Ant colony optimization for mining gradual patterns

Dickson Odhiambo Owuor1 · Thomas Runkler2 · Anne Laurent3 · Joseph Onderi Orero1 · Edmond Odhiambo Menya1

Received: 27 July 2020 / Accepted: 19 July 2021 / Published online: 24 July 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract
Gradual pattern extraction is a field in Knowledge Discovery in Databases that maps correlations between attributes of a data set as gradual dependencies. A gradual dependency may take the form: "the more Attribute\textsubscript{K}, the less Attribute\textsubscript{L}". Classical approaches for extracting gradual patterns extend either a breadth-first search or a depth-first search strategy. However, these strategies can be computationally expensive and inefficient especially when dealing with large data sets. In this study, we investigate 3 population-based optimization techniques (i.e. ant colony optimization, genetic algorithm and particle swarm optimization) that may be employed to improve the efficiency of mining gradual patterns. We show that ant colony optimization technique is better suited for gradual pattern mining task than the other 2 techniques. Through computational experiments on real-world data sets, we compared the computational performance of the proposed algorithms that implement the 3 population-based optimization techniques to classical algorithms for the task of gradual pattern mining and we show that the proposed algorithms outperform their classical counterparts.

Keywords Ant colony optimization · Data mining · Genetic algorithm · Gradual patterns · Particle swarm optimization · Swarm intelligence

1 Introduction
Knowledge Discovery in Databases (KDD) is a Computer Science field that attempts to unearth hidden information from huge data sets, which may otherwise take human analysts so long to discover. Gradual pattern (GP) discovery is a descriptive extension in the KDD field that expresses attribute correlations linguistically in the form of gradual rules [1–3, 8, 20].

Mining gradual dependencies may lead to uncovering new knowledge about numeric data sets in various kinds of domains. For instance, in a medical domain useful knowledge may be extracted through gradual rules like: "the faster the population growth of bacteria X, the slower the population growth of bacteria Y in organ G"—may be denoted as \{((\text{bacteriaX, ↑}), (\text{bacteriaY, ↓}))\}.

In order to formulate such causality, a candidate rule is generated from respective attribute combinations and its quality validated through a frequency support measure (which is derived from the sequence of all ordered tuples respecting the rule). For example the data set in Table 1 has 3 attributes \{Game, Win, Injury\} from which we can formulate gradual rules like \{((\text{Game, ↑}), (\text{Win, ↓}))\} (which is respected by at least 3/5 tuples), \{((\text{Game, ↓}), (\text{Win, ↓}), (\text{Injury, ↑}))\} (which is respected by at least 2/5 tuples) etc.

Many GP mining approaches extend either a breadth-first search (BFS) or depth-first search (DFS) strategy for mining gradual item sets. In this study, we propose an ant colony optimization (ACO) which uses a probabilistic approach to improve efficiency of both BFS-based and
DFS-based strategies for gradual item set mining. ACO, as originally described by [10], is a general-purpose heuristic approach for solving discrete optimization problems [4, 6, 11, 29, 32].

ACO imitates the positive feedback reinforcement behavior of biological ants as they search for food—where the more ants following a path, the more chemical pheromones are deposited on that path and the more appealing that path becomes for being followed by other ants [9, 10].

**Example 1.1** We consider a sample graph of artificial ants moving along on the edges of nodes A, B, C, D, E and F as shown in Fig. 1.

In order to make an accurate interpretation of ACO, assume that the paths between nodes B and D, D and E have longer lengths than paths between nodes B and C, C and E (as indicated by the weighted distances in Fig. 1a). Let us consider what happens at regular discrete time intervals: \( t = 0, 1, 2, \ldots \). Suppose that 12 new ants come to node B from A and 12 new ants come to node E from F at each time interval. Each ant travels at a speed of 2 weighted distance per time interval and that by moving along at time \( t \) it deposits pheromones of intensity 1, which completely and instantaneously evaporates in the middle of the successive time interval \( (t + 1, t + 2) \).

At time \( t = 0 \) no path has any pheromone intensity and there are have 12 new ants at node B and 12 new ants at node E. The ants select between nodes C and D randomly; therefore, on average 6 ants from each node move towards C and 6 ants towards D (Fig. 1b).

At time \( t = 1 \) there are 12 new ants at node B and 12 new ants at node E. The 12 new ants at node B will find a pheromone intensity of 6 on the path that leads to node D deposited by the 6 ants that used it previously from node B and a pheromone intensity of 12 on the path that leads to node C deposited by 6 ants that used it coming from node B and 6 that arrived from node E. Therefore, the ants will find the path to node C more desirable than that to node D. The probability of the ants choosing to move towards node C is \( 2/3 \), while that of moving towards node D is \( 1/3 \) and 8 ants will move towards node C and 4 ants towards node D. The same is true for the 12 new ants at node E (Fig. 1c).

### Table 1

| Id | Game | Win | Injury |
|----|------|-----|--------|
| r0 | 30   | 3   | 1      |
| r1 | 35   | 2   | 2      |
| r2 | 40   | 4   | 2      |
| r3 | 50   | 1   | 1      |
| r4 | 52   | 7   | 1      |

Our experiment results show that algorithms built on top of population-based optimization techniques perform computationally better than their classical counterparts.

**1.1 Main contributions**

The main contributions of the study are listed below:

- An ACO approach to BFS problem in GP mining.
- An ACO approach to DFS problem in GP mining.
- Alternative population-based optimization techniques for GP mining.

The remainder of the paper is organized as follows: we provide preliminary concepts in Sect. 2; we review related literature in Sect. 3; we propose ant colony optimization for GP extraction in Sect. 4; we show the results of our experiments in Sect. 5. Finally, we conclude and give future research directions in Sect. 6.
2 Preliminary concepts and notations

2.1 Gradual patterns

In this section, we provide some preliminary definitions of gradual item sets (as given by [8, 20, 25]). In order to describe gradual patterns, we assume a data set $D_n$ is defined by attributes $\{A_1, A_2, \ldots, A_n\}$ and consists of tuples $\{r_0, r_2, \ldots, r_n\}$ as shown in Table 1.

![Table 1](image)

Definition 2.1 (Gradual Item). A gradual item is a pair $(i, v)$ where $i$ is an attribute and $v$ is a variation $v \in \{1, 1\}$. $\uparrow$ stands for an increasing variation while $\downarrow$ stands for a decreasing variation.

For example, $(\text{wins}, \uparrow)$ can be interpreted as “the more games, the more wins”.

Definition 2.2 (Gradual Pattern). A gradual pattern $\text{GP}$ is a set of gradual items: $\text{GP} = \{(i_1, v_1), \ldots, (i_n, v_n)\}$.

For example, $\{(\text{games}, \uparrow), (\text{wins}, \uparrow), (\text{losses}, \downarrow)\}$ is a GP that can be interpreted as “the more games, the more wins, the less losses”.

In the following, let a data set $D$ with attribute $A$ and $n$ tuples and $x$ be a tuple such that: $x \in D$ and $A(x)$ denotes the value $A$ takes for $x$. A gradual dependency $(A, \uparrow)$ (the more $A$) holds if $\forall x, x' \in D$, $A(x) < A(x')$ and vice-versa for $(A, \downarrow)$. The quality of gradual item sets is measured by frequency support. The frequency support $\text{sup}$ of a GP is: “the proportion of tuple couples that respect the constraints by all the gradual items in the pattern”, as given by the formula:

$$\text{sup}(\text{GP}) = \frac{1}{|D'|} \left| \{(x, x') \in D' / \forall j \in [1, n] A_j(x) * A_j(x')\} \right|$$

(1)

where $\text{GP} = \{(A_1, v_1), \ldots, (A_n, v_n)\}$, $* \in \{<, >\}$ and $D'$ is transaction data set (derived from $D$) defined by attributes' gradual dependencies $(A_n, v_n)$ with the corresponding tuple pairs $(x, x')$ that respect them.

Similarly, support describes the extent to which a GP holds for a given data set. Therefore, given a user-specified threshold $\sigma$ a GP is said to be frequent if:

$$\text{sup}(\text{GP}) \geq \sigma$$

(2)

2.2 Gradual rule traversal techniques

With regards to GP mining, two aspects complicate its mining process: (1) determining frequency support and (2) the complementary notion of gradual item sets (for each attribute, there exist two gradual item sets). These two challenging aspects are brought about by the nature of patterns and thus traversal because we need to compare lines. As can be seen in Table 2a, b, association rule mining deals with the transactions of a data set while gradual rule mining deals with the attributes of a data set.

To elaborate, in association rule mining a single transaction is enough to determine the occurrence frequency of an item set. For instance in Table 2a given transaction $t_3$ only, we can tell that item set $\{d\}$ is not frequent. In gradual rule mining at least two or more transactions are needed to determine the frequency occurrence of an item set. For instance in Table 2b given transaction $t_3$ only, either $(a, \uparrow)$ or $(a, \downarrow)$ is possible. Further in order to mine gradual item sets, the complementary notion requires that; for each attribute, there exist two gradual item sets. For instance attribute $a$ creates gradual item sets $(a, \uparrow)$ and $(a, \downarrow)$.

