Abstract: This paper describes an application to form teams from a given set of employees. The teams are assigned to perform a task. To accomplish a task a team has to fulfill specific requirements which have to been minded while assigning teams to task. The goal is the forming of a chain of task meeting specific requirements and assigning a team to each task able to solve it. These chains have to be optimized for a synergy value representing how well employees work together. There is one synergy value for each possible pair of employees.

Key words: Swarm intelligence, ant colony optimization, team formation, metaheuristic.

1. Introduction of ACO

The formation of ant streets and the capability of ants to solve shortest path puzzles [1] can be used in computer science. ant colony optimization is an IT approach to solve computable problems by imitating the behavior of ants. One important behavior pattern of ants for ACO is stigmergy, the indirect communication by manipulating the environment [2], [3].

Ant workers search for food sources [4] and mark a found path to the food source with pheromones. More pheromone on a path higher the likelihood that other ants take this path. Ants validate a path by laying down additional pheromones. A shorter path is completed by more ants in a given time than a longer one which results in more pheromone on short than on long paths. A colony of ants is capable of finding short paths to food sources. This leads to the formation of ant streets.

The first approaches of using this behavior for computational problems were algorithms called Ant System(s) [5], [6]. ACO is a met heuristic framework first described by Dorigo & Di Caro in 1999 [7]. The program discussed in this paper is based on ideas and methods of ACO.

2. Presentation of the Problem

A company or organization has a known set of employees. To solve tasks, these employees form teams. A team is built by two employees. employees have qualifications in five different fields, indicated by values from 1 (low qualification) to 5 (high qualification). Each employee has a specific cost value, which indicates how expensive his workforce is.

Each possible team has a synergy value, which indicates how well two employees perform tasks together. Discussing optimization possibilities for this synergy value is the topic of this paper. If two employees work in a team, their qualifications supplement each other. By this mean, the qualification values for a team are determined by using the better value from the employees for every field.
A task is characterized by an input and an output value and a qualification set, which is needed to fulfill the task. A team must have the same or a higher value for every of the 5 fields compared to the requirement of the task. Input and output values are characters from A to H. One employee can only work in one team and at one task at the time for simulation, all values are generated random.

3. Complexity

\( x \) is the number of employees and \( y \) is the number of tasks. The number of possible team combinations, regarding the fact, that one team is build by two different employees is

\[
x \times (x-1) \over 2
\]

One building block for the chain of tasks needed to fulfill the requirements is a combination of one team and one task.

The number of these building blocks:

\[
\frac{x \times (x-1)}{2} \times y = \frac{1}{2} x y (x^2 - x)
\]

This number ascends linear for higher number of tasks and in a cubic magnitude for higher number of employees.

It is not practical to assign one pheromone value to every block to perform the ACO, because of the fast growing number of these building blocks. Instead two optimizations are performed, one for connecting a chain of task and another one to find fitting members. The algorithm for the tasks will optimize for low skill requirements to make the tasks solvable for a variety of teams. The algorithm for the teams will optimize for a good average synergy and, if needed, for low cost of the teams.

4. Program Structure

The skill value of a team or a task is represented by a single int value to reduce needed space (see Fig. 1).

```
A 32 bit integer:
000000000000 0011 0111 1111 0011 0111
/ \
All five skill values encoded

0000 = skill of one
0001 = skill of two
0011 = skill of three
0111 = skill of four
1111 = skill of five
```

Fig. 1. Five skill values encapsulated in a 32 bit integer.

The skill value of a team is represented by the skill values of both team members connected with a bitwise OR operation.

A team fits the skill requirements of a task, if the skill values of task and team connected with a bitwise AND operation is equal to the skill requirements.

The information about and pheromone values for teams and tasks are stored in lightweight, pointerless, array-based trees. These array trees allow the random selection of elements, represented by an index. The chance for one element to be selected is depending of the pheromone value associated with the element. An array tree consists of an array with pheromone data and some metadata to speed up the selection process.
This metadata consists of the sum of all deployed pheromone, the root element of the tree and the height of
the tree. The classes team array tree and task array tree extend array tree to provide additional information
like skill or synergy values for a selected element. This information is stored in additional arrays and can be
accessed with the index representing the selected element gained thru the random selection.

