Mining Frequent Movement Patterns in Large Networks: A Parallel Approach Using Shapes

Mohammed Al-Zeyadi, Frans Coenen and Alexei Lisitsa

Abstract This paper presents the Shape based Movement Pattern (ShaMP) algorithm, an algorithm for extracting Movement Patterns (MPs) from network data that can later be used (say) for prediction purposes. The principal advantage offered by the ShaMP algorithm is that it lends itself to parallelisation so that very large networks can be processed. The concept of MPs is fully defined together with the realisation of the ShaMP algorithm. The algorithm is evaluated by comparing its operation with a benchmark Apriori based approach, the Apriori based Movement Pattern (AMP) algorithm, using large social networks generated from the Cattle tracking Systems (CTS) in operation in Great Britain (GB) and artificial networks.

Keywords Knowledge Discovery and Data Mining · Distributed AI Algorithms · Systems and Applications

1 Introduction

Networks of all kinds feature with respect to many application domains; social networks [1], computer networks [2], Peer-to-Peer Networks [3], road traffic networks [4] and so on. A network G is typically defined in terms of a tuple of the form $\langle V, E \rangle$, where V is a set of vertices and E is a set of edges [5]. The vertices can represent individuals (as in the case of social networks), inanimate entities (as in the case of computer networks) or locations (as in the case of distribution and road traffic networks). The edges then indicate connections between vertices (virtual or actual). These edges might be indicative of some relationship, such as a friend relationship,

M. Al-Zeyadi (✉) · F. Coenen · A. Lisitsa
Department of Computer Science, University of Liverpool, Ashton Building, Ashton Street, Liverpool L69 3BX, UK
e-mail: m.g.a.al-zeyadi@liv.ac.uk

F. Coenen
e-mail: Coenen@liv.ac.uk

A. Lisitsa
e-mail: lisitsa@liv.ac.uk

M. Bramer and M. Petridis (eds.), *Research and Development in Intelligent Systems XXXIII*, DOI 10.1007/978-3-319-47175-4_4

as in the case of social networks; or be indicative of a "hard" connection as in the case of a wired computer network or a road traffic network. However in this paper we conceive of edges as indicating traffic flow. For example: the number of messages sent from one individual to another in a social network, the volume of data exchanges between two computers in a compute network, the quantity of goods sent in a distribution network or the amount of traffic flow from one location to another in a road traffic flow network. As such our edges are directed (not necessarily the case in all net works). To distinguish such networks from other networks we will use the term *movement network*; a network $G(V, E)$ where the edges are directed and indicate some kind of "traffic flow" (for example messages, data, goods or vehicles) moving from one vertex to another.

An example movement network is given in Fig. 1 where $V = \{\phi_1, \phi_2, \ldots, \phi_5\}$ and $E = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_9\}$. Note that some of the vertices featured in the network double up as both "to" and "from" vertices. Thus the set of from vertices and the set of to vertices are not disjoint. Note also that some vertices in the figure are connected by more than one edge. This is because we do not simply wish to consider traffic flow in a binary context (whether it exists or does not exist) but in terms of the nature of the traffic. Thus where vertices in the figure are connected by more than one edge this indicates more than one kind of traffic flow. For example in a distribution movement network two edges connecting one vertex to a another might indicate the dispatch of two different commodities. As such edges have a set of attributes associated with them A_E. Similarly the vertices will also have a set of attributes associated with them, A_V. The nature of these attribute sets will depend on the nature of the application domain of interest, however each attribute will have two or more values associated with them. Where necessary we indicate a particular value j belonging to attribute i using the notation v_{i_j}.

Given a movement network of the form shown in Fig. 1 the nature of the traffic exchange between the vertices in the network can be described in terms of a *Movement Pattern* (MP); a three part pattern describing some traffic exchange comprising:

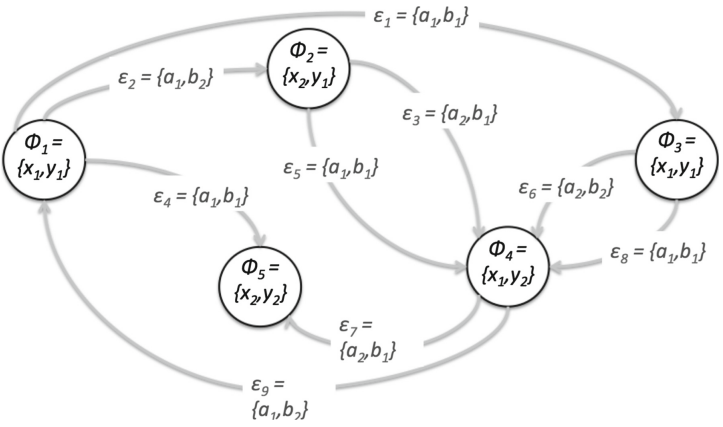


Fig. 1 Example movement network ($V = \{\phi_1, \phi_2, \ldots, \phi_5\}$ and $E = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_9\}$)

