Solving the Quality-based Software-Selection and Hardware-Mapping Problem with ACO

Samaneh HoseinDoost
s.hoseindoost@eng.ui.ac.ir

Meysam Karimi
meysam.karimi@eng.ui.ac.ir

Shekoufeh Kolahdouz-Rahimi
sh.rahimi@eng.ui.ac.ir

Bahman Zamani
zamani@eng.ui.ac.ir

MDSE research group
Department of Software Engineering
Faculty of Computer Engineering
University of Isfahan, Iran

Abstract

This paper presents a solution for the Quality-based Software-Selection and Hardware-Mapping problem using the ACO algorithm. ACO is one of the most successful swarm intelligence algorithms for solving discrete optimization problems. The evaluation results show that the proposed approach generates correct results for all evaluated test cases. Also, better results in terms of performance and scalability are given in comparison with the ILP and EMFeR approaches.

1 Introduction

The TTC 2018 case [1] is a Searched Based Optimization Problem (SBSE) to provide an optimal mapping of software implementation to hardware components for a given set of user requests. In recent years, SBSE has been applied successfully in the area of model and program transformation [2, 3]. Variety of algorithms exist to solve optimization problems with large or infinite search spaces. In this paper, the Ant Colony Optimization (ACO), one of the most successful swarm intelligence algorithms [4], is applied for solving the TTC case.

The remainder of this paper is structured as follows. In section 2, the concept of ACO is overviewed. Section 3 presents the proposed solution to the TTC case. The evaluation of ACO algorithm in terms of performance, quality and scalability is presented in Section 4. Finally, section 5 concludes the paper.

2 ACO Overview

ACO is one of the most successful swarm intelligence algorithms proposed by M. Dorigo [5]. This algorithm is inspired by the foraging behavior of real ants in nature. In this algorithm ants communicate indirectly with each other through modification of environment. When an individual ant modifies the environment, others will realize and respond to the new modification. Real ants use a chemical trail called pheromone, which left on the ground during their trips to modify the environment. When another ant choosing their path, they tend to

Copyright © by the paper's authors.

In: A. Garcia-Dominguez, G. Hinkel and F. Krikava (eds.): Proceedings of the 11th Transformation Tool Contest, Toulouse, France, 29-06-2018, published at http://ceur-ws.org

choose paths marked by strong pheromone. This helps the ant to choose the best path. After an ant finds food and returns to the nest, it will deposit pheromone again and emphasizes on the correct path. If an ant does not find a food it does not deposit pheromone on the path passed previously. Figure 1 shows an experiment with a real colony of Argentine ants done by Goss et al. [6]. They will choose a shorter route, gradually. When faced with an obstacle, there is an equal chance for an ant to choose a path (the left or right). As the left path is shorter than the right one, the ant eventually deposits a higher level of pheromone. More ants take the left path, when more level of pheromone will be on that path.

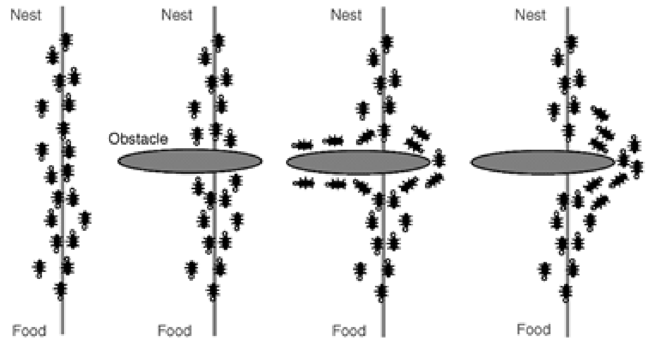

Figure 1: Optimal route between food and nest by real colony of ants

Since, the ants may not have chosen very good paths especially in the early stages, it is essential to gradually withdraw these paths from the candidates of new selections. Therefore, there is a need for a concept called evaporation. The evaporation causes the pheromones to evaporate gradually, in order to omit the inappropriate paths.