In spite of this, techniques have been developed that allow for gradual rule mining through BFS and DFS strategies. In the case of BFS strategy for gradual rule mining, [8, 20] propose approaches GRITE (GRadual ITEm-set Extraction) and GRAANK (GRAdual rANKing) that represent a gradual item set as a binary matrix of orders...
Table 3 Binary matrices $M_{G_i}$, $M_{G_j}$ for gradual items: (a) $i1 = (a, \downarrow)$, (b) $i2 = (b, \uparrow)$

| r  | r1 | r2 | r3 | r4 |
|----|----|----|----|----|
| r1 | 0  | 1  | 1  | 1  |
| r2 | 0  | 0  | 1  | 1  |
| r3 | 0  | 0  | 0  | 1  |
| r4 | 0  | 0  | 0  | 0  |

Table 4 (a) Transactional encoding of data set in Table 2b, (b) reduced data set of the encoded data set (where tids - transaction ids with similar item sets and weight—number of appearance)

| Id   | Item-sets                      |
|------|--------------------------------|
| $t_{i1,i2}$ | $\{(a, \downarrow), (b, \uparrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i1,i3}$ | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i1,i4}$ | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i2,i3}$ | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i2,i4}$ | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i3,i4}$ | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |

| Tids   | Weight | Item-sets                      |
|--------|--------|--------------------------------|
| $t_{i1,i2}$ | 1      | $\{(a, \downarrow), (b, \uparrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i1,i3}$ | 1      | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i1,i4}, t_{i3,i4}$ | 3      | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |
| $t_{i2,i4}$ | 1      | $\{(a, \downarrow), (b, \downarrow), (c, \downarrow), (d, \downarrow)\}$ |

and this enables the use of a level-wise BFS technique to generate $k$-itemset gradual candidates.

For example using Table 2b, let us consider a 1-itemset gradual rule $i1 = (a, \downarrow)$. We have the list of tuples respecting this item set as $G_{i1} = \{(r1, r2), (r1, r3), (r1, r4), (r2, r3), (r2, r4), (r3, r4)\}$. The set of orders may be modeled using a binary matrix of size $4 \times 4$ (where tuple pairs respecting the rule take the value of 1 and those not respecting take the value of 0) as illustrated in Table 3a. Table 3b shows the binary matrix of gradual item $(b, \uparrow)$.

In the case of DFS strategy for gradual rule mining, [24] extends LCM (Linear time Closed item set Miner) presented by [35, 36] to propose ParaMiner. Before applying a DFS technique to build a (FP-Tree) frequent pattern tree (a set enumeration tree of long frequent items recursively grown from short ones) of gradual item sets; ParaMiner encodes a numeric data set into transactional data set containing gradual dependencies of attributes with the corresponding concordant pairs as shown in Table 4a.

Both GRAANK, and ParaMiner techniques have their merits and demerits. GRAANK’s relies on the bitmap representation of gradual variations using a binary matrix. This makes arithmetic computations of candidates very efficient. ParaMiner employs a parallel algorithm which relies on a data set reduction technique to improve its efficiency. Table 4b illustrates how a data set is reduced by grouping similar gradual items (with weights) and removing those whose support do not surpass the user-specified threshold.

However, since GRAANK is based on a BFS strategy, it generates large numbers of candidates when dealing with data sets having huge numbers of attributes. This may lead to a combinatorial candidate explosion which overwhelms the algorithm. ParaMiner is based on DFS whose major drawback is finding the parent node of the tree with the longest length. DFS-based techniques employs recursion (which has an exponential computational complexity) to achieve this.
In the section that follows, we propose a heuristic solution to the problem of combinatorial candidate explosion and finding parent node of an FP-Tree.

3 Review works

In this section, we present a review of some population-based optimization techniques that may be applied to the case of GP mining. In GP mining, evaluating all possible GP candidates in order to find those that are frequent is computationally expensive, especially when a huge number of attributes is involved [8, 13, 20, 25].

This problem may be alleviated by utilizing population-based optimization techniques (such as genetic algorithm (GA) [22], particle swarm optimization (PSO) [19] and ant colony optimization [10]) to find the frequent GP candidates without having to generate the whole candidate population.

GA is among the first population-based optimization techniques to be proposed in history [14, 22]. It is a meta-heuristic inspired by natural selection and the “survival of the fittest” systems originally proposed by Charles Darwin in the book titled “The Origin of Species”. Based on the fitness value, in a competing environment, only the strongest individuals survive and attract mating partners. These individuals that are considered strong mate to produce offsprings and the good attributes from both parents may produce even stronger offsprings. In this way species become more well suited in their environment [14, 16, 22, 27].

Data mining is one application area of GAs. For instance, Saravanan et al., [30] propose a novel approach mining sequential patterns using GA. Kabir et al., [18] conduct a comparative study of GA-based techniques versus Apriori-based techniques for mining maximal frequent patterns and they prove that GA-based algorithms perform computationally better than their Apriori-based counterparts.

PSO is a meta-heuristic algorithm implementation that (originally proposed by [19]) to imitate the movements of birds or schools of fishes as they search for food) simulates the behaviors and movements of swarms in order to iteratively optimize a numeric problem [27, 28]. PSO can also be classified as an evolutionary computation technique that is inspired by the metaphor of social interaction and communication (such as bird flocking and fish schooling) [31].

In the realm of pattern mining, [23, 31] discover and prove that a PSO-based frequent pattern (FP)-growth technique is more efficient in mining fuzzy frequent patterns than traditional FP-growth techniques.

ACO, as originally described by [10], is a general-purpose meta-heuristic approach for optimizing various combinatorial problems. It exploits the behavior of a colony of artificial ants in order to search for approximate solutions to discrete optimization problems [4, 6, 11, 29, 32]. The application areas for ACO are vast; for instance in the telecommunication domain, [33] employed it to optimally load balance circuit-switched networks.

It is important to highlight that GA and PSO based techniques use best solution and global best solution to generate new individual solutions [27]. On the contrary, ACO based techniques use pheromone levels to generate individual solutions [29]. In regards to GP mining, possible GP candidate solutions, more often, do not converge to a single best solution. Rather, candidate solutions are made up of combinations of 2 or more 1-item set GP solutions (see Sect. 2). Therefore, the problem is more of a ‘combinatorial problem’ than a ‘fitness value’ problem. In the sections that follow, we illustrate how ACO is better suited than GA and PSO in optimizing the problem of finding GP candidates.

4 Ant colony optimization for GP extraction

4.1 Ant colony optimization

ACO imitates the positive feedback reinforcement behavior of biological ants as they search for food [9, 10]. According to [15], ACO utilizes a set of artificial ants to probabilistically contrive solutions S through a collective memory, pheromones stored in matrix T, together with a problem specific heuristic η.

First, we introduce formal mathematical notations taken from literature concerning ACO. We use the traveling salesman problem (TSP) [17] in order to describe these preliminary mathematical notations [9, 10].

Given a set of n towns, the TSP problem can be stated as the problem of finding the shortest route that visits each town once. The path between town i and j can be represented as $d_{ij}$. Let $\{b_{ij}(t) \mid i, j \in [1, \ldots, n]\}$ be the number of ants in town i at time t and $q = \sum_{i=1}^{n} b_{ii}(t)$ be the total number of ants. Each ant is a simple agent that:

- chooses a town to go to based on the probability that is a function of the town distance and amount of pheromone present on the connecting edge;
- is prohibited from returning to already visited towns until a tour visit of all towns is completed; and
- deposits pheromone trails on each visited edge $(i, j)$, once it completes the tour.

Let $\tau_{ij}(t)$ be the pheromone intensity on edge $(i, j)$ at time t and each ant at time t chooses the next town to visit, where it will be at time $t+1$. We define q movements made by q ants in the interval $(t, t+1)$ a single iteration of the ACO algorithm. Therefore, every n iterations of the algorithm each ant completes a visit of all the towns (tour cycle) and
at this point the pheromone intensity is updated according to the formula:
\[
\tau_{ij}(t + n) = (1 - \rho) \cdot \tau_{ij}(t) + \Delta \tau_{ij}
\]
(3)
where \(\rho\) (usually < 1) is a coefficient such that \((1 - \rho)\) represents the evaporation of pheromone intensity between time \(t\) and \(t + n\),
\[
\Delta \tau_{ij} = \sum_{k=1}^{q} \Delta \tau_{ij}^k
\]
where \(\Delta \tau_{ij}^k\) is the quantity per unit of length of pheromone substance laid on edge \((i, j)\) by the \(k\)th ant between time \(t\) and \(t + 1\) and it is given by:
\[
\Delta \tau_{ij}^k = \left\{ \begin{array}{ll}
\frac{C}{L_k} & \text{if } k\text{th ant uses path}(i, j) \\
0 & \text{(between time } t \text{ and } t + n) \\
\end{array} \right.
\]
where \(C\) is a constant and \(L_k\) is the tour length of the \(k\)th ant.
In order to satisfy the constraint that an ant visits all the \(n\) towns, each ant is associated with a tabu list (tabu\(_k\)) that stores the towns already visited by the ant up to time \(t\) and forbids it from revisiting them until the tour cycle is complete. At the end of the tour cycle, the tabu list is used to compute the distance of the path followed by the ant then, it is emptied and the next tour cycle begins.
The probability \(p_{ij}^k(t)\) of the \(k\)th ant moving from town \(i\) to \(j\) as shown in Eq. (4):
\[
p_{ij}^k(t) = \left\{ \begin{array}{ll}
\frac{[\tau_{ij}(0)^\alpha \cdot |\eta_{ij}|^\beta]}{\sum_{k \in \text{allow}_k} [\tau_{ij}(0)^\alpha \cdot |\eta_{ij}|^\beta]} & \text{if } j \in \text{allow}_k \\
0 & \text{otherwise} \\
\end{array} \right.
\]
where \(\eta_{ij} = 1/d_{ij}\) and \(\text{allow}_k = \{N - \text{tabu}_k\}\) and \(\alpha\) and \(\beta\) are parameters that control the relative importance of the pheromone intensity against visibility.
In this paper, we will consider one variant of ACO called ‘MAX–MIN ant system’ [34] to optimize both BFS and DFS for the case of GP mining. The remainder of this section is organized as follows: we describe existing BFS and DFS traversal strategies for extracting gradual rules; we describe ACO approaches for optimizing these strategies and propose the corresponding mathematical notations.

