in a later section.

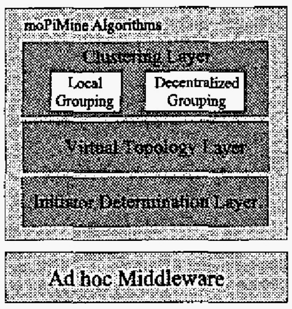

Fig. 1. Architecture of the *moPiMine* Algorithm Entity

IV. *moPiMine* 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.

A. moPiMine *Algorithm Entity*

The most important part of a *moPiMine* application is the Algorithm Entity (AE). The design of this essential entity is shown in figure 1. 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 services to find new communication partners. As middleware, we use a JXTA approach and for the future we try to migrate our scenario to a Bluetooth application.

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.

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 which 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.

B. Initiator Determination

Before the spanning tree is created, the initiators must be determined who are allowed to send the first `creation`-messages. Without initiators all hosts 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 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.* [9]). These algorithms are rather complex, i. e. a lot of messages are sent which is very time consuming. They guarantee that only one leader is elected and in case of link failures that another host takes the initiator role. Such a procedure is not appropriate and not necessary for *moPiMine*, because the initiator is only needed once and it matters little if more than one initiator is present. Therefore, we decided for the passive determination method, which is similar to Gafni and Bertsekas [6]. 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 neighbor IDs, we only allow a host being an initiator, if its ID is larger than all IDs of its neighbors. The initiator is in charge of starting the virtual topology algorithm, described in the next section.

C. Virtual Topology Creation

Having confined the number of initiators, an ad hoc 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 guarantees, that for every G a spT(G) exists.

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, 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 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 which is already in the STSL, it is removed from the list.

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 (more are also possible) trees.

In order to limit the tree size a hop-counter c_h is enclosed to each message and is each time decremented, the message is forwarded. If the counter is equal to zero, the forwarding process stops. Note, 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 tree around 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

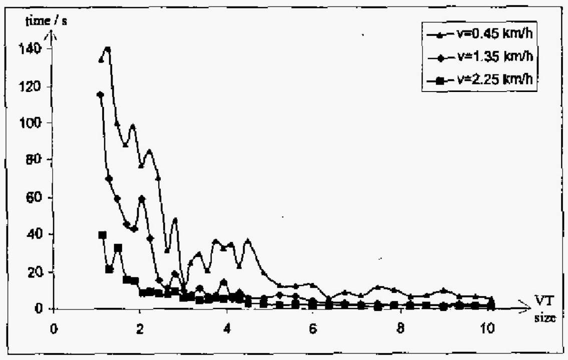

Fig. 2. Virtual topology stability

and a node does notice it does still not belong to a tree, an initiator determination is started by this host. Two cases must be distinguished. In the first one 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 an `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 starts creating a new tree.

2) Stability of the Virtual Topology: Due to the mobility of the nodes a created virtual topology is not stable. Figure 2 envisages the stability time of a virtual topology for different velocities. The picture shows the stability for a virtual tree topology.

The more members a virtual topology has, the less stable is it, because the probability that a member leaves the topology increases. As it can be seen int the picture the tree topology is more stable with a small number of topology members but decreases more rapidly than the ring topology. With a large number of members both topologies behave equally.

The essence of figure 2 is, that a virtual topology for a huge amount of members and for high velocities cannot be maintained.

D. 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 we use the group profit function, defined in section III-A.

If a new node v_i is added to the group the necessary condition of equation 1 in section III must hold.

The algorithm adds in each step exactly one new local group member. Initially, a host scans all known profiles and adds that one, with the smallest distance to him. If the group with two points will bring a greater benefit, the points is added to the

```
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;
```

Fig. 3. PseudoCode of the Local Grouping Algorithm

point. The group now has two members. It may only one point be added in one step, because else the shape of a group gets beyond control A host A can add another host B in the exactly opposite direction than a host D is added by host C. If more than one points should be added, coordination is needed.

