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
University of Isfahan, Iran

Abstract

The optimal mapping from software implementations to hardware components for a given set of user requests is a challenging topic which is presented in TTC 18. This paper presents a solution to this case using the ACO algorithm.

1 Introduction

The TTC 2018 case 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 [Fleck17], [Kappel12]. 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, is applied for solving the TTC case.

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, which is the main focus of 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 [Dorigo92]. 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 choose, paths marked by strong pheromone. This helps the ant to choose the best path. After an ant finding a 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. [Goss89]. 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.

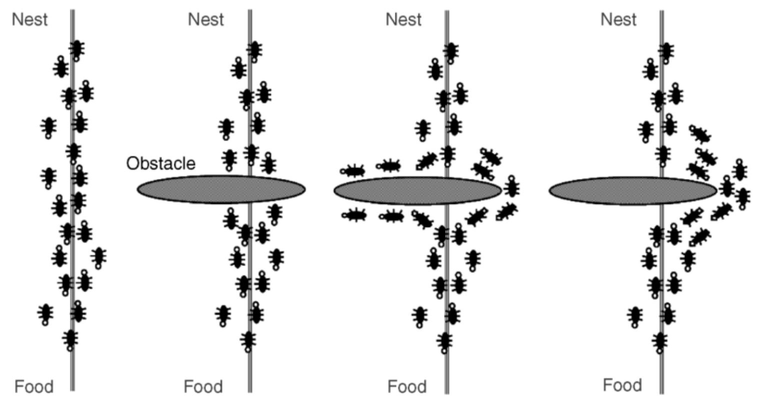


Figure 1: Optimal Route Between Food And Nest By Real Colony of Ants

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.

```
Initialize including the pheromone trails, 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
```

Figure 2: Generic Algorithm Of ACO

Figure 2 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. However, important parameter which belongs to ACO, is pheromone initialization.

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.

Equation 1:
$$P_{ij} = \begin{cases} \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}} \end{cases} \quad , i \in [1, n], j \in S$$

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:
$$\tau_{ij} = \tau_{ij} + \Delta$$

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

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.

3 Solving The Quality-Based Software-Selection And Hardware-Mapping As Model Transformation Problem With ACO

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 best optimal result will be generated. The solution is available as a Github repository[1]. Additionally, the implementation is provided in Share virtual machine[2].

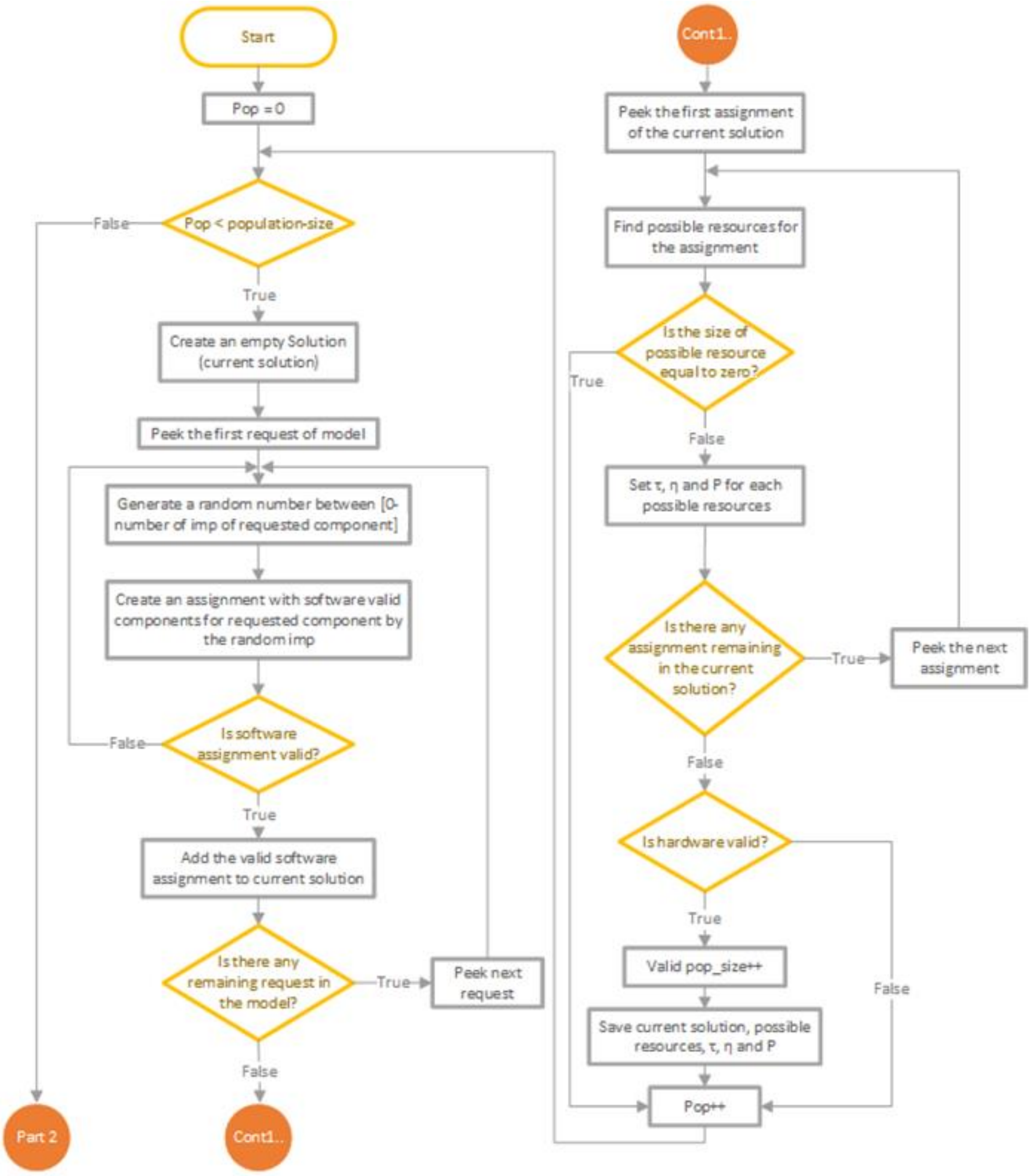
Fig. 3 presents the flowchart of the ACO Solver. This flowchart has two main parts. In part 1 valid software solutions are generated and in part 2, the solutions are passed to swarm of ants for finding the best valid solutions. The population size and iteration size are predefined parameters that determine the termination of parts 1 and 2, respectively.