### 4.2 ACO for BFS candidate generation

In order to apply ACO to the GRAANK approach, we need to identify a suitable heuristic representation of the gradual item set candidate generation problem that can be solved using ACO.

Given a set of \(n = \{a_1, a_2, \ldots\}\) attributes of a data set \(D\) (as shown in Table 5), GRAANK seeks to find frequent GPs \(M = \{m_1, m_2, \ldots, m_k\}\) (where \(\forall m \in M : m \subseteq n\)) by generating and testing numerous candidates.

**Example 4.1** We consider a sample graphs of artificial ants moving on the edges of gradual items \(a+, a-, b+, b-, c+\) and \(c-\) (where + denotes ‘increasing’, – denotes ‘decreasing’, \(a+\) implies \((a, 1)\), \(a-\) implies \((a, -1)\), and minimum support threshold \(\sigma = 0.5\)) as shown in Fig. 2. For the sake of simplicity, we remove all edges connecting to nodes (or gradual items) whose frequency support \((\sup)\) < \(\sigma\) and assume that \(\sup(a+, c+) > \sigma\) but \(\sup(a+, c+, b+) < \sigma\).

In order to make an accurate interpretation of the ACO for BFS, assume that the distances of all node edges are equal. Let us consider what happens at regular discrete time intervals: \(t = 0, 1, 2, \ldots\). Suppose that 12 new ants come to nodes \(a\) and \(b\) respectively at time \(t = 0\) and another 119 new ants to nodes \(a\) and \(b\) at time \(t = 1, 2, \ldots\). Each ant travels at a speed of 2 edge distances per time interval and that by moving along at time \(t\) it deposits pheromones of intensity 1 only if all the nodes visited up to that time \(t\) form a gradual item set whose support \(\geq \sigma\). The pheromone evaporates by a factor \((1 - \rho)\) between successive time interval \((t + 1, t + 2)\).

At time \(t = 0\) no path has any pheromone intensity; therefore, the 12 new ants at nodes \(a\) and \(b\) choose all paths with equal probability as shown in Fig. 2b. Since \(\sup(a+, c+, b+) < \sigma\) the ant from \(a\) that chose edges between nodes \(a+, c+\) and \(b+\) and the 3 ants from \(b\) that chose edges between nodes \(b+, c+\) do not deposit any pheromone on any of these edges.

At time \(t = 1\) 119 new ants arrive at nodes \(a\) and \(b\). The 119 new ants at node \(a\) will find a pheromone intensity 6 on the path that leads to \(a-\) deposited by ants that used it previously from node \(a-\) and a pheromone intensity of 15 at node \(a+\); 6 on the path that leads to \(b+\) (3 deposited by ants that departed from \(a+\) and 3 deposited by ants that arrived from \(b+\)), 9 on the path that leads to \(c+\) (3 deposited by ants that departed from \(a+\) and 6 deposited by ants that arrived from \(c+\)). The probability of the ants choosing to move towards node \(a-\) is 6/21, that of moving towards node \(a+\) is 15/21, that of moving towards node \(b+\) is 6/15 and that of moving towards \(c+\) 9/15. The same is true for the new 119 ants at node \(b\), as shown in Fig. 2c. Therefore, the best variation routes (or candidates) are: \(\{a+, c+, b-\}, \{a+, b+, \{a-, c-\}\}. 

| Id | a | b | c |
|----|---|---|---|
| r1 | 5 | 30 | 43 |
| r2 | 4 | 35 | 33 |
| r3 | 3 | 40 | 42 |
| r4 | 1 | 50 | 49 |
GPCG problem, we define each node of the ACO graph as a singleton gradual item derived from its attribute and the variation edge distance $d_{ij}$ between nodes $i, j$ is given by formula in Eq. (5):

$$d_{ij} = \begin{cases} 1 & \text{if } \text{sup}(i) \geq \sigma \text{ and } \text{sup}(j) \geq \sigma \\ \infty & \text{otherwise} \end{cases}$$

where $\text{sup}(i)$ is the frequency support of gradual item set $i$, $\sigma$ is a user-specified minimum support threshold—see Sect. 2.1. The possibility of gradual item $(i, j)$ being frequent is either visible ‘1’ or not visible ‘$\infty$’.

Let $\langle b_i(t) \mid i = 1, 2, \ldots, n \rangle$ be the number of ants at node $i$ at time $t$ and $q = \sum_{i=1}^{n} b_i(t)$ be the total number of ants. Each ant is a simple agent that:

- chooses a node to go to based on the probability that is a function of the amount of pheromone present on the connecting edge;
- is prohibited from returning to already visited nodes until a tour visit of all nodes is completed (this is managed by a tabu list); and
- deposits pheromone trails on all visited edges if support of all visited nodes $\text{sup}(N) \geq \sigma$, once it completes the tour. This is because a gradual candidate is formed by combining all the visited nodes into a set; and if the candidate is not frequent then its path is also not appealing. Lastly, every edge $(i, j)$ is removed if: $\text{sup}(i) < \sigma$ and $\text{sup}(j) < \sigma$.

Let $\tau_{ij}(t)$ be the pheromone intensity on edge $(i, j)$ at time $t$ and each ant at time $t$ chooses the next node to visit, where it will be at time $t+1$. We define $q$ movements made by $q$ ants in the interval $(t, t+1)$ a single iteration of the ACO algorithm. Therefore, every $n$ iterations of the algorithm each ant completes a visit of all valid nodes (tour cycle) and at this point the pheromone intensity is updated according to the formula in Eq. (6).

$$\tau_{ij}(t + n) = \begin{cases} (1 - \rho) \cdot \tau_{ij}(t) + \Delta \tau_{ij} & \text{if } \text{sup}(N_n) \geq \sigma \\ 0 & \text{otherwise} \end{cases}$$

where $\rho$ and $\Delta \tau_{ij}$ are similar to Equation (3) and $N_n$ is the set of all nodes visited by the ant at end of $n$ iterations $N_n = \{i_t\}_{t=0}^{n}$.

Similarly, each ant is associated with a tabu list $\text{tabu}_i$ that stores the nodes already visited up to time $t$ and forbids the ant from revisiting them until the tour cycle is complete. In addition, the tabu list also keeps track of node edges which do not meet the minimum frequency support requirement.

In GP mining, a potential candidate must not include any infrequent singleton gradual item set (anti-monotonicity property [25]). Therefore, when apply ACO to the case of
\( \eta_{ij} = 1/d_{ij} \) is consequently affected. The probability \( p^k_{ij}(t) \)
of the \( k \)th ant moving from node \( i \) to \( j \) becomes:

\[
p^k_{ij}(t) = \begin{cases} 
\frac{[\tau_{ij}(t)^{\alpha}]}{\sum_{k \in \text{allow}_j} [\tau_{ij}(t)^{\alpha}]} & \text{if } j \in \text{allow}_k \text{ and } (\text{sup}(i), \text{sup}(j) \geq \sigma) \\
0 & \text{otherwise}
\end{cases}
\]  

(7)

where \( \text{allow}_j \) and \( \alpha \) are similar to Eq. (4).

Therefore, the probability \( p^k_{ij}(t) \) is a function of pheromone intensity at time \( t \): \( \tau_{ij}(t) \) (which instructs that the edge \((i, j)\) with the most ant traffic should be selected with highest probability—implementing an autocatalytic process). In GP mining visibility between nodes \( i \) and \( j \) can be compared to the possibility of forming a candidate by combining node \( i \) and \( j \); and it is either ‘visible’ to the ant if both nodes are frequent or ‘invisible’ to the ant if one of the nodes is infrequent. So, \( \eta_{ij} \) is denoted as either 1 or 0 for ‘visible’ or ‘invisible’ respectively.

4.2.2 ACO-GRAANK algorithm

ACO-GRAANK algorithm is an optimized version of the GRAANK algorithm [20] and it inherits the efficient binary representation of concordant tuple pairs, but applies ACO to generate candidate item sets.

We take the position that a gradual item set may also be referred to as a pattern solution. In that case all possible gradual item set solutions \( S^n \) are admissible and can be generated based on the pheromone matrix \( (T_{aj}) \). Notwithstanding, all the gradual item sets in a generated solution will be evaluated and the solution updated with only valid item sets.