The shown class diagrams don’t depict all methods, variables or classes (see Fig. 2).

Fig. 2. Basic storage classes.

The following image depicts a simple example of an array tree with 8 entries. knots is an array of double.
The tree-like depiction of knots is needed later to explain specific operations on an array tree. The
pheromone values of the entries are not stored directly in memory. The knots array with its implicated tree
structure is stored instead (see Fig. 3).

Fig. 3. Example of 8 elements stored in an array tree.

The value of each knot represents the sum of all values under its left child knot. The pheromone values
are leafs which are not stored in memory because their values can be calculated if the path from the root to
the leaf is known. After initialization the array tree serves mainly for fast picking of one id considering the
specific pheromone value of each element (team or task) and updating the tree structure to achieve new
pheromone values.

The selection of a random element with consideration of the pheromone values is represented by
following pseudo code.

The method random(x, y) returns a value v with x ≤ v ≤ y (see Fig. 4).
Fig. 4. Pseudo code for the selection of a random element of an array tree based on pheromone values.

The left knots list is needed in the update process. The random select on a team array tree or task array tree will result in the returning of other types of simple DTO objects. Team and Task are those DTOs and transport more specific information together with a tree path (see Fig. 5-Fig. 6).

The next figure shows the selection process of one element from the example array tree. After every iteration of an application run the array trees need to be updated to include the impact of found team and task chains for future ones.

The method increase pheromones (tree path) is called to update the pheromone values after the main
algorithm assigned pheromone to add values to the tree paths (see Fig. 7).

The pseudo code for the update process is:

```java
pheromoneSum = pheromoneSum + pheromoneToAdd
for each int i in leftKnot
    knot[i] = knot[i] + pheromoneToAdd
```

Fig. 7. Pseudo code for a pheromone update within an array tree.

The selected element is updated. The value is increased by 2 as shown in the following graphic (see Fig. 8):

![Fig. 8. The previous example of an array tree with one updated element.](image)

These array trees are of fixed size but cheaper in memory than a pointer based tree. However, the main advantage of an array tree is the function get Tree Path By Pheromone where Tree Path is a simple DTO representing one element of the array tree. The possibility for a Tree Path to be returned by this function correspond with its pheromone value compared to the sum of pheromone values from the complete array tree. It isn't needed to form a specific set of Tree Sets and do any pheromone calculation before selecting one element like in a roulette wheel attempt (see Fig. 8).

All teams are presorted by their skill values. Teams with the same skill value are stored in a team array tree. A team array tree head contains references to multiple team array trees and extends the get team by pheromone method over all contained team array trees (see Fig. 9).

![Fig. 9. Important attributes and methods of the class team array tree head.](image)

For every possible skill value there is a team array tree head, holding references to all team array trees with skill values, fitting requirements of tasks with this skill value.

If a task is selected to be solved, the team array tree head with the same skill value is used to select one of all team who are able to solve the task.

This saves a lot of running time compared to selecting a random team and testing the skill value. If this random team doesn't fit the requirements a new team must be randomly selected. To find a team for a task with high requirements it is possible to perform millions of random team selections before finding one fitting team. If the presorted teams are used, reselecting the team for a task is only required when one of the
team members is already used in another team solving another task.

Other important classes are the employee task and team repo, which contain instances of the storage classes and provide methods for data management and the my parameter class, which is a central point for changing the different parameters to alter the behavior of the program.

The Messene twister is the random number generator of the application and the classes pair and skill util provide small aids for different other classes.

The classes ant and ant colony are discussed in depth in the following part (see Fig. 10).

5. Algorithm

There are two modes to run the application. The first mode tries to find an optimal chain from a given start input for the tasks to a given end output. This modus leads to very short chains, most times only one building block in length. The second mode forms chains with a given length e.g. 10 tasks and 10 teams. The start input is set, the end output not.

After starting the application test data is generated. The tasks are equal distributed by the possible input values. One task array tree is generated for every input to provide the ability to always select tasks with matching input value to append the current task chain. The output and skill values for every task are generated random. Next the employees are generated with random cost and skill values.