(i) a from vertex (F), (ii) the nature of the traffic (E) and (iii) a to vertex (T). Thus "sender", "details of movement" and "receiver". A movement pattern can thus be described as a tuple of the form $\langle F, E, T \rangle$, where F and T subscribe to the attribute set A_V, and E to the attribute set A_E. In the figure $A_V = \{x, y\}$ and $A_E = \{a, b\}$; each attribute x, y, a and b has some value set associated with it, in the figure the value sets are $\{x_1, x_2\}$, $\{y_1, y_2\}$, $\{a_1, a_2\}$ and $\{b_1, b_2\}$ respectively (value sets do not all have to be of the same length). Given a movement network G we wish to find the set of movement patterns M in G so that they can be used to predict movement in a previously unseen unconnected graph $G'(V')$ so as to generate a predicted connected graph $G'(V', E')$. More specifically we wish to be able to extract the set M from very large networks comprising more than one million vertices. To this end the Shape Movement Pattern (ShaMP) algorithm is proposed and evaluated in this paper. The algorithm leverages knowledge of the restrictions imposed by the nature of MPs to obtain efficiency advantages not available to more traditional pattern mining algorithms that might otherwise be adopted such as Frequent Itemset Mining (FIM) algorithms. A further advantage of the ShaMP algorithm, as will be demonstrated, is that it readily lends itself to parallelisation [6]. The algorithm is fully described and its operation compared to a benchmark Apriori style approach using both artificial movement networks and real life networks extracted from the Cattle Tracking System (CTS) database in operation in Great Britain.

The rest of this paper is organised as follows. Section 2 presents a review of previous and related work on Movement Patterns in large networks and how parallel processing can improve the efficiency of such algorithms. Section 3 provides a formal definition of the movement pattern concept. Section 4 then describes the proposed ShaMP algorithm and the Apriori Movement Pattern (AMP) benchmark algorithm used for comparison proposes in the following evaluation section, Sect. 5. Section 6 summarises the work, itemises the main findings and indicates some future research directions.

2 Literature Review

In the era of big data the prevalence of networks of all kinds has growing dramatically. Coinciding with this growth is a corresponding desire to analyse (mine) such networks, typically with a view to some social and/or economic gain. A common application is the extraction of customer buying patterns [7, 8] to support business decision making. Network mining is more focused but akin to graph mining [9, 10]. Network mining can take many forms; but the idea presented in this paper is the extraction of Movement Patterns (MPs) from movement networks. The concept of Movement Pattern Mining (MPM) as conceived of in this paper, to the best knowledge of the authors, has not been previously addressed in the literature. However, patten mining in general has been extensively studied. This section thus presents some background to the work presented in the remainder of this paper. The section commences with a brief review of the pattern mining context with respect to large

graph networks, then continues with consideration of current work on frequent movement patterns and the usage of parallel algorithms to find such patterns in large data networks.

A central research theme within the domain of data mining has been the discovery of patterns in data. The earliest examples are the Frequent Pattern Mining (FPM) algorithms proposed in the early 1990s [11]. The main objective being to discover sets of attribute-value pairings that occur frequently. The most well established approach to FPM is the Apriori approach presented in [11]. A frequently quoted disadvantage of FPM is the significant computation time required to generate large numbers of patterns (many of which may not be relevant). The MPM concept presented in this paper shares some similarities with the concept of frequent pattern mining and hence a good benchmark algorithm would be an Apriori based algorithm such as the AMP algorithm presented later in this paper. The distinction between movement patterns and traditional frequent patterns is that movement patterns are more prescriptive, as will become apparent from the following section; they have three parts "sender, details of movement and receiver"; none of these parts can be empty. Note also that the movement patterns of interest with respect to this paper are network movement patterns and not the patterns associated with the video surveillance of individuals, animals or road traffic; a domain where the term "movement pattern" is also used. To the best knowledge of the authors there has been very little (no?) work on movement patterns as conceptualised in this paper.

The MPM concept also has some overlap with link prediction found in social network analysis where we wish to predict whether two vertices, representing individuals in a social network, will be linked at some time in the future. MPM also has some similarity the problem of inferring missing links in incomplete networks. The distinction is that link prediction and missing link resolution is typically conducted according to graph structure, dynamic in the case of link prediction [12] and static in the case of missing link resolution [13].

As noted above, the main challenge of finding patterns in network data is the size of the networks to be considered (in terms of vertices and edges). Parallel or distributed computing is seen as a key technology for addressing this challenge. One of the main advantages offered by the proposed ShaMP algorithm is that it readily lends itself to parallelisation (unlike traditional FPM algorithms which require significant interchange of information). The design of parallel algorithms involves a number of challenges in addition to those associated with the design of serial algorithms [6]. In the case of MPM the main challenge is the distribution of the data and tasks across the available processes.