Listing 1 presents the generic algorithm of ACO. The parameters are set in the initialization phase of this algorithm. Most parameters of ACO are similar to population based metaheuristic algorithms including Genetic and PSO. The most important parameter belonging to ACO is pheromone initialization.

Listing 1: Generic algorithm of ACO

```
Initialize including the pheromone trails and evaporation rate
Repeat
  FOR each ant Do
    Solution construction using pheromone trails
    Evaporation
    Reinforcement
  Until stopping criteria
OUTPUT: Best solution found or a set of solutions
```

Each artificial ant can be seen as a greedy procedure that constructs a feasible solution in probabilistic fashion. In each step an ant selects a path based on the pheromone and heuristic information. For example in Travel Sales Man (TSP) problem, it can be considered as the distance between cities that each artificial ant cares about to make a better choice. Equation 1 shows the decision transition probabilities. S shows not yet visited solutions in current iteration.

$$P_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}, \quad i \in [1, N], j \in S \tag{1}$$

After each ant made a feasible solution, pheromone trail, which is the most important concept of ant communication, will be updated in two phases. Reinforcement phase, where pheromone trails is updated based on the newly constructed solutions by colony in current iteration. Then, an evaporation phase is applied, where pheromone trail will decrease by a fixed proportion called evaporation rate.

Equation 2 shows updating pheromone. Δ is a positive value added by ants in current iteration and τ_{ij} shows the pheromone trail. Evaporation phase is shown in equation 3, where ρ shows evaporation rate which has been set in the initialization phase.

$$\tau_{ij} = \tau_{ij} + \Delta \tag{2}$$

$$\tau_{ij} = (1 - \rho)\tau_{ij} \tag{3}$$

3 Solution

In order to apply the ACO algorithm to this problem, first a swarm of ants is created by transforming the problem into an ACO specific solution. Following that each ant is evaluated and then deposited pheromone according to the result of evaluation. New solutions are then generated from heuristic information and the amount of pheromone deposited in predecessor iterations. The solutions are evaluated again to make better results. The algorithm is terminated according to the predefined condition and finally the most optimal result will be generated. The solution is implemented as two Java files: `ACOSolver.java` and `Ant.java`. It is available as a Github repository[1]. Additionally, the implementation is provided in Share virtual machine[2].

Listing 2 presents the psudocode of our ACO solver (For more details refer to Appendix A.1). This psudocode has two main parts. In part 1 valid software solutions are generated and in part 2, the solutions are passed to swarm of ants for finding the best valid solutions. The population size and iteration size are predefined parameters that determine the termination of parts 1 and 2, respectively.

In part 1 of the psudocode an empty solution is created and for every requested component of the problem model, a valid software assignment is added to the solution. Each component has an implementation list that one of them should be selected randomly for the assignment. Each implementation has some required components and resources that should be mapped to a valid component and resource aiming at generating better abject values. For mapping required component to a valid component, an implementation is selected randomly and then is checked if a valid assignment for the required component is provided. Another implementation is selected until a valid assignment is produced. Following that it is essential to map valid software assignments to their required resources. To this end, all resources of the model are tested for each valid software assignment. If a resource does not violate the validity of assignment, it will be added to the list of possible resources of assignment. The values of τ, η and ρ are set for each resource of the list and then saved into two dimension matrixes in which columns indicate assignments and rows represent resources. Each cell of the matrixes presents values of τ, η, or p. An invalid hardware solution is recognized, if a solution contains an assignment with no possible resource. Any other solutions are considered as valid software after which possible valid hardware solutions are passed to the ants.