First we mention that initially there exists an equal chance for any attribute \( A \) (of data set \( D \)) to either increase (+) or decrease (−) or be irrelevant (×). As the algorithm acquires more knowledge about valid patterns, the possibilities of the 3 options are adjusted accordingly. We propose an artificial pheromone matrix of 2-Dimensional whose size is shown in Eq. (8). The matrix contains knowledge about pheromone proportions of the 3 gradual options for each attribute.

\[
T_{aj} = q \times 3
\]

(8)

where \( q \) : number of attributes and \( j \in \{+,-,\times\} \)

At the beginning all artificial pheromones \( p_{aj} \) in matrix \( T_{aj} \) are initialized to 1, then they are updated as follows at time interval \((t, t + 1)\):

- every generated gradual item set solution is evaluated and only valid solutions are used to update the artificial pheromone matrix. Invalid solutions are stored and used to reject their super-sets; and

- in a given valid solution, each gradual item set is used to update the corresponding pheromone \( p_{aj}(t) \) (where \( j \) is either + or −) by formula: \( p_{aj}(t) = p_{aj}(t) + 1 \).

The probability rule \( P_{aj}(t) \) is given by calculating the proportions of its artificial pheromones \( p_{aj} \), as shown in Eq. (9).

\[
P_{aj}(t) = \frac{p_{aj}}{\sum_{k \in \{+,-,\times\}} p_{ak}(t)}
\]

(9)

The main steps of ACO-GRAANK algorithm are:

1. Build binary matrices of 1-itemset from input data set \( D \) (described in GRAANK [20]).
2. Generate gradual item sets from pheromone matrix \( \tau_{aj} \) and validate each generated item set by comparing its support against the minimum support \( \sigma \).
3. If the generated pattern is valid, use gradual items to update the pheromone matrix.
4. Repeat steps 2-3 until the algorithm starts to generate similar gradual item sets.

4.2.3 Computational complexity of ACO-GRAANK

The big-O notation [7, 27, 37] is used to analyze the computational complexity of ACO-GRAANK. For every GP candidate that is generated, ACO-GRAANK algorithm builds binary matrices, performs a binary AND operation on the matrices and calculates support [20]. We formulate the problem and show its complexity.

Problem formulation Given a dataset \( D \) with \( m \) attributes and \( n \) objects, we can generate numerous GP candidates each with \( k \) item sets (where \( k \leq m \)). For example, if \( D \) has 3 attributes \( \{a, b, c\} \), one of the candidates may be \((a, 1), (b, 1)\) which has 2 item sets. For each candidate, we build binary matrices of every item set as shown in Table 3.

Next we perform a binary AND on these binary matrices and calculate frequency support. Using big-O notation building binary matrices results in a complexity of \( O(k \cdot n^2) \). The binary AND operation and support calculation have small complexities in comparison to that of the latter operation.

Search space size It is important to highlight that, GP candidate search space grows almost exponentially as the number of attributes increase. Given a huge search space, the aim of ACO-GRAANK algorithm is to find useful candidates (candidates whose frequency support \( \geq \sigma \)) and ignore useless candidates in the search space. This can be achieved by minimizing the number of algorithm iterations required to learn the valid candidates. Therefore, if an ACO-GRAANK algorithm iterates \( x \) times, it generates \( x \) GP candidates and its computational complexity is \( O(x \cdot k \cdot n^2) \).
Table 6 (a) Sample data set \( D \) and (b) its corresponding sorted reduced transactional data set with minimum length of \( t \cdot d \) is 3—see Sect. 2.2

| Id | a | b | c | d |
|----|---|---|---|---|
| r1 | 5 | 30 | 43 | 97 |
| r2 | 4 | 35 | 33 | 86 |
| r3 | 3 | 40 | 42 | 108 |
| r4 | 1 | 50 | 49 | 27 |

| Item | Tids |
|------|------|
| \((a, 1)\) | \{t_{(1,2)}, t_{(1,3)}, t_{(1,4)}, t_{(2,3)}, t_{(2,4)}, t_{(3,4)}\} |
| \((b, 1)\) | \{t_{(1,2)}, t_{(1,3)}, t_{(1,4)}, t_{(2,3)}, t_{(2,4)}, t_{(3,4)}\} |
| \((c, 1)\) | \{t_{(1,4)}, t_{(2,3)}, t_{(2,4)}, t_{(3,4)}\} |
| \((d, 1)\) | \{t_{(1,2)}, t_{(1,4)}, t_{(2,3)}, t_{(2,4)}\} |

4.3 ACO for DFS FP-tree search

Similarly, for the purpose of applying ACO to a DFS-based approach, we identify a suitable heuristic representation to the problem of finding the parent node of the longest FP-tree.

Given a set of \( n = \{a_1, a_2, \ldots, a_k\} \) attributes of a data set where each attribute has a set of tuples \( a_k = \{r_1, r_2, \ldots\} \) (as shown in Table 6a), gradual DFS techniques seek to find frequent GPs \( M = \{m_1, m_2, \ldots\} \) (where \( \forall m \in M : m \subseteq n \) by encoding the data set into a transactional data set (as shown in Table 6b) and recursively searching the transactional data set for the longest FP-Tree.

Example 4.2 We consider a sample graphs of artificial ants moving on the edges of nodes (or tuples) \( r_1, r_2, r_3 \) and \( r_4 \) as shown in Fig. 3. For the case of DFS in GP mining, we propose to use the occurrence count of tuples in the encoded transactional data set to determine the length of distance between nodes (such as \( d_{ij} = \frac{1}{1+\sum_{t \in S} t_{ij}} \)).

In order to make an accurate interpretation of ACO for BFS, assume that the paths between nodes \( r_1 \) and \( r_3 \), \( r_3 \) and \( r_2 \) have longer lengths than paths between nodes \( r_1 \) and \( r_4 \), \( r_4 \) and \( r_2 \) (as indicated by the distances in Fig. 3a). Let us consider what happens at regular discrete time intervals: \( t = 0, 1, 2 \ldots \). Suppose that 12 new ants come to node \( r_1 \) and 12 new ants come to node \( r_2 \) at each time interval. Each ant travels at a speed of 0.5 distance per time interval and that by moving along at time \( t \) it deposits pheromones of intensity 1, which completely and instantaneously evaporates in the middle of the successive time interval \( (t+1, t+2) \).

At time \( t = 0 \) no path has any pheromone intensity and there are have 12 new ants at node \( r_1 \) and 12 new ants at node \( r_2 \). The ants select between nodes \( r_4 \) and \( r_3 \) randomly;
therefore, on average 6 ants from each node move towards $r_4$ and 6 ants towards $r_3$—as shown in Fig. 3b.

At time $t = 1$ there are 12 new ants at node $r_1$ and 12 new ants at node $r_2$. The 12 new ants at node $r_1$ will find a pheromone intensity of 6 on the path that leads to node $r_3$ deposited by the 6 ants that used it previously from node $r_1$ and a pheromone intensity of 12 on the path that leads to node $r_4$ deposited by 6 ants that used it coming from node $r_1$ and 6 that arrived from node $r_2$. Therefore, the ants will find the path to node $r_4$ more desirable than that to node $r_3$. The probability of the ants choosing to move towards node $r_4$ is 2/3, while that of moving towards node $r_3$ is 1/3 and 8 ants will move towards node $r_4$ and 4 ants towards node $r_3$. The same is true for the 12 new ants at node $r_2$, as shown in Fig. 3c. Therefore, the most appealing FP-Tree is $\{(r_1, r_1), (r_2, r_2)\}$ which both appear in transactions with gradual items $\{(b, \uparrow), (c, \uparrow), (d, \downarrow)\}$.

4.3.1 Proposed mathematical notations of ACO for GPFP-tree problem

The gradual pattern FP-Tree (GPFP-Tree) problem can be stated as:

"the problem of finding the longest FP-Tree from which frequent gradual patterns can be constructed."

In the case of DFS for GP mining, we inherit the concept of encoding data sets into transactional data sets and harness this to represent the GPFP-Tree problem as a slightly different version of the TSP problem. Given a set of $n$ tuples of a data set, we seek to find the cheapest route that visits the most tuples once. The path between tuples $i$ and $j$ may be represented as $d_{ij}$ and an instance of the GPFP-Tree may also be represented as a graph composed of tuples (or nodes) and edges. Each edge distance is derived from the formula:

$$d_{ij} = \frac{1}{1 + \sum (r_i, r_j)_{\text{count}}}$$

(10)

where $\sum (r_i, r_j)_{\text{count}}$ is the occurrence count of tuple pair $(r_i, r_j)$ in an encoded transactional data set.

Let $\langle b_i(t) \mid i = 1, 2, \ldots , n \rangle$ be the number of ants in node $i$ at time $t$ and $q = \sum_{i=1}^{n} b_i(t)$ be the total number of ants. Each ant is a simple agent that:

- chooses a node to go to based on the probability that is a function of the node distance and amount of pheromone present on the connecting edge;
- is prohibited from returning to already visited nodes until a tour visit of all allowed nodes is completed (this is managed by a taboo list); and
- deposits pheromone trails on each visited edge $(i, j)$, once it completes the tour.