In the context of the proposed ShaMP algorithm a Distributed Memory Systems (DMS) was used; a memory model in which each process has a dedicated memory space. Using a distributed memory approach only the input data and task information is shared. There are several programming models that may be used in the context of DMS. However, by far the most widely used for large-scale parallel applications is the *Message Passing Interface* (MPI) [14]; a library interface specification that supports distributed-memory parallelisation, where data is divided among a set of "processes" and each process runs in its own memory address space. Processes

typically identify each other by "ranks" in the range $\{0, 1, \ldots, p-1\}$, where p is the number of processes [15]. MPI is a very powerful and versatile standard used for parallel programming of all kinds. APIs that adopt the MPI standard, at their simplest, typically provide send and receive functionality, however most such APIs also provide a wide variety of additional functions. For example, there may be functions for various "collective" communications, such as "broadcast" or "reduction" [15]. MPI based APIs exist in a range of programming languages such as C, C++, Fortran and Java; well-tested and efficient implementations include: MPICH [16], MPICH-G2 [17] and G-JavaMPI [18]. However, in the context of the research presented in this paper MPJ Express [19] was adopted; this is an implementation of MPI using the Java programming language. Interestingly, MPJ has two ways of configuring the same code; Multicore configuration and Cluster configuration. In this paper we have adopted both Multicore configuration (by using a single multicore machine) and Cluster configuration using a Linux cluster.

3 Formalism

From the introduction to this paper we are interested in mining Movement Patterns (MPs) from movement networks. A MP comprises a tuple of the form $\langle F, E, T \rangle$ where F, E and T are sets of attribute values. The minimum number of attribute values in each part must be at least one. The maximum number of values depends on the size of the attribute sets to which F, E and T subscribe, an MP can only feature a maximum of one value per subscribed attribute. More specifically the attribute value set F represents a "From" vertex (sender), T a "To" vertex (receiver), and E an "Edge" connecting the two vertices describing the nature of the traffic ("details of movement"). The attribute set to which F and T subscribe is given by $A_V = \{\phi_1, \phi_2, \ldots\}$, whilst the attribute set for E is given by $A_E = \{\varepsilon_1, \varepsilon_2, \ldots\}$. Note that F and T subscribe to the same attribute set because they are both movement network vertices, and every vertex (at least potentially) can be a "from" or a "to" vertex in the context of MPM (as illustrated in Fig. 1).

The movement networks from which we wish to extract MPs (the training data) can also be conceived of as comprising $\langle F, E, T \rangle$ tuples. In fact an entire network G can be represented as a tabular data set D where each row comprises a $\langle F, E, T \rangle$ tuple defined as described above. The movement network presented in Fig. 1 can thus be presented in tabular form as shown in Table 1 (for ease of understanding the rows are ordered according to the edge identifiers used in the figure). We refer to such data as FET data (because of the nature of the tuples that the rows describe). Thus MPM can be simplistically thought of as searching a training set D and extracting the set M of all frequently occurring MPs, that can then be used to build a model of D. This model can then be used to predict traffic in some previously unseen network D' (G') comprised solely of vertices (no known edges, the edges are what we wish

Table 1 Movement network from Fig. 1 presented in tabular form

$\langle\{x_1,y_1\},\{a_1,b_1\},\{x_1,y_1\}\rangle$
$\langle\{x_1,y_1\},\{a_1,b_2\},\{x_2,y_1\}\rangle$
$\langle\{x_2,y_1\},\{a_2,b_1\},\{x_1,y_2\}\rangle$
$\langle\{x_1,y_1\},\{a_1,b_1\},\{x_2,y_2\}\rangle$
$\langle\{x_2,y_1\},\{a_1,b_1\},\{x_1,y_2\}\rangle$
$\langle\{x_1,y_1\},\{a_2,b_2\},\{x_1,y_2\}\rangle$
$\langle\{x_1,y_1\},\{a_1,b_1\},\{x_1,y_1\}\rangle$
$\langle\{x_1,y_2\},\{a_2,b_1\},\{x_2,y_2\}\rangle$
$\langle\{x_1,y_2\},\{a_1,b_2\},\{x_1,y_1\}\rangle$

Table 2 The MPs (the set M) extracted from the movement network given in Fig. 1 using $\sigma = 30\,\%$

$\langle\{x_1\},\{a_1\},\{x_1\}\rangle$	#3
$\langle\{y_1\},\{a_1,b_1\},\{x_1\}\rangle$	#3
$\langle\{y_1\},\{a_1,b_1\},\{y_2\}\rangle$	#3
$\langle\{y_1\},\{a_1\},\{x_1\}\rangle$	#3
$\langle\{y_1\},\{a_1\},\{y_2\}\rangle$	#3
$\langle\{y_1\},\{b_1\},\{x_1\}\rangle$	#4
$\langle\{y_1\},\{b_1\},\{x_1,y_2\}\rangle$	#3
$\langle\{y_1\},\{b_1\},\{y_2\}\rangle$	#4

to predict). An MP is said to be frequent, as in the case of traditional FPM [20], if its occurrence count in D is in excess of some threshold σ expressed as a proportion of the total number of FETs in D. With reference to the movement network given in Fig. 1, and assuming $\sigma = 30\,\%$, the set of MPS, M, will be as listed in Table 2 (the numbers indicated by the # are the associated occurrence counts).