Listing 2: The psudocode of the ACO solver

```
/* Part 1: Generating valid software solutions */
Initialize population_size, τ0, α, β
FOR each population
  IF time has exceeded from maxSolvingTime THEN
    exit
  END IF
  create an empty solution
  FOR each requested components
    create an empty assignment_i
    create a valid component mappings for the assignment_i
    find the possible resources from all of the available resources //the resources that don't violated the required
        property clauses and request constraints
    IF an assignment has no possible resources THEN
      Ignore the solution
    ELSE
      FOR each possible resources_j
```

$$\text{set } \tau_{ij} = \tau_0,\ \eta_{ij} = \frac{1}{objective\ of\ assignment_i} \text{ and } P_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{\forall j}(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}$$

```
      END FOR
    END IF
  END FOR
  Save the solutions // All component mappings of the solutions are valid and their assignments have at least one
      possible resource.
END FOR

/* Part 2: Finding the best valid solution */
Initialize iteration_size
bestSolution is empty
FOR each iteration
  IF time has exceeded from maxSolvingTime THEN
```

[1]https://github.com/Ariyanic/TTC18

[2]http://is.ieis.tue.nl/staff/pvgorp/share/?page=MyVirtualDiskImageBanners#XP-TUe_TTC18-ACO.vdi

```
    exit
  END IF
  FOR each saved solution
    Create an ant, and pass to current solution, possible resources, τ, η and P into it
  END FOR
  Run all ants in parallel
  Update τ and P to new values
  IF bestSolution is empty or the objective of antSolution is better than the objective of bestSolution
    Place antSolution as the bestSolution
  END IF
END FOR
OUTPUT: The bestSolution
```

In part 2, for each saved solution, an ant is created and the solution with its possible resource and the τ, η and ρ matrix are passed to the ant. Following that ants are run for finding the best solution. Additionally, the value of the τ and ρ matrix is updated for running ants in the next iteration. At the end of each iteration, the best of ant solutions is added to an array. Finally, the algorithm is terminated according to the predefined condition, and the last solution of the array is identified as an optimal solution.

Listing 3 shows the psudocode of the `ant.run()` method (For more details refer to Appendix A.2). Each ant selects assignments of the received solution one by one with respect to a number of possible resources, such that assignments with the smallest number of possible resources have higher priority than other assignments. This is because the resource cannot be shared between more than one assignment. If an assignment with a small number of possible resources is investigated later, it is possible that all of its possible resources are allocated by other assignments, and therefore the solution becomes invalid. In this paper the map data structure is used for implementing this mechanism. Each map consists of keys and values. The key factors correspond to the number of possible resources and values relate to the index of the assignment. The map is constructed in the first part of the ACO solver after finding possible resources, and then it is passed to the ant with the τ, η, and ρ parameters. In this step, each ant investigates the existence of possible resources for the selected assignment, which have not been used by other assignments. If a resource is found the ant calculates the probability of selecting any possible resources of the assignment. It then selects one of them using a roulette wheel mechanism based on the probability of selecting the resource. If the Roulette Wheel selects an unused resource, the resource is allocated to the assignment, and the τ and ρ for the selected resource are updated (based on equations 2 and 3). Finally, the updated solution with the best resource mapping is returned by the ant, if the process is terminated successfully for all the assignment of the solution.

Listing 3: The psudocode of the `ant.run()` method

```
Initialize α, β, ρ and Q
Sort the assignments of the solution based on the number of possible resources
FOR each assignments
  IF there is at least one possible resource that has not been used by other assignment
    Select a possible resource_j using Rolette Wheel mechanism that has not been used by other assignment
    Assign resource_j to assignment_i
    Update τ_ij -> τ_ij = τ_ij + Q / (objective of assignment_i)
    Evaporate τ_ij -> τ_ij = (1 − ρ) ∗ τ_ij
    Calculate P_ij = (τ_ij)^α (η_ij)^β / Σ_∀j (τ_ij)^α (η_ij)^β
  END IF
END FOR
OUTPUT: The solution
```

4 Evaluation

In order to evaluate our solution in comparison with the other solutions, all of them are run on a standard Windows 7 PC using an Intel® Core™ i5-2430M with 2.40GHz processor and 4.00 GB RAM. Table 1 shows the results of evaluation w.r.t. their correctness, performance, solution quality and scalability. As explained in the case study description [1]:

- "Correctness denotes that only solutions not violating the minimum requirements of the users are considered valid".