Let $\tau_{ij}(t)$ be the pheromone intensity on edge $(i, j)$ at time $t$ and each ant at time $t$ chooses the next node to visit, where it will be at time $t + 1$. We define $q$ movements made by $q$ ants in the interval $(t, t + 1)$ a single iteration of the ACO algorithm. Therefore, every $n$ iterations of the algorithm each ant completes a visit of all the nodes (tour cycle) and at this point the pheromone intensity is updated according to the formula in Eq. (11).

$$\tau_{ij}(t + n) = (1 - \rho) \cdot \tau_{ij}(t) + \Delta \tau_{ij}$$

(11)

where $\rho$ and $\Delta \tau_{ij}$ are similar to Eq. (3).

Similar to the original ACO, each ant is associated with a tabu list $\text{tabu}_k$ that stores the nodes already visited by the ant up to time $t$ and forbids it from re-visiting them until the tour cycle is complete. At the end of the tour cycle, the tabu list is used to compute the distance of the path followed by the ant then, it is emptied and the next tour cycle begins. The probability $p^k_{ij}(t)$ of the $k$th ant moving from node $i$ to $j$ as shown in Eq. (12).

$$p^k_{ij}(t) = \left\{ \begin{array}{ll} \frac{[\tau_{ij}(t)]^a \cdot [\eta_{ij}]^b}{\sum_{k \in \text{allow}_k}[\tau_{ij}(t)]^a \cdot [\eta_{ij}]^b} & \text{if } (i, j) \in \text{allow}_k \\ 0 & \text{otherwise} \end{array} \right.$$  

(12)

where $\text{allow}_k$ and $a$ and $b$ are similar to Eq. (4).

Therefore, in the GPFP-Tree problem probability $p^k_{ij}(t)$ is a trade-off between $\eta_{ij}$ and $\tau_{ij}(t)$. Put differently, it is a trade-off between a greedy constructive heuristic and an autocatalytic process.

4.3.2 ACO-ParaMiner algorithm

The ACO-ParaMiner algorithm which is an optimized version of ParaMiner technique proposed by [24]. ACO-ParaMiner inherits the technique of transactional encoding from ParaMiner, but applies ACO to find the longest FP-Tree.

We propose the following ACO optimizations to the ParaMiner approach (proposed in [24]): (1) in addition to reducing the transactional data set by combining similar item set transactions and removing infrequent ones, we construct a cost matrix of all corresponding nodes. (2) we replace the recursive function with a non-recursive heuristic function that quickly learns the longest FP-tree with the help of the cost matrix.

To illustrate, we use data set $\mathcal{D}$ in Table 7. In order to remove infrequent items, the transactional data set is sorted by item occurrence as shown in Table 8b. If we set the minimum length of $\text{tids}$ to 3, we remove infrequent items as illustrated in Table 9a.

We define a 2-Dimensional cost matrix whose size is: $C_{ij} = n \times n$ (where $n$ is number of tuples in the original numeric data set, such as Table 7) and initialize it to 1. Using the sorted and reduced transactional data set
Table 7 Sample data set $\mathcal{D}$

| Tids | $a$ | $b$ | $c$ | $d$ |
|------|-----|-----|-----|-----|
| r1   | 5   | 30  | 43  | 97  |
| r2   | 4   | 35  | 33  | 86  |
| r3   | 3   | 40  | 42  | 108 |
| r4   | 1   | 50  | 49  | 27  |

Table 8 (a) example of a reduced transactional data set, (b) sorted items by occurrence

| Tids | $w$ | Item-sets |
|------|-----|-----------|
| $t_{(1,2)}$ | 1 | \{(a, 1), (b, 1), (c, 1), (d, 1)\} |
| $t_{(1,3)}$ | 1 | \{(a, 1), (b, 1), (c, 1), (d, 1)\} |
| $t_{(1,4)}$, $t_{(2,4)}$ | 3 | \{(a, 1), (b, 1), (c, 1), (d, 1)\} |
| $t_{(3,4)}$ | 1 | \{(a, 1), (b, 1), (c, 1), (d, 1)\} |

| Item | Tids |
|------|------|
| (a, 1) | \{$t_{(1,2)}, t_{(1,3)}, t_{(1,4)}, t_{(2,3)}\$, \{$t_{(2,4)}, t_{(3,4)}\$.} |
| (b, 1) | \{$t_{(1,2)}, t_{(1,3)}, t_{(1,4)}, t_{(2,3)}\$, \{$t_{(2,4)}, t_{(3,4)}\$.} |
| (c, 1) | \{$t_{(1,4)}, t_{(2,3)}, t_{(2,4)}, t_{(3,4)}\$.} |
| (d, 1) | \{$t_{(1,2)}, t_{(1,3)}\$.} |
| (a, 1) | \{∅\} |
| (b, 1) | \{∅\} |

Table 9 (a) sorted reduced transactional data set, (b) corresponding cost matrix

| Item | Tids |
|------|------|
| (a, 1) | \{$t_{(1,2)}, t_{(1,3)}, t_{(1,4)}, t_{(2,3)}\$, \{$t_{(2,4)}, t_{(3,4)}\$.} |
| (b, 1) | \{$t_{(1,2)}, t_{(1,3)}, t_{(1,4)}, t_{(2,3)}\$, \{$t_{(2,4)}, t_{(3,4)}\$.} |
| (c, 1) | \{$t_{(1,4)}, t_{(2,3)}, t_{(2,4)}, t_{(3,4)}\$.} |
| (d, 1) | \{$t_{(1,2)}, t_{(1,4)}, t_{(2,3)}\$, \{$t_{(3,4)}\$.} |

| $r$ | r1 | r2 | r3 | r4 |
|-----|----|----|----|----|
| r1  | 1  | 1/4| 1/3| 1/5|
| r2  | 1  | 1  | 1  | 1  |
| r3  | 1  | 1  | 1  | 1  |
| r4  | 1  | 1  | 1  | 1  |

in Table 9a, we update elements of cost matrix that correspond to the inverse of occurrence count of tids as shown in Eq. (13). Table 9b illustrates the cost matrix of Table 9a. In this case we observe that tids with more occurrences have whose size is shown in Eq. (14). This matrix contains knowledge about the pheromone proportions of every node of the data set.

\[
C_{ij} = \frac{1}{1 + \sum (ri, rj)_{\text{count}}}
\]

where \((ri, rj)_{\text{count}}\) is the occurrence count of tid pair \((ri, rj)\).

We define an artificial 2-dimensional pheromone matrix $\mathbf{P}$, where $pi$ is an artificial 2-dimensional pheromone proportion of every node of the data set. The pheromone proportion of node $i$ to node $j$ is defined as $p_{ij} = \frac{1}{1 + \sum (ri, rj)_{\text{count}}}$, where $ri, rj$ are the occurrence count of tid pair $(ri, rj)$.
\[ I_{ij} = n \times n \] (14)

where \( n \) is the number of tuples in numeric data set.

At the beginning all artificial pheromones \( p_{ij} \) in matrix \( I_{ij} \) are initialized to 1, then they are updated as follows at time interval \( (t, t + 1) \):

- every generated node, is used to retrieve the corresponding attributes from the sorted reduced transactional data set. A set intersection of the \( tids \) of all these attributes provides an FP-Tree whose length is tested against the specified threshold;
- if the length of the FP-tree surpasses the specified threshold, the pheromones corresponding to the \( tids \) are incremented by 1: \( p_{ij} = p_{ij}(t) + 1 \); and
- if the length of the FP-Tree falls below the specified threshold, the pheromones corresponding to the \( tids \) are evaporated by a factor: \((1 - \rho) \cdot p_{ij}(t)\).

We propose a probabilistic rule that allows us to learn a parent node of the longest FP-Tree in a non-recursive manner. This rule is shown in Eq. (15).

\[
P_{ij}(t) = \frac{p_{ij}(1/C_{ij})}{\sum_{k=1}^{n} p_{ik}^e(t)/(C_{ij}^e)}
\]

The main steps for the ACO-ParaMiner algorithm are:

1. Encode a data set \( D \) into a transactional data set and reduce it by combining similar item sets and the removing infrequent ones. Use the reduced transactional data set \( D_{red} \) to construct a Cost Matrix \( C \).
2. Generate nodes that have high probability of being parent nodes using the Cost Matrix and Pheromone Matrix \( I_{ij} \).
3. For each generated node, retrieve corresponding attributes from \( D_{red} \) and determine the intersection of their \( tids \). If the length of the resulting intersection surpasses the minimum length threshold \( \zeta \), then update the pheromones.
4. Repeat steps 2-3 until the algorithm starts to generate similar nodes.

4.3.3 Computational complexity of ACO-ParaMiner

We use big-O notation [7, 37] to analyze the computational complexity of ACO-ParaMiner algorithm. Firstly, ACO-ParaMiner constructs a cost matrix of all corresponding nodes. Secondly, ACO-ParaMiner learns the longest FP-Tree with the help of the cost matrix. We formulate the problem and show its complexity.

**Problem formulation** Given a dataset \( D \) with \( m \) attributes and \( n \) objects, an encoded transactional data set is built and used to construct a cost matrix of size \( n \times n \). In order to build an encoded transactional data set (as shown in Table 8a) every tuple object is paired with all other tuple objects and the resulting gradual patterns recorded. Each tuple pairing results into an iteration that identifies gradual patterns; therefore, the computation complexity is equal to the total number of pairings \( O\left(\frac{n^2}{2}\right)\). Next, using this data set \( tids \) are generated for every possible singleton gradual item and those whose length do not satisfy the user-defined threshold are discarded (as shown in Tables 8b and 9a respectively). It should be remembered that for every attribute, 2 singleton gradual items are possible (for example, for attribute \( a \) we have \((a, 1)\) and \((a, 1)\)). Every singleton gradual item results into an iteration that finds \( tids \) from the previous data set and may be equated to a constant complexity of \( C_1 \); therefore, complexity is equal to the total number of the items \( (O(2m \cdot C_1)) \). The computational complexity of building a cost matrix from resulting transactional data set is minuscule in comparison to the previous processes. On that account, the total complexity is \( O\left(\frac{n^2}{2} + 2m \cdot C_1\right)\).