4 Movement Pattern Mining

In this section the proposed ShaMP algorithm is presented together with the Apriori Movement Pattern (AMP) algorithm used for comparison purposes later in this paper (Sect. 5). The AMP algorithm is a variation of the traditional Apriori frequent itemset mining algorithm [7] redesigned so as to address the frequent MP mining problem. We refer to this benchmark algorithm as the Apriori Movement Pattern (AMP) algorithm. The major limitation of the Apriori strategy used by the AMP algorithm is the large number of candidate patterns that are generated during the process, which limits the size of the networks that can be considered. Also, when using the AMP algorithm, with respect to a given network G (dataset D), the run time is inversely proportional with the support threshold σ used. The ShaMP algorithm, however, uses knowledge of the nature of the FET "shapes" to be considered (the fact that they have three parts and that each part must feature at least one attribute-value) of which these are only a limited number. Broadly, the ShaMP algorithm operates by generating a set of MP

shapes to which records are fitted (there is no candidate generation). The algorithm offers two advantages: (i) the nature of the value of σ has very little (no?) effect on the algorithm's run time and (ii) individual shapes can be considered in isolation hence the algorithm is well suited to parallelisation (not the case with respect to the AMP algorithm which operates level by level in a top down manner and, when parallelised, features significant message passing after each level). Consequently, as will be demonstrated in Sect. 5, the ShaMP algorithm is able to process much larger networks than in the case of the AMP algorithm. The ShaMP algorithm is considered in Sect. 4.1 below, while the AMP algorithm is considered in Sect. 4.2.

4.1 The Shape Based Movement Pattern (ShaMP) Algorithm

From the foregoing the ShaMP algorithm, as the name suggests, is founded on the concept of "shapes". A shape in this context is a MP template with a particular configuration of attributes taken from A_L and A_E (note, shapes do not specify particular attribute values combinations). The total number of shapes that can exist in a FET dataset D can be calculated using Eq. 1, where: (i) $|A_L|$ is the size of the attribute set A_L and (ii) $|A_E|$ is the size of the attribute set A_E. Recall that attributes for F and T are drawn from the same domain. Thus if $|A_L| = 2$ and $|A_E| = 2$, as in the case of the movement network given in Fig. 1, there will be $(2^2 - 1) \times (2^2 - 1) \times (2^2 - 1) = 2 \times 2 \times 2 = 16$ different shapes. If we increase $|A_E|$ to 5 there will be $(2^2 - 1) \times (2^5 - 1) \times (2^2 - 1) = 3 \times 31 \times 3 = 279$ different shapes. In other words the efficiency of the ShaMP algorithm is directly related to the number of different shapes that need to be considered.

$$(2^{|A_V|} - 1) \times (2^{|A_E|} - 1) \times (2^{|A_V|} - 1) \tag{1}$$

The pseudo code for the ShaMP Algorithm is given in Algorithm 1. The input is a network G, represented in terms of a FET dataset D, and a desired support threshold σ. The output is set of frequently occurring MPs M together with their support value represented in tuples of the form $\langle MP_i, count_i \rangle$. The algorithm commences (line 5) by generating the available set of shapes, *ShapeSet*, and setting M to the empty set (line 6). We then loop through the *ShapeSet* (lines 7–15), and for each shape loop through D comparing each record $r_j \in D$ with the current shape $shape_i$. A record r_j matches a $shape_i$ if the attributes featured in the shape also feature in r_j. Where a match is found the relevant attribute values in r_i form a MP. If the identified MP is already contained in M we simply update the associated support value, otherwise we add the MP to M with a support value of 1. Once all shapes have been processed

we loop through M (lines 15–16) and remove all MPs whose support count is less than σ.

```
   Input:
 1   D = Collection of FETs, {r1, r2, ...} describing a network G
 2   σ = Support threshold
   Output:
 3   M = Set of frequently occurring MPs {⟨MP1, count1⟩, langleMP2, count2⟩, ...}
 4 Start:
 5 ShapeSet = the set of possible shapes {shape1, shape2, ...}
 6 M = ∅
 7 forall shapei ∈ ShapeSet do
 8     forall ri ∈ D do
 9         if ri matches shapei then
10         MPk = MP extracted from ri
11         if MPk in M then increment support
12         else M = M ⋃ ⟨MPk, 1⟩
13     end
14 end
15 forall MPi ∈ M do
16     if count for MPi < σ then remove MPi from M
17 end
```