While creating the team repo all possible teams get a random skill value assigned. All team IDs with the same skill value are stored together in one team array tree. For every team ID a random synergy value is generated. The team array tree heads for all possible skill values are formed after this.

After generating the test data an ant colony is created and its method search for optima is called. The ant colony repeats the following steps for a given number of iterations.

Ant-Threads are created and started. The main-method waits for all Ants to stop. The ant colony collects all important information like found task and team chains from every ant. The ant colony checks for a possible new best average synergy value and updates all pheromone values as preparation for the next iteration (see Fig. 11).

An Ant first forms a task chain matching the requirements of the mode the application is running. The method get next task from the task repo is used to find task depending on their pheromone matrix for a specific output. After completing a chain of tasks the skill requirements of each task is used to find a fitting team by calling get next team from the team repo. This procedure is stopped if the accumulated cost value exceeds the given maxima.

One major feature of an ACO implementation is the evaporation of pheromone over time [8]. Reducing all pheromone values after every iteration is very costly in computation time, especially if there is a high number of values. Instead of reducing the already deployed pheromone by a specific percentage it is much faster to simply increase the new deployed pheromone every iteration. One simple multiplication to
increase newly deployed pheromone can replace the need to change up to millions of values in the repositories.

![Fig. 11. Class diagrams of ant and ant colony.](image)

E.g. reducing all pheromone values by 10% every iteration is equal to higher newly deployed pheromone by 11.1%.

Before every iteration a maximum addable pheromone value is calculated by the ant colony and given to the Ants to accomplish this computational speed advantage.

After every iteration every Ant has to calculate the deployed pheromone for every used team and task.

The calculation for the pheromone for one team is split up for two different parameters, the synergy value of the team and its cost. The pheromone calculation for the task depends only on the skill value necessary to fulfill the task.

The variable max is the pheromone value added to the entire chain of task or teams.

\[ p_{\text{Syn}} \text{ and } p_{\text{Cost}} \text{ represent the ratio between optimization the team chain for synergy or cost. } p_{\text{Syn}} + p_{\text{Cost}} = 1 \]

syn is the synergy value of the current processed team.

\[ \text{synSum is the sum of all synergy values for a team chain.} \]

costDiff is the difference between the cost value of a team and the maximum possible cost value.

\[ \text{costDiffSum is the sum of all these cost differences for a team chain.} \]

\[ x \text{ is the pheromone value added to the team.} \]

\[
x_{\text{Syn}} = \max \times p_{\text{Syn}} \times \frac{\text{syn}}{\text{synSum}} \tag{3}
\]

\[
x_{\text{Cost}} = \max \times p_{\text{Cost}} \times \frac{\text{costDiff}}{\text{costDiffSum}} \tag{4}
\]

\[ x = x_{\text{Syn}} + x_{\text{Cost}} \tag{5} \]

Skill diff is the difference between the sum of all skill values of a task and the maximum possible skill sum.

\[ \text{Skill diff sum is the sum of all skill diff values of a task chain.} \]

\[ Y \text{ is the pheromone value added to the task.} \]

\[
y = \max \times \frac{\text{skillDiff}}{\text{skillDiffSum}} \tag{6} \]

The information about which team and task has to be updated with which pheromone value will be read
and processed by the ant colony.

6. Parameter

There are multiple parameters to modify the behavior of the application.

The number of ants generated by the ant colony enables the application to effectively use multi core CPUs.

More iterations result in better results but higher the runtime of the application. The rate at which the deployed pheromone is increased every iteration can manipulate the moment when the application stops finding better results. A higher rate speeds up this process but also increases the possibility for the application to get stuck in a local maximum. A low rate and a high number of iterations gives the best results.

The ratio between the maximum cost of an employee and the maximum budget can change the importance of the cost fraction of the pheromone calculation. Team chains not fulfilling the budget demand are dropped which extend the runtime of the application. Increasing ratio for cost is the way to counteract this behavior. But lowering the ratio of synergy can also slow down the optimization for the best average synergy value.

7. Example

After discussing the general cycle of the application a simplified example will follow.