**Search space size** According to [24], the process of finding the longest FP-Tree using a transactional data set is recursive. It can be shown that a recursive function introduces a large search space which makes it computationally expensive. The aim of ACO-ParaMiner is to find the longest FP-Tree using a non-recursive function. As a result, if an ACO-ParaMiner algorithm generates \( x \) FP-Trees. Every FP-Tree that is generated, is applied on the reduced encoded transactional data set (as shown in Table 9a) in order to join all singleton gradual items that share that FP-Tree to form a gradual pattern. Next, the \( tid \) length of the formed gradual pattern compared against a user-defined threshold. This process may be equated to a constant complexity of \( C_2 \). The overall complexity of ACO-ParaMiner becomes \( O\left(\frac{n^2}{2} + 2m \cdot C_1 + x \cdot C_2\right)\).

Notably, although complexities of \( C_1 \) and \( C_2 \) are almost constant for every iteration; in the long run, they produce a significant computational expense.

5 Experiments

In this section, we present an experimental study of computational performance of our algorithms. We implemented BFS-based GRAANK (as described in [20]) and ACO-GRAANK; and DFS-based ParaMiner (as described in [24]) and ACO-ParaMiner algorithms in Python language. These ACO-based algorithms can also be described as algorithms that are built on top of a population-based optimization technique (that is ACO). As discussed in Sect. 3, GA and PSO are alternative population-based optimization techniques which may also be applied in GP mining. As a result, we implement GA-GP and PSO-GP algorithms as
alternative algorithms to the ACO-based algorithms and we perform experiments in order to compare their computational performances. In addition to these experiments, we also compare the performance of these population-based algorithms against their classical GP mining algorithm counterparts (that is GRAANK and ParaMiner). All experiments were conducted on a HPC (High Performance Computing) platform Meso@LR\(^1\). We used one node made up of 14 cores and 128GB of RAM.

### 5.1 Source code

The Python (language) source code of the 6 algorithms is available at our GitHub repository: [https://github.com/owuordickson/ant-colony-gp.git](https://github.com/owuordickson/ant-colony-gp.git). Since [24] does not provide a Python implementation of ParaMiner, we extend a LCM Python source code ([https://github.com/scikit-mine.git](https://github.com/scikit-mine.git)) to implement our ParaMiner algorithm.

### 5.2 Data set description

Table 10 shows the features of the data sets used in the experiments for evaluating the computational performance of the algorithms. All the data sets described in this table are numerical data sets. The ‘Breast Cancer’ data set, obtained from UCI Machine Learning Repository (UCI-MLR) [26], is composed of 10 quantitative predictors and binary variable indicating the presence or absence of breast cancer. The predictors are recordings of anthropometric data gathered from the routine blood analysis of 116 participants (of whom 64 have breast cancer and 52 are healthy).

The ‘Cargo 2000’ data set, obtained from UCI-MLR [21], describes 98 tracking and tracing events that span 5 months of transport logistics execution. The ‘Power Consumption’ data set, obtained from UCI-MLR [12], describes the electric power consumption in one household (located in Sceaux, France) in terms of active power, voltage and global intensity with a one-minute sampling rate between 2006 and 2010.

The ‘Directio’ data set is one of 4 data sets obtained from OREMES’s data portal\(^2\) that recorded swell sensor signals of 4 buoys near the coast of the Languedoc-Roussillon region in France between 2012 and 2019 [5].

### 5.3 Experiment results

In this section, we present our experiment results which reveal the computational behavior of GRAANK, ParaMiner, ACO-GRAANK, ACO-ParaMiner, GA-GP, PSO-GP algorithms when applied on the 4 data sets described in Sect. 5.2. The results are a mean different test runs of each algorithm on each data set at different parameter settings. All these results are available at: [https://github.com/owuordickson/meso-hpc-lr/tree/master/results/swarm](https://github.com/owuordickson/meso-hpc-lr/tree/master/results/swarm).

#### 5.3.1 Experiment 1: population-based algorithms

In this experiment we tune the respective parameters of ACO-GRAANK, GA-GP, and PSO-GP algorithms and apply each to GP mining. We apply the 3 algorithms on the 4 data sets described in Sect. 5.2, we set the maximum iteration count to 100, minimum support threshold \((\sigma)\) to 0.5 and we observe their behaviors as they mine for GPs. Each test run is repeated several times in order to confirm each behavior. We compute the mean count of extracted GPs by each algorithm and plot the results in Fig. 4. More detailed results are shown in Tables 11, 12, 13 and 14.

It is important to note that for the case of ACO-GRAANK, GA-GP, and PSO-GP, the term ‘best cost’ or ‘best position’ refers to a value that is computed after every iteration using the respective fitness function which aims to find GPs with high frequency support. In other words, the higher the frequency support of a GP, the lower its ‘cost’ or ‘position’.

In the case of ACO-GRAANK algorithm, we varied the evaporation factor value between 0.1 to 0.9. It is important to recall that evaporation factor determines how fast unused pheromone trails disappear (see Sect. 4.1). The lower the evaporation factor value, the faster the unused pheromone trails disappear. In this experiment, ACO-GRAANK algorithm fetches almost the same number of GPs regardless of the evaporation factor value.

In the case of GA-GP, we varied the ‘Proportion of Children’ (PC) factor value between 0.1 to 0.9. It is important to recall that the variable ‘PC’ factor values determine how fast unused pheromone trails disappear (see Sect. 4.1). The lower the evaporation factor value, the faster the unused pheromone trails disappear. In this experiment, ACO-GRAANK algorithm fetches almost the same number of GPs regardless of the evaporation factor value.

In the case of GA-GP, we varied the ‘Proportion of Children’ (PC) factor value between 0.1 to 0.9. It is important to recall that the variable ‘PC’ factor values determine how fast unused pheromone trails disappear (see Sect. 4.1). The lower the evaporation factor value, the faster the unused pheromone trails disappear. In this experiment, ACO-GRAANK algorithm fetches almost the same number of GPs regardless of the evaporation factor value.

1. [https://meso-lr.umontpellier.fr](https://meso-lr.umontpellier.fr).
2. [https://data.oreme.org](https://data.oreme.org).
increasing PC value increases the chances of producing higher quality offsprings whose fitness correspond to finding GPs with high frequency supports. In the case of PSO-GP, we varied coefficients C1 and C2 values between 0.1 to 0.9. C2 coefficient value gives the importance of the global best position, while C1 coefficient value gives the importance of the personal best position [31]. Regardless of the values of C1 and C2 coefficients, PSO-GP algorithm finds almost the same number of GPs which have highest frequency support within a few iterations.

Given these results, we observe that ACO-GRAANK algorithm is best suited for finding numerous frequent GPs, while GA-GP and PSO-GP are best suited for finding only GPs with the highest frequency support.

### 5.3.2 Experiment 2: memory utilization

This experiment identifies the algorithms with better memory utilization while extracting GPs. For this experiment, we apply ACO-GRAANK (ACG), ACO-Para-Miner (ACP), GA-GP (GAG), GRAANK (GRA), Para-Miner (PRM) and PSO-GP (PSO) algorithms on B&C, Buoys, C2K and UCI data sets. We perform multiple test runs of each algorithm on every data set as we vary minimum support threshold ($\sigma$) from 0.5 to 0.9, we compute the mean of allocated memory-size
Table 12 **Buoys data set**: computational comparison of ACG, GAG, GRA and PSO at different support (sup) values