Algorithm 1: Shape-Based Movement Pattern Algorithm

4.2 The Apriori Based Movement Pattern (AMP) Algorithm

The AMP algorithm is presented in this section. The significance is that it is used for evaluation purposes later in this paper. As noted above, the AMP algorithm is founded on the the Apriori frequent itemset mining algorithm presented in [11] (see also [10]). The pseudo code for the AMP algorithm is presented in Algorithm 2. At face value it operates in a very similar manner to Apriori in that it adopts a candidate generation, occurrence count and prune cycle. However, the distinction is that we are dealing with three part MPs ($\langle F, E, T\rangle$) and that none of these parts should be empty. Thus, where the variable k in the traditional Apriori algorithm refers to itemset size (starting with $k = 1$), the variable k in the AMP algorithm refers to levels. We start with the $k =$ level 1 MPs which feature one attribute value in each part ($\langle F, E, T\rangle$). At level two, $k =$ level 2, we add an additional attribute-value to one of the parts; and so on until no more candidates can be generated. Candidate generation and occurrence counting is therefore not straight-forward, it is not simply a matter of uniformly "growing" K-itemsets to give $K + 1$-itemsets and passing through D looking for records where each K-itemset is a subset. Candidate set growth is thus a complex procedure (not included in Algorithm 2) and requires knowledge of the values for

A_L and A_E. It should also be noted that when conducting occurrence counting only certain columns in D can correspond to each element in a candidate FET.

```
   Input:
 1   D = Binary valued input data set ; σ = Support threshold
   Output:
 2   M = Empty set of frequently occurring MPs {⟨MP1, count1⟩, langleMP2, count2⟩, ...}
 3 Start:
 4 M = ∅, k = level 1
 5 Ck = Level k candidate movement patterns
 6 while Ck ≠ ∅ do
 7     S = {0, 0, ...} Set of length |Ck| to hold occurrence counts (one-to-one Ck
         correspondence)
 8     G = ∅ Empty set to hold frequently occurring level K movement patterns
 9     forall ri ∈ D do
10         forall cj ∈ Ck do
11             if cj ⊂ ri then sj = sj + 1 (sj ∈ S)
12         end
13     end
14     forall cj ∈ Ck do
15         if sj ≥ σ then
16         G = G ∪ cj
17         M = M ∪ cj
18     end
19     k = k + 1
20     Ck = Level k candidate movement patterns drawn from G
21 end
```

Algorithm 2: Apriori based Movement Pattern (AMP) Algorithm

Returning to Algorithm 2, as in the case of the ShaMP algorithm, the input is a FET dataset D, and a desired support threshold σ. The output is a set of frequently occurring MPs M together with their support values. The algorithm commences (line 3) by setting M to the empty set. The algorithm then proceeds level by level starting with the level one ($k = 1$) candidate sets (line 5), and continues until no more candidates can be generated. On each iteration a set S and a set G is defined to hold occurrence counts and the eventual size K MPs (if any). We then loop through the records in D (line 9), and for each record $r_i \in D$ loop through C_k (line 10). For each each $c_j \in C_k$ if c_j is a subset of r_i we increment the relevant support count in S (line 12). Once all the records in D have been process we loop through C_k again, compare each associated s_j value with σ and if it is larger add it to the sets G and M. We then increment k (line 20), generate a new candidate set C_k from G and repeat. The main weaknesses of the algorithm, as noted above, are the large number of candidates that may be generated (the maximum number equates to $2^{|V_{A_L}|+|V_{E_L}|+|V_{A_L}|} - 1$). Note that, in common with more general Apriori algorithms, the number of relevant candidates to be considered becomes larger when low values for σ are used (as opposed to the ShaMP algorithm where the σ value dose not have a significant effect).

5 Experiments and Evaluation

This section reports on the experiments conducted to analyse the operation of the proposed ShaMP algorithm and compare this operation with the benchmark AMP algorithm. The evaluation was conducted using two categories of network data: (i) networks extracted from the CTS database introduced in Sect. 1 and (ii) artificial networks where the parameters can be easily controlled. The objectives of the evaluation were as follows:

1. To determine whether the nature of the σ threshold used would in anyway adversely affect the ShaMP algorithm (unlike as anticipated in the case of the AMP algorithm).
2. To determine the effect on the ShaMP algorithm of the size of the network under consideration, in terms of the number of FETs, and in comparison with the AMP algorithm.
3. To determine the effect on the ShaMP algorithm on the number shapes to be considered, and if there is a point where the number of shapes is so large that it is more efficient to use an algorithm such as the AMP algorithm.
4. The effect of increasing the number of processes available with respect to the ShaMP algorithm (the first three sets of experiments were conducted using a single processor).