The points already belong to the group form a fractional 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) the left side is on 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 the aforementioned definition. The pseudocode of the local grouping process is shown in figure 3

E. 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 [3].

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 section IV-C.1). If a message arrives, the enclosed local-group is taken and it is merged with its current local view of the host to get a new local view. The merging 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

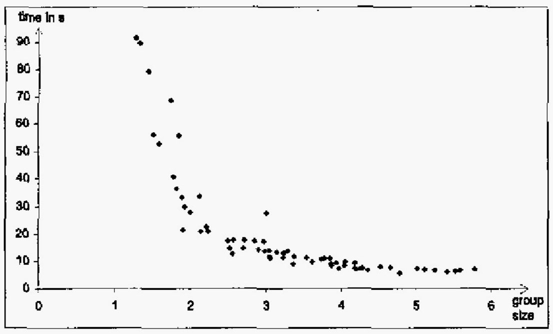

Fig. 4. Group stability time

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 hosts initiate the algorithm and a host receive several EXPLORER-messages, then only the EXPLORER-message from that host are forwarded, which has the higher ID (message extinction).

In order to determine when the algorithm has finished we need a termination criterion, because we do not know how many messages need to be sent to reach an agreement. Currently, a host stops sending messages if the local views of all direct neighbors do not differ from its own local view. This can be determined if equal local views are received from these neighbors.

F. Group Stability

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

We developed a simulation tool to test, how long the groups are stable. In the simulation environment the assumptions are:

- We do not rely on any central component.
- There is no location information available, e. g. GPS-data or cell-info.
- Each mobile device has a permanent, constant unique ID.
- The transmission range r_t of all hosts is equal.
- Each host knows all its neighbors and its associated ID.
- The communication is done in pure ad hoc mode, no base station are needed.

The velocity of the mobile hosts is uniformly distributed in the interval $[0; 5.2]$, the average velocity of the mobile hosts 2.6 $\frac{km}{h}$. 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 transmission radius of 50 meters. Figure 4 shows this dependency. It shows, that the time a group is stable, decreases extremely. A group with 2 people exists on the average for 30 seconds, whereas a group with 5 people is only stable for 9 seconds. Nevertheless, a group that is stable for 9 seconds is still sufficient for our algorithms. In order to investigate the dependencies numerous simulations were carried out.

To express the result, let r_t be the transmission radius and $\overline{v}$ be the average speed of the mobile peers, then these simulations show, that $\frac{\pi r}{2\overline{v}}$ = const. This expression leads to the following formula for the stability-time t_s of a group with n members:

$$t_s = \frac{\pi r}{2\overline{v}} \cdot \frac{1}{n-1} \quad .$$

With this formula for t_s we have an upper threshold for the time in which the grouping process must be finished.

V. Conclusion

In this paper we presented a kind of ad hoc applications called *Mobile Profile mining*, also called *moPiMine*). Each mobile host is endowed with its user's profile and while the user walks around clusters are to be found, which are composed of hosts with similar profiles.

Our approach deals with two kinds of groups. The first one is benefit based grouping, where only groups are formed when the members gain a profit by the group. The other approach is a similarity based grouping, where people with similar profiles form a group.

The algorithms of a *moPiMine* application are explained. This entity is responsible for ad hoc network partitioning as well as 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 long enough to run the algorithms.

We apply the *moPiMine* framework in a taxi sharing scenario, where people with similar destinations share a taxi and therefore save money. For the future we will apply *moPiMine* in other domains, e. g. the intelligent manufacturing area.

References

[1] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. Technical report, Univ. of Maryland at College Park, 2000.

[2] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of the IEEE International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN), Perth.*, pages 310–315, 1999.

[3] E. J. H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, SE-8(4):391–401, July 1982.

[4] D. Fasulo. An analysis of recent work on clustering algorithms. Technical report, University of Washington, 1999.

[5] C. Fraley and A. E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, 41(8):578–588, 1998.

[6] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, COM-29(1):11–18, January 1981.

[7] E. Kolatch. Clustering algorithms for spatial databases: A survey. Technical report, Department of Computer Science, University of Maryland, College Park, 2001.

[8] R. Maitra. Clustering massive datasets. In statistical computing at the 1998 joint statistical meetings., 1998.

[9] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proc. of the Fourth Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 96–103, 2000.