| Sup | Alg | Run-time (s) | No. of gradual patterns | Memory (KiB) |
|-----|-----|-------------|------------------------|--------------|
|     |     | Std. dev | Best | Mean | Worse | Std. dev | Fewest | Mean | Most | Std. dev | Fewest | Mean | Most | Std. dev | Min | Mean | Max |
| 0.5 | ACG | 26.737  | 136.200 | 165.933 | 188.000 | 6.658 | 24.0 | 28.333 | 36.0 | 8.083 | 156.0 | 163.333 | 172.0 |
|     | GAG | 16.626  | 124.500 | 140.133 | 157.600 | 0.577 | 1.0 | 1.333 | 2.0 | 2.887 | 103.0 | 104.667 | 108.0 |
|     | PSO | 6.034   | 106.400 | 113.367 | 116.900 | 0.577 | 1.0 | 1.667 | 2.0 | 0.306 | 95.1 | 95.367 | 95.7 |
| 0.6 | ACG | 23.426  | 110.200 | 130.633 | 156.200 | 2.646 | 16.0 | 18.000 | 21.0 | 9.074 | 146.0 | 156.333 | 163.0 |
|     | GAG | 18.251  | 116.100 | 129.267 | 150.100 | 0.000 | 1.0 | 1.000 | 1.0 | 0.577 | 104.0 | 104.333 | 105.0 |
|     | PSO | 6.034   | 106.400 | 113.367 | 116.900 | 0.577 | 1.0 | 1.667 | 2.0 | 0.819 | 93.0 | 93.900 | 94.6 |
| 0.7 | ACG | 20.676  | 138.500 | 153.900 | 177.400 | 2.517 | 7.0 | 9.333 | 12.0 | 3.786 | 156.0 | 158.667 | 163.0 |
|     | GAG | 9.582   | 108.100 | 118.933 | 126.300 | 0.000 | 1.0 | 1.000 | 1.0 | 0.577 | 103.0 | 103.333 | 104.0 |
|     | GRA | 46.634  | 531.400 | 567.567 | 620.200 | 0.000 | 40.0 | 40.000 | 40.0 | 0.000 | 122.0 | 122.000 | 122.0 |
|     | PSO | 0.985   | 28.380  | 30.297 | 31.210 | 8.737 | 12.0 | 21.667 | 29.0 | 0.577 | 199.0 | 199.667 | 200.0 |
| 0.6 | ACG | 2.823   | 18.700  | 21.637 | 24.330 | 2.309 | 10.0 | 11.333 | 14.0 | 1.793 | 99.8 | 100.933 | 103.0 |
|     | GAG | 0.789   | 14.480  | 15.380 | 15.950 | 0.000 | 1.0 | 1.333 | 2.0 | 0.000 | 101.0 | 102.000 | 103.0 |
|     | GRA | 15.437  | 156.700 | 170.567 | 187.200 | 0.000 | 2.0 | 2.000 | 2.0 | 0.000 | 103.0 | 104.000 | 105.0 |
|     | PSO | 5.184   | 13.930  | 19.403 | 24.240 | 0.000 | 1.0 | 1.000 | 1.0 | 7.425 | 93.5 | 95.267 | 104.0 |

Bold values highlight the best results under each category

Table 13 **C2K data set**: computational comparison of ACG, GAG, GRA and PSO at different support (sup) values

| Sup | Alg | Run-time (s) | No. of gradual patterns | Memory (KiB) |
|-----|-----|-------------|------------------------|--------------|
|     |     | Std. dev | Best | Mean | Worse | Std. dev | Fewest | Mean | Most | Std. dev | Fewest | Mean | Most | Std. dev | Min | Mean | Max |
| 0.5 | ACG | 0.987   | 29.250 | 30.297 | 31.210 | 8.737 | 12.0 | 21.667 | 29.0 | 0.577 | 199.0 | 199.667 | 200.0 |
|     | GAG | 4.319   | 24.650 | 27.370 | 32.350 | 0.577 | 1.0 | 1.333 | 2.0 | 0.000 | 1.0 | 1.000 | 1.0 |
|     | GRA | 12.156  | 389.700 | 400.167 | 413.500 | 0.000 | 604.0 | 604.000 | 604.0 | 0.000 | 122.0 | 122.000 | 122.0 |
|     | PSO | 0.985   | 28.380 | 30.297 | 31.210 | 8.737 | 12.0 | 21.667 | 29.0 | 0.577 | 199.0 | 199.667 | 200.0 |
| 0.6 | ACG | 2.823   | 18.700 | 21.637 | 24.330 | 2.309 | 10.0 | 11.333 | 14.0 | 1.793 | 99.8 | 100.933 | 103.0 |
|     | GAG | 0.789   | 14.480 | 15.380 | 15.950 | 0.000 | 1.0 | 1.333 | 2.0 | 0.000 | 101.0 | 102.000 | 103.0 |
|     | GRA | 15.437  | 156.700 | 170.567 | 187.200 | 0.000 | 2.0 | 2.000 | 2.0 | 0.000 | 103.0 | 104.000 | 105.0 |
|     | PSO | 5.184   | 13.930 | 19.403 | 24.240 | 0.000 | 1.0 | 1.000 | 1.0 | 7.425 | 93.5 | 95.267 | 104.0 |

Bold values highlight the best results under each category
values and we plot the graphs shown in Fig. 5. More detailed results are shown in Tables 11, 12, 13 and 14.

In Fig. 5a, we observe that ACP algorithm requires the highest memory allocation. It is important to mention that ACP and PRM algorithms yield an ’Out of Memory Error’ and no run-time when applied on Buoys, C2K and UCI data sets. GRAANK yields the same error when applied on Buoys data set with at least 150,000 tuples (Fig. 5b) and UCI data set with more than 47,000 tuples. ACG, GAG and PSO do not yield an ’Out of Memory Error’ in any of these experiments as shown in Fig. 5a–d.

5.3.3 Experiment 3: computational performance

Similar to Sect. 5.3.2, this experiment compares the computational performance of the 6 algorithms (in terms of run-time) when applied on the 4 data sets. Using the results, we plot graphs in Fig. 6 and more detailed results are shown in Tables 11, 12, 13, 14.

In Fig. 6a, we observe that PRM algorithm slowest run-time in comparison to ACG, ACP, GAG and GRA algorithms. In Fig. 6b, GRA yields an ’Out of Memory Error’ when \( \sigma \leq 0.5 \) and it has the slowest run-time when \( \sigma \geq 0.7 \). Similarly in Fig. 6c, GRA has the slowest run-time that reduces as \( \sigma \) increases. In Fig. 6d, ACG, GRA, GAG and PSO algorithms have almost the same run-times. It should be noted that ACP and PRM algorithms yield an ’Out of Memory Error’ when applied to Buoys, C2K and UCI data sets.

5.3.4 Experiment 4: consistent gradual patterns

This experiment identifies consistent GPs extracted by all algorithms from data sets: B&c, C2K, Buoys with 15,000 tuples, UCI with 47,000 tuples. The results are shown in Table 15.

5.4 Discussion of results

5.4.1 Memory usage

We observe that generally ParaMiner and ACO-ParaMiner algorithms have higher requirements for memory usage than ACO-GRAANK, GA-GP, GRAANK and PSO-GP algorithms. This phenomenon explains why ParaMiner and ACO-ParaMiner algorithms (1) yield ’Memory Error’ outputs for relatively large data sets and (2) have slower run-times for relatively small data sets. As described in Sect. 2.2, ParaMiner is based on a DFS strategy whose drawbacks (for the case of GP mining) are as follows:

1. A numeric data set has to be encoded into a transactional data set before DFS can be applied. This significantly increases the size of the data set and consequently size of usage memory required; and
2. DFS employs a recursive approach to find all the longest paths. However, we notice from the experiment results that ACO-ParaMiner has improved run-time since it uses a non-recursive heuristic approach to achieve this.

For the case of ACO-GRAANK, GA-GP, GRAANK and PSO-GP algorithms, their relatively low memory usages may be largely attributed to the modeling of tuple orders using a binary matrix. However, for relatively large data sets GRAANK algorithm also yields a ’Memory Error’ output because it has to hold all possible candidate GPs’ binary matrices in memory. ACO-GRAANK, GA-GP and PSO-GP algorithms have low memory requirements for relatively large data sets because they generate fewer GP candidates and learn to reject the useless candidates stochastically. This implies that they only store few binary matrices in memory. It should be noted that for data sets that can be handled by GRAANK algorithm, (for these data sets) ACO-GRAANK, GA-GP and PSO-GP algorithms have slightly higher memory usage requirements than the latter due to the stochastic process that they use to search for useful GP candidates.

5.4.2 Execution run-time

We observe that generally BFS-based algorithms (ACO-GRAANK, GA-GP, GRAANK and PSO-GP) have relatively faster run-times than DFS-based algorithms (ACO-ParaMiner and ParaMiner). This mainly is due to low memory usage requirements explained in Sect. 5.4.1. Also, we observe that ACO-GRAANK, GA-GP and PSO-GP algorithms have relatively faster run-times than GRAANK algorithm because they employ a heuristic approach that learns useful candidate GPs quicker than GRAANK’s classical level-wise approach.
Fig. 5 Plot of memory usage against minimum support

(a) B&C: #attributes=10 #objects≈116

(b) Buoys: #attributes=21 #objects≈15K

(c) C2K: #attributes=98, #objects≈4K

(d) UCI: #attributes=9 #objects≈47K
For the case of GRAANK algorithm, we observe that in almost all the 4 data sets (B&C, Buoys, C2K and UCI) the run-times reduce significantly as \( \sigma \) increases from 0.5 to 0.9. This behavior may be attributed to the fact that the number of possible GP candidates always reduce as \( \sigma \) increases. However, ACO-GRAANK, GA-GP and PSO-GP algorithms do not generate and test all possible GP candidates. Instead, they use stochastic processes to learn only the useful GP candidates and this phenomenon explains their relatively constant run-times as \( \sigma \) increases.

### 5.4.3 Extracted gradual patterns

We observe that generally the number of GPs extracted by ParaMiner algorithm is relatively constant against different supports because it searches for all closed patterns (which correlates as many attributes as possible) [24], while the number of GPs extracted by GRAANK algorithm varies significantly with support since it sear-ches for all frequent patterns (especially minimal and maximal patterns). ACO-GRAANK, GA-GP and PSO-GP algorithms ignore minimal patterns and heuristically search for frequent maximal GPs (which are fewer but each is composed of several minimal patterns) and ACO-ParaMiner stochastically searches for closed GPs.