The remainder of this section is divided into five sections as fallows. Section 5.1 gives an overview of the CTS and artificial networks used with respect to the reported evaluation. The remaining four section, Sects. 5.2–5.5, report respectively on the experimental results obtained with respect to the above four objectives.

5.1 Data Sets

The CTS database was introduced in the introduction to this paper. The database was used to generate a collection of time stamped networks where for each network the vertices represent locations (cattle holding areas) and the edges represent occurrences of cattle movement between vertexes. The database was preprocessed so that each record represented a group of animals moved of the same type, bread and gender, from a given "from location" to a given "to location" on the same day. The set A_L comprised: (i) holding area type and (ii) county name. While the set A_E comprised: (i) number of cattle moved, (ii) breed, (iii) gender, (iv) whether the animals moved are beef animals or not, and (v) whether the animals moved are dairy animals or not. Thus $A_V = \{holdingAreaType,\ county\}$, and $A_E = \{numCattle,\ bread,\ gender,\ beef,\ dairy\}$. Note that the attributes *beef* and *dairy* are boolean attributes, the attribute *numCattle* is a numeric attribute, while the remaining attributes are categorical (discrete) attributes. Both the ShaMP and AMP algorithms were designed to operate with binary valued data (as in the case of traditional frequent item set mining). The

values for the *numCattle* attribute were thus ranged into five sub ranges: $n \leq 10$, $11 \leq n \leq 20$, $21 \leq n \leq 30$, $31 \leq n \leq 40$ and $n > 40$. Thus each record represents a FET. The end result of the normalisation/discretisation exercise was an attribute value set comprising 391 individual attribute values.

We then used this FET dataset to define movement networks where vertices represent locations and edges traffic (animals moved), to which the proposed algorithms were applied to extract MPs. In total four networks were generated covering the years 2003, 2004, 2005 and 2006. The number of vertices in each network was about 43, 000, while the number of edges was about 270,000. Using Eq. 1 the number of shapes that will need to be considered by the ShaMP algorithm will be:

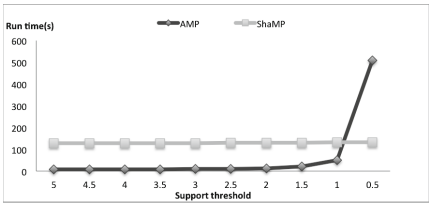
$$|ShapeSet| = 2^2 - 1 \times 2^5 - 1 \times 2^2 - 1 = 3 \times 15 \times 3 = 279$$

The maximum number of candidate MPs that the AMP algorithm may need to consider on iteration one will be $102 \times 188 \times 102 = 1{,}955{,}952$, and this is likely to increase exponentially on the following iterations (thus a significant difference).

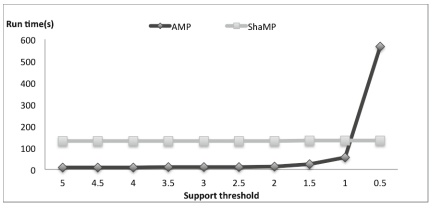
The purpose of also using artificially generated networks was that the parameters could be controlled, namely the number of FETs and the number of attributes in the sets A_L and A_E. Further detail concerning individual, or groups, of artificial movement networks will be given where relevant in the following sections where the evaluation results obtained with respect to individual experiments are reported and discussed.

5.2 Support Threshold

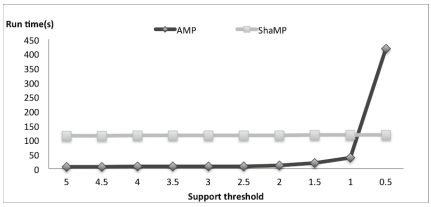
In terms of Frequent Itemset Mining (FIM) the lower the σ value the more frequent itemsets that will be found, low σ values are thus seen as desirable because by finding many frequent itemsets there is less chance of missing anything significant. The same is true for MPM. In terms of FIM it is well established that efficiency decreases as the number of potential frequent itemsets increases (as the value of σ decreases). It was anticipated that this would also be true with respect to the AMP algorithm. How this would effect the ShaMP algorithm was unclear, although it was conjectured that changing σ values would have little effect. A sequence of experiments was thus conducted using a range of σ values from 5.0 to 0.5 decreasing in steps of 0.5, and using the four CTS movement networks described above. The results are presented in Fig. 2 where, for each graph, the x-axis gives the σ values and the y-axis runtime in seconds. Similar results are recorded in all cases. As expected, in the case of the AMP algorithm, run time increases exponentially as σ decreases. Also, as conjectured, from the figures it can be seen that *sigma* has little effect on the ShaMP algorithm. It is interesting to note that it is not till σ drops below 1.0 that usage of the the ShaMP algorithm becomes more advantageous than usage of the AMP algorithm. This last is significant because we wish to identify as many relevant MPs as possible and to do this we would need to use low σ values ($\sigma \leq 1.0$).