- "The performance of an approach describes how fast a solution can be computed for a given problem". In order to prevent running the benchmarks very long, the timeout is set on 15 minutes.
- "The solution quality quantifies how close the computed solution is to the optimal solution". For this purpose, the objective value of the solutions is measured based on the objective function in the current model. "The objective function is represented as minimization of either a weighted sum or the maximum of all variables"; Therefore the solutions with less objective value are better in terms of quality (except zero points which means the solution has failed in the validity test).
- Finally, "scalability is represented by the size of the largest problem". As shown in Table 1, for each scalability level, the population and iteration size of the ACO algorithm is increased in order to find a better objective for the solution.

The outcome indicates that this approach generates correct results for the evaluated test cases. Also, better results in terms of performance and scalability are given in comparison with the results of the ILP and EMFeR approaches.

Table 1: Evaluation results for the solutions

Scalability Level	Name	Execution Time (Performance) in ms	Best objective of the valid solution (Quality)	Valid	TimeOut
trivial	ACO (pop=1, iter=1)	13	226823.7	TRUE	FALSE
	ILP-direct / external	619 / 445	226823.7 / 226823.7	TRUE / TRUE	FALSE / FALSE
	EMFeR	366	226823.7	TRUE	FALSE
	Simple	1	226823.7	TRUE	FALSE
small	ACO (pop=5, iter=1)	20	34620.2	TRUE	FALSE
	ILP-direct / external	89 / 311	34620.2 / 34620.2	TRUE / TRUE	FALSE / FALSE
	EMFeR	326	34620.2	TRUE	FALSE
	Simple	4	34620.2	TRUE	FALSE
small-many-hw	ACO (pop=8, iter=1)	31	34620.2	TRUE	FALSE
	ILP-direct / external	73 / 325	34620.2 / 34620.2	TRUE / TRUE	FALSE / FALSE
	EMFeR	336	34620.2	TRUE	FALSE
	Simple	6	34620.2	TRUE	FALSE
small-complex-sw	ACO (pop=80, iter=1)	737	974728.2	TRUE	FALSE
	ILP-direct / external	1015 / 2298	968744 / 968744	TRUE / TRUE	FALSE / FALSE
	EMFeR	679197	0	FALSE	FALSE
	Simple	900002	0	FALSE	TRUE
medium	ACO (pop=5000, iter=50)	113005	8747556	TRUE	FALSE
	ILP-direct / external	726894 / 900172	4746869 / 0	TRUE / FALSE	FALSE / TRUE
	EMFeR	GC overhead limit exceeded			
	Simple	Run time error			

5 Conclusion

In the paper the ACO algorithm is used to solve the Quality-based Software-Selection and Hardware-Mapping problem. ACO is a comprehensive algorithm for solving discrete optimization problems. It is a constructive algorithm that in each step a feasible solution for the next step is constructed. Additionally, ACO is a population-based algorithm, in which each ant is a candidate solution in a solution space. The population size can be increased according to the problem. Importantly, each ant can run autonomously and this makes an optimal result in terms of performance.

The scalability, performance and quality of the proposed solution are tested on different benchmarks. The outcome indicates that this approach generates correct results for the evaluated test cases. Also, better results in terms of performance and scalability are given in comparison with the results of the ILP and EMFeR approaches.

References

[1] Sebastian Götz, Johannes Mey, Rene Schöne and Uwe Aßmann, "Quality-based Software-Selection and Hardware-Mapping as Model Transformation Problem," in *Proceedings of the 11th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2018) federation of conferences* (A. Garcia-Dominguez, G. Hinkel, and F. Krikava, eds.), CEUR Workshop Proceedings, CEUR-WS.org, June 2018.

[2] M. Fleck, J. Troya, M. Kessentini, M. Wimmer, and B. Alkhazi, "Model transformation modularization as a many-objective optimization problem," *IEEE Trans. Softw. Eng*, vol. 43, no. 11, pp. 1009–1032, 2017.