The skill value will only consist of one value from 1 to 3.

There are 5 employees and 12 tasks. The maximum cost value for an employee is 10. The 5 employees result in 10 possible teams (see Table 1-Table 10).

<table>
<thead>
<tr>
<th>Team-ID</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>34</th>
<th>35</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Cost</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>16</td>
<td>9</td>
<td>12</td>
<td>6</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Synergy</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Pheromone</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

The ant colony uses 3 Ants to form chains of 3 tasks beginning with the input 1. The maximum budget is 33. At this point, every Task has the same likelihood to get picked.

The ants pick task chains.

| Task chain Ant 1: | 1(2) | 4(2) | 2(2) |
| Task chain Ant 2: | 2(2) | 6(1) | 9(2) |
| Task chain Ant 3: | 2(2) | 5(1) | 3(3) |

After forming the task chains every ant maps Teams onto the tasks. If the combined cost values are below the budget, the ant drops the team chain and tries again. The rule that every employee can only work at one
task at the time has to be dropped in order to form an example with such a little number of employees and teams. Every team can only perform one Task at the time.

Table 5. Team Chains of the first Iteration with Team Id(Cost, Skill)

<table>
<thead>
<tr>
<th>Team chain</th>
<th>Task</th>
<th>Team 1</th>
<th>Team 2</th>
<th>Team 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant 1</td>
<td>24</td>
<td>16.5</td>
<td>35</td>
<td>6.3</td>
</tr>
<tr>
<td>Ant 2</td>
<td>34</td>
<td>12.4</td>
<td>14</td>
<td>12.4</td>
</tr>
<tr>
<td>Ant 3</td>
<td>35</td>
<td>6.3</td>
<td>24</td>
<td>16.5</td>
</tr>
</tbody>
</table>

The ants calculate the pheromone values to add for the collected tasks and teams, after forming both the task and the team chain.

The start value of pheromone added to a chain is 0.5 and increases by 20% each iteration. pCost is 0.2 and pSyn is 0.8

Some values are rounded to reduce the number of needed fractional digits. The sum of pheromone values from one chain should be 0.5 except all elements of the chain have the same x or y value.

The calculation of the additional pheromone value for task 1 and team 24 from the task and team chain of the first ant are shown in detail.

Table 7. Calculations of the First Iteration for the Teams

<table>
<thead>
<tr>
<th>TeamID</th>
<th>Syn</th>
<th>synSum</th>
<th>costDiff</th>
<th>costDiffSum</th>
<th>x</th>
<th>xSyn</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>37</td>
<td>0.21</td>
<td>0.16</td>
</tr>
<tr>
<td>35</td>
<td>3</td>
<td>5</td>
<td>10</td>
<td>37</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>45</td>
<td>4</td>
<td>4</td>
<td>10</td>
<td>37</td>
<td>0.21</td>
<td>0.16</td>
</tr>
<tr>
<td>45</td>
<td>3</td>
<td>5</td>
<td>10</td>
<td>37</td>
<td>0.21</td>
<td>0.16</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>37</td>
<td>0.21</td>
<td>0.16</td>
</tr>
</tbody>
</table>

Table 6. Task and Team Chains of the first Ant with Task ID(Skill) and Team ID(Cost, Skill)

<table>
<thead>
<tr>
<th>TaskID(skill)</th>
<th>1(2)</th>
<th>4(2)</th>
<th>2(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TeamID(cost,skill)</td>
<td>24(16.5)</td>
<td>35(6.3)</td>
<td>45(11.2)</td>
</tr>
</tbody>
</table>

\[
xSyn = \max \times pSyn \times \frac{syn}{synSum}
\]

\[
xCost = \max \times pCost \times \frac{costDiff}{costDiffSum}
\]

\[
x = xSyn + xCost
\]

\[
y = \max \times \frac{skillDiff}{skillDiffSum}
\]

\[
0.2 = 0.5 \times 0.8 \times \frac{5}{5+3+2}
\]

\[
0.01 = 0.5 \times 0.2 \times \frac{4}{4+14+19}
\]

\[
0.21 = 0.2 + 0.01
\]

\[
0.17 = 0.5 \times \frac{1}{1+1+1}
\]