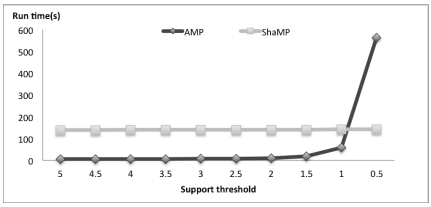
(a) $Year = 2003$ ($|D| = 2712603$)



(b) $Year = 2004$ ($|D| = 2757081$)



(c) $Year = 2005$ ($|D| = 2449251$)

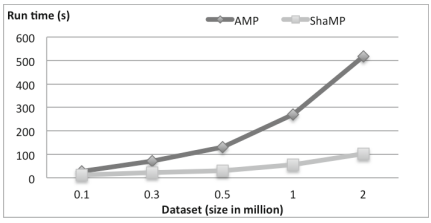


(d) $Year = 2006$ ($|D| = 2982695$)

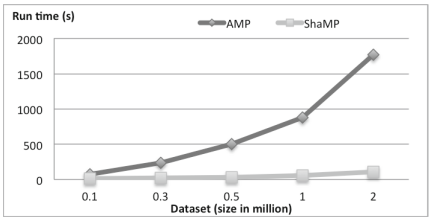
Fig. 2 Runtime (secs.) comparison between ShaMP and AMP using CTS network datasets from 2003–2006 with σ values

5.3 Number of FETs

The correlation between the algorithms and number of FETs was tested using five artificial datasets of increasing numbers of FETs 100,000, 200,000, 300,000, 500,000 and 1,000,000. The algorithms were applied to each of these data sets twice, once using $\sigma = 1.0$ and once using $\sigma = 0.5$. Note that for the artificial data the size of the attribute value sets used were $|V_{A_L}| = 102$ and $|V_{A_E}| = 188$. The results are presented in Fig. 3. From the figures it can clearly be seen that, as was to be expected, the run time increases as the number of records considered increases. However, the ShaMP algorithm is clearly more efficient than the AMP algorithm. It can also be seen that the run time with respect to the AMP algorithm increases dramatically as the number of FETs increases, while the increase with respect to the ShaMP algorithm is less



(a) $Supportthreshold = 1.0$



(b) $Supportthreshold = 0.5$

Fig. 3 Runtime (secs.) comparison between ShaMP and AMP using artificial networks

dramatic. If we increase the number of FETs beyond 1,000,000 the AMP algorithm is not able to cope. However the ShaMP algorithm, can process 10,000,000 FETs using a single processor with $\sigma = 0.5$, in 480 s of runtime.

5.4 Number of Shapes

The experiments discussed in the foregoing two sections indicated the advantages offered by the ShaMP algorithm. However as $|A_L|$ and/or $|A_E|$ increases the number of shapes will also increase. There will be a point where the number of shapes to be considered is such that it is no longer effective to use the ShaMP algorithm and it would be more beneficial to revert to the AMP algorithm. To determine where this point might be a sequence of artificial data sets were generated using $|A_V| = 2$ but increasing $|A_E|$ from 500 to 5000 in steps of 500. Each attribute had two possible attribute-values so as to allow the AMP algorithm to perform to its best advantage. (For the experiments $\sigma = 2$ was used). The results are shown in Fig. 4. As expected, the run time for both algorithms increased as the number of Shapes (attribute values) increased. It is interesting to note that there is "cross-over" when the number of shapes reaches about 2600; in practice it is difficult to image movement networks where the traffic has to be described in terms of 2600 attributes or more (Fig. 5).

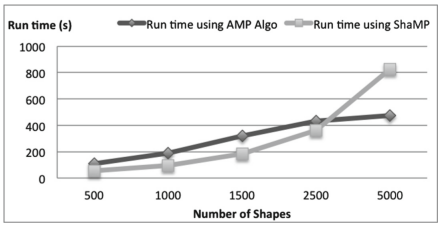


Fig. 4 Runtime plotted against increasing numbers of shapes ($\sigma = 2$)

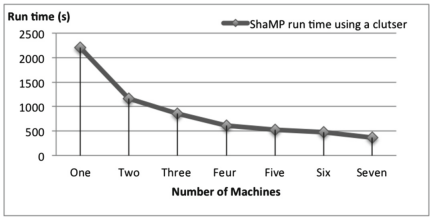


Fig. 5 Runtime plotted against increasing numbers of machines in a cluster

5.5 *Distributed ShaMP*

In the context of the experiments conducted to determine the effect of distributing the ShaMP algorithm the desired parallelisation was achieved using MPJ Express [19]. A cluster of Linux machines was used, of increasing size from 1 to 7, each with 8 cores (thus a maximum of $7 \times 8 = 56$ cores). Shapes were distributed evenly across machines. For the experiments an artificial data set comprising 4600 different Shapes were used and $\sigma = 2$. Note that the AMP algorithm was unable to process FET datasets of this size. The results are shown in Fig. 4. As predicted, as the number of machines increases the run time decreased significantly, there is no significant "distribution overhead" as a result of increasing the parallelisation. A good result.