[3] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Model transformation by-example: a survey of the first wave," in *Conceptual Modelling and Its Theoretical Foundations*, pp. 197–215, Springer, 2012.

[4] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing*, vol. 8, no. 2, pp. 239–287, 2009.

[5] M. Dorigo, "Optimization, learning and natural algorithms," *PhD Thesis, Politecnico di Milano*, 1992.

[6] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels, "Self-organized shortcuts in the argentine ant," *Naturwissenschaften*, vol. 76, no. 12, pp. 579–581, 1989.

A Appendix: Details of our solution

A.1 ACOSolver.java

```java
package ir.ac.ui.eng;

import de.tudresden.inf.st.mquat.jastadd.model.*;
import de.tudresden.inf.st.mquat.solving.BenchmarkableSolver;
import de.tudresden.inf.st.mquat.solving.Solver;
import de.tudresden.inf.st.mquat.solving.SolverUtils;
import de.tudresden.inf.st.mquat.solving.SolvingException;
import de.tudresden.inf.st.mquat.utils.StopWatch;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.util.*;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.TimeUnit;

/**
 * @author Samaneh Hoseindoost
 * @author Meysan Karimi
 * @author Shekoufeh Kolahdouz-Rahimi
 * @author Bahman Zamani
 */
public class ACOSolver implements BenchmarkableSolver {

  int population_size = 50000;
  int iteration_size = 50;
  private static final Logger logger = LogManager.getLogger(ACOSolver.class);
  private Solution lastSolution;
  private long lastSolvingTime;
  private int solutionCounter;
  private long maxSolvingTime;
  private StopWatch stopWatch;
  private boolean timedOut;
  int numAssignments = 0;

  public ACOSolver() {
    this(Long.MAX_VALUE);
  }

  public ACOSolver(long maxSolvingTime) {
    this.maxSolvingTime = maxSolvingTime;
    reset();
  }

  public ACOSolver setPopulation_size(int population_size) {
    this.population_size = population_size;
    return this;
  }
```

```java

  public ACOSolver setIteration_size(int iteration_size) {
    this.iteration_size = iteration_size;
    return this;
  }

  public Assignment ACOCreateSoftwareAssignment(Request request, Component component, boolean topLevel, int i) {
    Assignment assignment = new Assignment();
    assignment.setRequest(request);
    assignment.setTopLevel(topLevel);
    Implementation implementation = component.getImplementation(i);
    assignment.setImplementation(implementation);
    for (ComponentRequirement requirement : implementation.getComponentRequirementList()) {
      for (Instance instance : requirement.getInstanceList()) {
        int rangeMin = 0;
        int rangeMax = requirement.getComponentRef().getRef().getImplementationList().getNumChild();
        Assignment ass = null;
        do{
          int randomNum = ThreadLocalRandom.current().nextInt(rangeMin, rangeMax);
          ass = ACOCreateSoftwareAssignment(request, requirement.getComponentRef().getRef(), false, randomNum);
        }while(!ass.isSoftwareValid());
        assignment.addComponentMapping(new ComponentMapping(instance, ass));
      }
    }
    for (Instance instance : implementation.getResourceRequirement().getInstanceList()) {
      assignment.setResourceMapping(new ResourceMapping(instance, null, new de.tudresden.inf.st.mquat.jastadd.model.
           List<>()));
    }
    return assignment;
  }

  private static void assignResource(Assignment assignment, Resource resource) {
    Implementation impl = assignment.getImplementation();
    ResourceMapping mapping = new ResourceMapping(impl.getResourceRequirement().getInstance(0), resource,