The calculations for all other tasks and teams for all ants are displayed in the following tables. One task or team can be used by different ants and can appear multiple times in these tables.
The best average synergy value found in this iteration was 4.33. After calculation these x and y values the pheromone values of the used tasks and teams are updated. The updated team and task repositories contain the following:

**Table 9. Tasks after the First Iteration**

<table>
<thead>
<tr>
<th>Task-ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Input</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Output</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Pheromone</td>
<td>0.67</td>
<td>0.96</td>
<td>0.5</td>
<td>0.67</td>
<td>0.83</td>
<td>0.75</td>
<td>0.5</td>
<td>0.5</td>
<td>0.63</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Table 10. Teams after the First Iteration**

<table>
<thead>
<tr>
<th>Team-ID</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Cost</td>
<td>10</td>
<td>7</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>16</td>
<td>9</td>
<td>12</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>Synergy</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Pheromone</td>
<td>0.5</td>
<td>0.78</td>
<td>0.65</td>
<td>0.5</td>
<td>0.77</td>
<td>0.99</td>
<td>0.5</td>
<td>0.67</td>
<td>0.81</td>
<td>0.63</td>
</tr>
</tbody>
</table>

There are some notable facts after this first iteration. The task 2 with a given input of 1 now has higher pheromone values than most other tasks because the tasks with the starting input are more likely to get picked. Task 2 even got picked twice ranking him first in pheromone value despite the fact, that there are tasks with better skill value. The tasks 5 and 6, both with a perfect skill value of 1, got picked both once but received the biggest pheromone bonus. The task 3 got picked too but didn’t receive pheromones because it has got the worst possible skill value which is 3.

Teams with the same synergy value like 14 and 34 (both 4) can get different values to add to their pheromones due to the weighted importance of synergy and cost value for teams. Team 34 got a bigger bonus than team 14 because it is more budget friendly. Teams picked twice can have a pheromone advantage against teams with better values. With a higher number of iterations this can smooth out or result in a local maximum. A high number of ants and iterations combined with slow increasing pheromone values can make getting stuck in such maxima less likely but trades of run time.

The main purpose of this example was showing the general behavior of the application and the algorithm. With the humble number of employees, tasks, ants and iterations only a raw overview can be provided. This simplified view should help to imagine the processes running this application with about 10 000 employees, 25 000 tasks and 40 ants over 1 000 iterations and with a lower pheromone increase per iteration.

8. Conclusion

One of the main challenges while developing this application was building a data structure to store big numbers of team and provide fast methods to select and update pheromone values. The array tree class was one major approach to solve this problem, because of its focus on small memory usage and reasonable performance. The presorting of team by skill values into different array trees and the management of these array trees via array tree heads was a second big improvement on performance and reliability of the application. To enable the application to run for thousands of iterations one design concept was reducing the runtime of iterative called method with extended services in the process of data initialization.

The shown simplification of the dynamic team formation can partly simulate reality. Most likely a company would store synergy values not for every possible team but for employees who could work together because of similar fields of expertise. The synergy value would more likely be represented by an integer, maybe in percentage, than a double. An integer representation has proven not suitable for this
simulation because with a low number of tasks in a chain and a high number of teams the problem can be solved by only considering perfect teams and by doing so guaranteeing the maximum average synergy value.

References


Carsten Mueller holds a M.Sc. degree from the University of Liechtenstein in Vaduz and a Ph.D. degree from the University of Economics (Department Informatics and Statistics) in Prague. He works since 15+ years as an independent consultant and trainer in the areas of IT process optimization, Object-oriented Modelling and IT Governance. He is currently the technical head of Processes and IT Management at a large airport in Germany and senior lecturer for software engineering, UML and algorithms at the Baden-Wuerttemberg Cooperative State University.

Christian Eichmann researches on swarm intelligence with the focus on Ant Colony Optimization and NP-hard problems. He is currently working for a leading global supplier of technology and service in Germany. At the Baden-Wuerttemberg Cooperative State University he studies Applied Informatics and is a member of a research group.