6 Conclusion and Future Work

In this paper, the authors have proposed the ShaMP algorithm for identifying Movement Pattens (MPs) in network data. The MPs were defined in terms of three parts: From (F), Edge (E) and To (T). The acronym FET was coined to describe such patterns. The MP concept has similarities with traditional frequent itemsets except that the attributes that can appear in a particular part is limited, consequently the search space can be considerably reduced. A particular challenge was the size of the networks that we wish to analyse. Thus, although in the first instance a traditional Apriori approach seems appropriate, the size of the networks to be considered means that this approach is unlikely to succeed for any realistic application (as illustrated by the presented evaluation). Instead a parallel approach was proposed facilitated by the nature of the ShaMP algorithm which readily lends itself to parallelisation. For comparison purposes an Apriori algorithm was also defined, the AMP algorithm. The evaluation was conducted using movement networks extracted from the GB Cattle Tracking System (CTS) and artificial networks. The main findings may be summarised as fallows. The ShaMP Algorithm can successfully identify MPs in large networks with reasonable computational efficiency; where more traditional approaches, such as AMP, fail because of the resource required. Parallelisation of the ShaMP algorithm, using MPJ, has a significant impact over the computational efficiency of the algorithm.

References

1. Matsumura, N., Goldberg, D.E., Llorà, X.: Mining directed social network from message board. In: Special Interest Tracks and Posters of the 14th International Conference on World Wide Web, pp. 1092–1093. ACM (2005)
2. Chandrasekaran, B.: Survey of Network Traffic Models, vol. 567. Washington University, St. Louis CSE (2009)

3. Datta, S., Bhaduri, K., Giannella, C., Wolff, R., Kargupta, H.: Distributed data mining in peer-to-peer networks. IEEE Internet Comput. **10**(4), 18–26 (2006)
4. Gonzalez, H., Han, J., Li, X., Myslinska, M., Sondag, J.P.: Adaptive fastest path computation on a road network: a traffic mining approach. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB Endowment, pp. 794–805 (2007)
5. Galloway, J., Simoff, S.J.: Network data mining: methods and techniques for discovering deep linkage between attributes. In: Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling, vol. 53, pp. 21–32. Australian Computer Society, Inc. (2006)
6. Grama, A.: Introduction to Parallel Computing. Pearson Education (2003)
7. Raorane, A., Kulkarni, R.: Data mining techniques: a source for consumer behavior analysis (2011). arXiv:1109.1202
8. Gudmundsson, J., Laube, P., Wolle, T.: Movement patterns in spatio-temporal data. In: Encyclopedia of GIS, pp. 726–732. Springer (2008)
9. Campbell, W.M., Dagli, C.K., Weinstein, C.J.: Social network analysis with content and graphs. Lincoln Lab. J. **20**(1) (2013)
10. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques. Elsevier (2011)
11. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proceedings of 20th International Conference on Very Large Data Bases, VLDB, vol. 1215, pp. 487–499 (1994)
12. Bliss, C.A., Frank, M.R., Danforth, C.M., Dodds, P.S.: An evolutionary algorithm approach to link prediction in dynamic social networks. J. Comput. Sci. **5**(5), 750–764 (2014)
13. Kim, M., Leskovec, J.: The network completion problem: Inferring missing nodes and edges in networks. In: SDM, vol. 11, pp. 47–58. SIAM (2011)
14. Forum, M.P.I.: Mpi: a message passing interface standard: version 2.2; message passing interface forum, September 4, 2009
15. Brawer, S.: Introduction to Parallel Programming. Academic Press (2014)
16. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the mpi message passing interface standard. Parallel Comput. **22**(6), 789–828 (1996)
17. Karonis, N.T., Toonen, B., Foster, I.: Mpich-g2: a grid-enabled implementation of the message passing interface. J. Parallel Distrib. Comput. **63**(5), 551–563 (2003)
18. Chen, L., Wang, C., Lau, F.C.: A grid middleware for distributed java computing with mpi binding and process migration supports. J. Comput. Sci. Technol. **18**(4), 505–514 (2003)
19. Baker, M., Carpenter, B., Shaft, A.: Mpj express: towards thread safe java hpc. In: 2006 IEEE International Conference on Cluster Computing, pp. 1–10. IEEE (2006)
20. Aggarwal, C.C.: Applications of frequent pattern mining. In: Frequent Pattern Mining, pp. 443–467. Springer (2014)