        new de.tudresden.inf.st.mquat.jastadd.model.List<>());
    SolverUtils.populateResourceMapping(mapping, impl.getResourceRequirement(), resource);
    assignment.setResourceMapping(mapping);
  }

  @Override
  public Solution solve(Root model) throws SolvingException {
    reset();
    if (model.getNumRequest() == 0) {
      return Solution.emptySolutionOf(model);
    }
    int numSoftwareSolutions = 0;
    int numTotalSoftwareSolutions = 0;
    stopWatch = StopWatch.start();
    List<Solution> solutions = new ArrayList<>();
    List<Solution> currentSolutions = new ArrayList<>();
    List<List<Set<Resource>>> currentPossibleResources = new ArrayList<>();
    List<List<List<Double>>> currentTau = new ArrayList<>(); // Pheromone for each resources
    List<List<List<Double>>> currentEta = new ArrayList<>(); // Objective for each resources
    List<List<Double>> currentDenominatorP = new ArrayList<>();
    List<List<List<Double>>> currentNumeratorP = new ArrayList<>();
  List<Map<Integer, List<Integer>>> currentSort = new ArrayList<>();

/* Part 1: Generating valid software solutions */
    for (int pop = 0; pop < population_size; pop++) {
      if (stopWatch.time(TimeUnit.MILLISECONDS) > maxSolvingTime) {
        timedOut = true;
        break;
      }
    Solution currentSolution = new Solution();
    currentSolution.setModel(model);
    de.tudresden.inf.st.mquat.jastadd.model.List<Request> requests = model.getRequests();
    for(Request request: requests){
      int rangeMin = 0;
      int rangeMax = request.getTarget().getRef().getImplementationList().getNumChild();
```

```
      Assignment ass = null;
      do{
        int randomNum = ThreadLocalRandom.current().nextInt(rangeMin, rangeMax);
        ass = ACOCreateSoftwareAssignment(request, request.getTarget().getRef(), true, randomNum);
      }while(!ass.isSoftwareValid());
      currentSolution.addAssignment(ass);
    }

    de.tudresden.inf.st.mquat.jastadd.model.List<Resource> resources = model.getHardwareModel().getResources();
    numTotalSoftwareSolutions++;
    List<Assignment> assignments = currentSolution.allAssignments();
    List<Set<Resource>> possibleResources = new ArrayList<>(assignments.size());
    boolean isHardwareValid = true;
    double tau0 = 1;
    List<List<Double>> tau = new ArrayList<>(); // Pheromone for each resources
    List<List<Double>> eta = new ArrayList<>(); // Objective
    double alpha = 1;
    double beta = 1;
    List<List<Double>> numeratorP = new ArrayList<>();
    List<Double> denominatorP = new ArrayList<>();
    Map<Integer, List<Integer>> SortIndexByPossibleResource = new HashMap<>();
    int index = 0;

    for (Assignment assignment : assignments) {
      Set<Resource> resourceList = new HashSet<>();
      List<Double> taui = new ArrayList<>();
      List<Double> etai = new ArrayList<>();
      List<Double> numeratorPi = new ArrayList<>();
      double sum = 0;
      int i = 0;
        for (Resource resource : resources) {
          assignResource(assignment, resource);
          if (assignment.isValid()) {
            resourceList.add(resource);
            taui.add(tau0); // Pheromone on antSolution.allAssignments().allResources;
            etai.add(1 / assignment.computeObjective());
            numeratorPi.add((taui.get(i) * alpha) + (etai.get(i) * beta));
            sum += numeratorPi.get(i);
            i++;
          }
        }
        if(i == 0){
          isHardwareValid = false;
          break;
        }
        possibleResources.add(resourceList);
        tau.add(taui);
        eta.add(etai);
        numeratorP.add(numeratorPi);
        denominatorP.add(sum);