The standard deviation is generally greater for the case of ACO-GRAANK, ACO-ParaMiner, GA-GP and PSO-GP algorithms than for the case of GRAANK and ParaMiner algorithms. This may imply that GRAANK and ParaMiner algorithms are more stable since produce almost similar number of patterns for the same experiment at different runs.

### 6 Conclusions and future works

In this paper, we describe an ant colony optimization tech-nique for BFS-based and DFS-based approaches for mining gradual patterns. Given the computational results in Sect. 5, we establish that ACO-GRAANK and ACO-ParaMiner, GA-GP and PSO-GP algorithms out-perform their classical counterparts GRAANK and ParaMiner in terms of faster run-time and low memory usage requirements. However, this comes at the cost of extracting relatively fewer number of patterns.

As can be seen in Sect. 5.3.3, the results reveal that generally standard deviation of number of gradual patterns for ACO-GRAANK and ACO-ParaMiner is larger than that of their classical counterparts. This shows that these ant-based algorithms are slightly unstable; therefore, they extract...
Fig. 6  Plot of run-time against minimum support

(a) B&C: #attributes=10  #objects≈116

(b) Buoys: #attributes=21  #objects≈15K

(c) C2K: #attributes=98,  #objects≈4K

(d) UCI: #attributes=9  #objects≈47K
different number of gradual patterns at different runs. Future work may involve improving their stability.

Another future work may involve extending the ant-based approach to probabilistically select tuple portions of a data set in order to estimate dominant gradual patterns. At the present time all the tuples of a data set are used to determine frequent gradual patterns. Under those circumstances, the computational complexity of extracting gradual patterns is directly proportional to the number of tuples in the data set. In contrast, if portions of few tuples are used to estimate the frequent gradual patterns, the computational complexity may be reduced to a constant regardless of the the data set size.3

Acknowledgements The authors would like to thank the French Government through the office of Co-operation and Cultural Service (Kenya) and the office of Campus France (Montpellier) for their involvement in creating the opportunity for this work to be produced. This work has been realized with the support of the High Performance Computing Platform: MESO@LR, financed by the Occitanie / Pyrénées-Méditerranée Region, Montpellier Mediterranean Metropole and Montpellier University.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Aryadinata YS, Laurent A, Sala M (2013) M2LG? Mining multiple level gradual patterns. Int J Comput Inf Eng 7(3):353–360
2. Aryadinata YS, Lin Y, Barcellos C, Laurent A, Libourel T (2014) Mining epidemiological dengue fever data from Brazil: a gradual pattern based geographical information system. Comput Oper Res 443 CCIS:414–423. https://doi.org/10.1007/978-3-319-08855-6_42
3. Berzal F, Cubero JC, Sanchez D, Vila MA, Serrano JM (2007) An alternative approach to discover gradual dependencies. Int J Uncertain Fuzziness Knowl Based Syst 15(05):559–570. https://doi.org/10.1142/S0218488X0700487X
4. Blum C (2005) Ant colony optimization: introduction and recent trends. Phys Life Rev 2(4):353–373. https://doi.org/10.1016/j.phyrev.2005.10.001
5. Bouchette F (2019) OREME: the coastline observation system. https://oreme.org/observation/ltc/. Accessed June 2021
6. Cicirello VA, Smith SF (2001) Ant colony control for autonomous decentralized shop floor routing. In: Proceedings 5th international symposium on autonomous decentralized systems, pp 383–390. https://doi.org/10.1109/ISADS.2001.917443
7. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. MIT Press, Cambridge
8. Di-Jorio L, Laurent A, Teisseire M (2009) Mining frequent gradual itemsets from large databases. In: Advances in intelligent data analysis VIII. Springer, Berlin, pp 297–308. https://doi.org/10.1007/978-3-642-03915-7_26
9. Dorigo M, Birattari M (2010) Ant colony optimization. Springer US, Boston, pp 36–39. https://doi.org/10.1007/978-0-387-30164-8_22
10. Dorigo M, Maniezzo V, Colomi A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern Part B (Cybern) 26(1):29–41. https://doi.org/10.1109/3477.484436
11. Dorigo M, Stützle T (2019) Ant colony optimization: overview and recent advances. Springer International Publishing, Cham, pp. 311–351. https://doi.org/10.1007/978-3-319-91086-4_10
12. Dua D, Graff C (2019) UCI machine learning repository. http://archive.ics.uci.edu/ml. Accessed June 2021
13. Fiot C, Masseglia F, Laurent A, Teisseire M (2009) Evolution patterns and gradual trends. Int J Intell Syst 24(10):1013–1038. https://doi.org/10.1002/int.20269
14. Goldberg DE, Holland JH (1988) Genetic Algorithms and Machine Learning. Mach Learn 3:95–99. https://doi.org/10.1023/A:1022602019183
15. Hartmann SA, Runkler TA (2008) Online optimization of a color sorting assembly buffer using ant colony optimization. In: Kalcicic J, Nicole S (eds) Operations Research Proceedings 2007. Springer, Berlin, pp 415–420
16. Holland JH (1992) Genetic algorithms. Sci Am 267(1):66–73
17. Jünger M, Reinelt G, Rinaldi G (1995) Chapter 4 the traveling salesman problem. In: Network models, handbooks in operations research and management science, vol 7, pp 225–330. Elsevier. https://doi.org/10.1016/S0927-0507(05)80121-5
18. Kabir MMI, Xu S, Kang BH, Zhao Z (2015) Comparative analysis of genetic based approach and apriori algorithm for mining maximal frequent item sets. In: 2015 IEEE congress on evolutionary computation (CEC), pp 39–45. https://doi.org/10.1109/CEC.2015.7256872
19. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN’95—international conference on neural networks, vol 4, pp 1942–1948. https://doi.org/10.1109/ICNN.1995.488968
20. Laurent A, Lesot MJ, Rifqi M (2009) Graank: exploiting rank correlations for extracting gradual itemsets. In: Proceedings of the 8th international conference on flexible query answering systems, FQAS ’09. Springer, Berlin, pp 382–393. https://doi.org/10.1007/978-3-642-04957-6_33
21. Metzger A, Leitner P, Ivanović D, Schmieders E, Franklin R, Carro M, Dustdar S, Pohl K (2015) Comparing and combining predictive business process monitoring techniques. IEEE Trans Syst Man Cybern Syst 45(2):276–290
22. Mirjalili S (2019) Genetic algorithm. Springer International Publishing, Cham, pp 43–55. https://doi.org/10.1007/978-3-319-93025-1_4
23. Mishra S, Mishra D, Satapathy SK (2011) Particle swarm optimization based fuzzy frequent pattern mining from gene expression data. In: 2011 2nd international conference on computer and communication technology (ICCCIT-2011), pp 15–20. https://doi.org/10.1109/ICCCIT.2011.6075204

Table 15 Consistent gradual patterns

| Data set | Consistent gradual patterns |
|----------|-----------------------------|
| B&C      | {(Insulin, ↓), (HOMA, ↓)}, support: 0.94 |
| C2K      | {(1.rfc_1.p, ↑), (1.rfc_1.e, ↑)}, support: 0.837 |
| Buoys    | {(Tz, ↓), (Tav, ↓)}, support: 0.945 |
| UCI      | {(Global_activepower, ↑), (Global_intensity, ↑)}, support: 0.954 |

[3] https://meso-lr.umontpellier.fr
24. Negrevergne B, Termier A, Rousset MC, Méhaut JF (2014) Paraminer: a generic pattern mining algorithm for multi-core architectures. Data Min Knowl Discov 28(3):593–633. https://doi.org/10.1007/s10618-013-0313-2

25. Owuor D, Laurent A, Orero J (2019) Mining fuzzy-temporal gradual patterns. In: 2019 IEEE international conference on fuzzy systems (FUZZ-IEEE). IEEE, New York, NY, USA, pp 1–6. https://doi.org/10.1109/FUZZ-IEEE.2019.8858883

26. Patricio M, Pereira J, Crisóstomo J, Matafome P, Gomes M, Seiça R, Caramelo F (2018) Using resistin, glucose, age and BMI to predict the presence of breast cancer. BMC Cancer 18(1):29. https://doi.org/10.1186/s12885-017-3877-1

27. Pourpanah F, Lim CP, Wang X, Tan CJ, Seera M, Shi Y (2019) A hybrid model of fuzzy min-max and brain storm optimization for feature selection and data classification. Neurocomputing 333:440–451. https://doi.org/10.1016/j.neucom.2019.01.011

28. Rajamohana S, Umanaheswari K (2018) Hybrid approach of improved binary particle swarm optimization and shuffled frog leaping for feature selection. Comput Electr Eng 67:497–508. https://doi.org/10.1016/j.compeleceng.2018.02.015

29. Runkler TA (2005) Ant colony optimization of clustering models. Int J Intell Syst 20(12):1233–1251. https://doi.org/10.1002/int.20111

30. Saravanan M, Jyothis VL (2014) A novel approach for sequential pattern mining by using genetic algorithm. In: 2014 international conference on control, instrumentation, communication and computational technologies (ICCICCT), pp 284–288. https://doi.org/10.1109/ICCICCT.2014.6992971

31. Shruti M, Debahuti M, Sandeep Ku S (2012) Fuzzy frequent pattern mining from gene expression data using dynamic multi-swarm particle swarm optimization. Procedia Technol 4:797–801.

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.