        SortIndexByPossibleResource.computeIfAbsent(i, k -> new ArrayList<>()).add(index);
        index++;
      }
      if(isHardwareValid == true){
        numSoftwareSolutions++;
        Solution clone = currentSolution.deepCopy();
        currentSolutions.add(clone);
        currentPossibleResources.add(possibleResources);
        currentTau.add(tau);
        currentEta.add(eta);
        currentNumeratorP.add(numeratorP);
        currentDenominatorP.add(denominatorP);
        currentSort.add(SortIndexByPossibleResource);
      }
    }

/* Part 2: Finding the best valid solution */
    for (int iteration = 0; iteration < iteration_size; iteration++) {
      if (stopWatch.time(TimeUnit.MILLISECONDS) > maxSolvingTime) {
```

```
        timedOut = true;
        break;
      }

    List<Ant> population = new ArrayList<>();
    for (int i = 0; i < currentSolutions.size(); i++) {
      Ant ant = new Ant(i, currentSolutions.get(i), currentPossibleResources.get(i), currentEta.get(i), currentSort.get(i), numAssignments);
      population.add(ant);
    }

    Parallel.ForEach(population, new LoopBody<Ant>() {
      @Override
      public void run(Ant ant) {
        int ant_Number = ant.id;
        List<List<Double>> tau = currentTau.get(ant_Number); ///////////// BEFORE RUN
        List<List<Double>> numeratorP = currentNumeratorP.get(ant_Number);
        List<Double> denominatorP = currentDenominatorP.get(ant_Number);

        Solution antSolution = ant.run(tau, numeratorP, denominatorP);  //////////// Tau CHANGE AFTER RUN.

        if (antSolution != null) {
          currentTau.set(ant_Number, tau);  ///////////// AFTER RUN
          currentNumeratorP.set(ant_Number, numeratorP);
          currentDenominatorP.set(ant_Number, denominatorP);
          numAssignments += ant.numAssignments;
          solutionCounter++;
          if (solutions.isEmpty() || antSolution.computeObjective() < solutions.get(solutions.size() - 1).computeObjective()) {
            Solution clone = antSolution.deepCopy();
            solutions.add(clone);
            logger.info("found a better solution with an objective of {}.",antSolution.computeObjective());
          }
        }
      }
    });
    }

    logger.info("Number of total software solutions: {}", numTotalSoftwareSolutions);
    logger.info("Number of iterated software solutions: {}", numSoftwareSolutions);
    logger.info("Number of iterated solutions: {}", numAssignments);
    logger.info("Number of correct solutions: {}", solutionCounter);

    if (solutions.size() > 0) {
      lastSolution = solutions.get(solutions.size() - 1);
    } else {
      lastSolution = Solution.emptySolutionOf(model);
      logger.warn("Found no solution!");
    }

    lastSolvingTime = stopWatch.time(TimeUnit.MILLISECONDS);
    return lastSolution;
  }

  private void reset() {
    this.lastSolution = null;
    this.solutionCounter = 0;
    this.lastSolvingTime = 0;
    this.timedOut = false;
  }

  @Override
  public String getName() {
    return "aco";
  }

  @Override
  public long getLastSolvingTime() {
    return lastSolvingTime;
```

```java
  }

  @Override
  public double getLastObjective() {
    if (lastSolution != null) {
      return lastSolution.computeObjective();
    } else {
      // TODO throw exception or do something reasonable
      return 0d;
    }
  }

  @Override
  public Solver setTimeout(long timeoutValue, TimeUnit timeoutUnit) {
    this.maxSolvingTime = timeoutUnit.toMillis(timeoutValue);
    return this;
  }

  @Override
  public boolean hadTimeout() {
    return this.timedOut;
  }
}
```

A.2 Ant.java

```java
package ir.ac.ui.eng;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Set;
import java.util.Stack;

import de.tudresden.inf.st.mquat.jastadd.model.Assignment;
import de.tudresden.inf.st.mquat.jastadd.model.Implementation;
import de.tudresden.inf.st.mquat.jastadd.model.Resource;
import de.tudresden.inf.st.mquat.jastadd.model.ResourceMapping;
import de.tudresden.inf.st.mquat.jastadd.model.Solution;
import de.tudresden.inf.st.mquat.solving.SolverUtils;

/**
 * @author Samaneh Hoseindoost
 * @author Meysan Karimi
 * @author Shekoufeh Kolahdouz-Rahimi
 * @author Bahman Zamani
 */
public class Ant {

  int id;
  Solution currentSolution;
  List<Set<Resource>> possibleResources;
  List<List<Double>> eta;
  Map<Integer, List<Integer>> Sort;
  int numAssignments;

  Ant(int i, Solution solu, List<Set<Resource>> pr, List<List<Double>> et, Map<Integer, List<Integer>> sort, int num
      ) {
    id = i;
    currentSolution = solu;
    possibleResources = pr;
    eta = et;
    Sort = sort;
    numAssignments = num;
  }

```

```java
private static void assignResource(Assignment assignment, Resource resource) {
  Implementation impl = assignment.getImplementation();

  ResourceMapping mapping = new ResourceMapping(impl.getResourceRequirement().getInstance(0), resource,
      new de.tudresden.inf.st.mquat.jastadd.model.List<>());
  SolverUtils.populateResourceMapping(mapping, impl.getResourceRequirement(), resource);
  assignment.setResourceMapping(mapping);
}

public int RoleteWhileSelection(double[] c) {

  double rangeMin = 0.0f;
  double rangeMax = c[c.length-1];
  Random r = new Random();
  double createdRanNum = rangeMin + (rangeMax - rangeMin) * r.nextDouble();
  int i;
  for (i = 0; i < c.length-1; i++) {
    if (createdRanNum <= c[i + 1])
      break;
  }
  return i;
}

public Solution run(List<List<Double>> tau, List<List<Double>> numeratorP, List<Double> denominatorP) {

  List<Assignment> assignments = currentSolution.allAssignments();
  Stack<Resource> usedResources = new Stack<>();
  double alpha = 1;
  double beta = 1;
  double rho = 0.1; // Evaporation rate
  double Q = 2;
  List<Integer> keys = new ArrayList<Integer>(Sort.keySet());
  Collections.sort(keys); // "keys" are number of possible resources & "values" are index of the assignments

  for (Integer key: keys) {

    int siz = Sort.get(key).size();
    for (int i = 0; i < siz; i++) {
      int index = Sort.get(key).get(i);
      Assignment assignment = assignments.get(index);
      List<Resource> resources = new ArrayList<Resource>(possibleResources.get(index));

      int remove = 0;
      for (Iterator<Resource> resIt = resources.iterator(); resIt.hasNext();) {
        Resource resource = resIt.next();
        if (usedResources.contains(resource)) {
          remove++;
        }
      }
      if (resources.size() == remove) {
        return null;
      }

      List<Double> numerator = numeratorP.get(index);
      double denominator = denominatorP.get(index);
      int size = resources.size();
      double[] p = new double[size];
      double[] c = new double[size + 1];
      c[0] = 0;
      for(int j=0; j<size; j++ ){
        p[j] = numerator.get(j)/denominator;
        c[j + 1] = c[j] + p[j];
      }

      int select = RoleteWhileSelection(c);
      Resource resource = resources.get(select);
      while (usedResources.contains(resource)) {
        select = RoleteWhileSelection(c);
        resource = resources.get(select);
```

```
        }

        assignResource(assignment, resource);
        usedResources.push(resource);
        denominatorP.set(index, denominatorP.get(index) - numeratorP.get(index).get(select));

        List<Double> taui = tau.get(index);

        // Upadte Pheromone -> tau[index][select] = tau[index][select] + (Q /assignment.computeObjective());
        taui.set(select, taui.get(select) + (Q / assignment.computeObjective()));

     // Evaporation on Pheromone of antSolution.allAssignments();
        taui.set(select, (1 - rho) * taui.get(select)); // tau[index][select] = (1 - rho) * tau[index][select];
        tau.set(index, taui);

        // p[index][select] = (tau[index][select] * alpha) + (eta[index][select] * beta);
        List<Double> sIndex = numeratorP.get(index);
        sIndex.set(select, (tau.get(index).get(select) * alpha) + (eta.get(index).get(select) * beta));
        numeratorP.set(index, sIndex);
        denominatorP.set(index, denominatorP.get(index) + numeratorP.get(index).get(select));
      }
    }

    numAssignments++;
    return currentSolution;
  }
}
```