In part 1 of the flowchart 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 p 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.

In part 2, for each saved solution, an ant is created and the solution with its possible resource and the τ, η and p matrix are passed to the ant. Following that ants are run for finding the best solution. Additionally, the value of the τ and p 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.

Fig. 4 shows the flowchart of an ant.run() method. 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 p 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 p 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.

[1] https://github.com/Ariyanic/TTC18
[2] http://is.ieis.tue.nl/staff/pvgorp/share/?page=MyVirtualDiskImageBanners#XP-TUe_TTC18-ACO.vdi



Start
Pop = 0
Pop < population-size
False
True
Create an empty Solution (current solution)
Peek the first request of model
Generate a random number between [0-number of imp of requested component]
Create an assignment with software valid components for requested component by the random imp
Is software assignment valid?
False
True
Add the valid software assignment to current solution
Is there any remaining request in the model?
True
Peek next request
False
Part 2
Cont1..
Cont1..
Peek the first assignment of the current solution
Find possible resources for the assignment
Is the size of possible resource equal to zero?
True
False
Set τ, η and P for each possible resources
Is there any assignment remaining in the current solution?
True
Peek the next assignment
False
Is hardware valid?
True
False
Valid pop_size++
Save current solution, possible resources, τ, η and P
Pop++

Figure 3: The Flowchart of the ACO Solver – Part 1

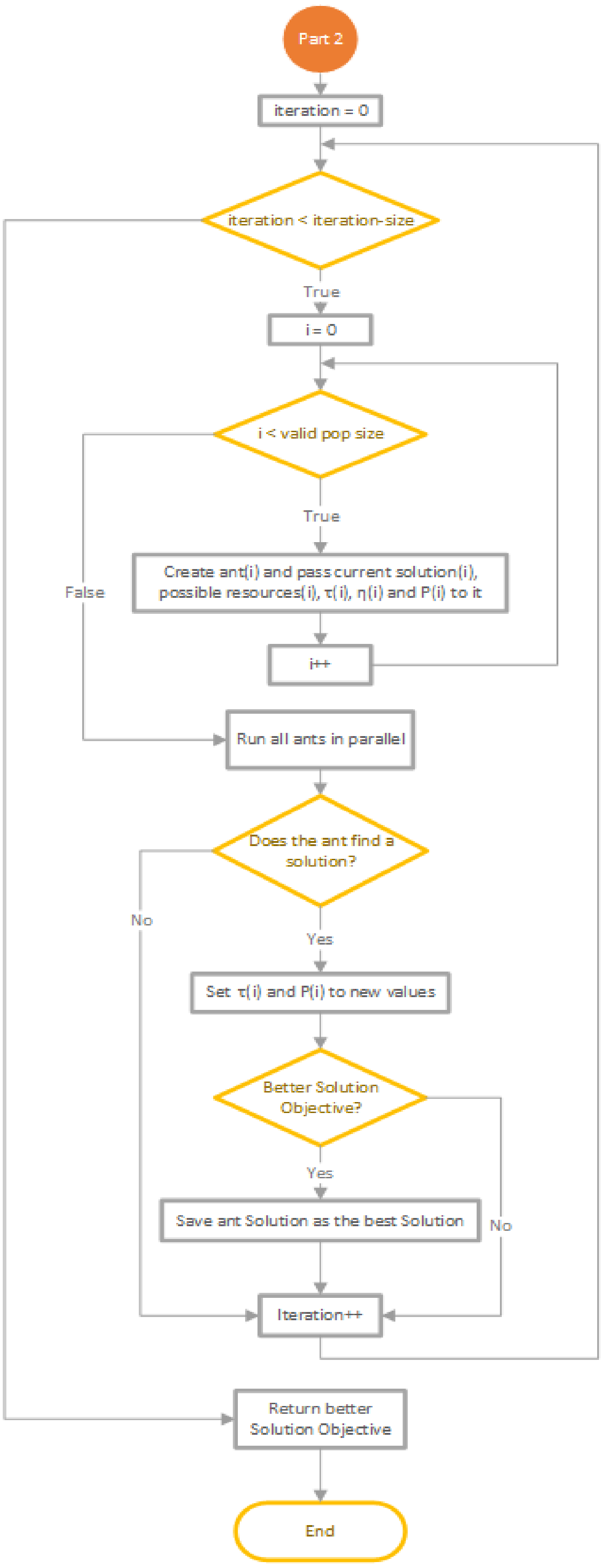

Part 2
iteration = 0
iteration < iteration-size
True
i = 0
i < valid pop size
True
Create ant(i) and pass current solution(i), possible resources(i), τ(i), η(i) and P(i) to it
False
i++
Run all ants in parallel
Does the ant find a solution?
No
Yes
Set τ(i) and P(i) to new values
Better Solution Objective?
Yes
Save ant Solution as the best Solution
No
Iteration++
Return better Solution Objective
End

Figure 3: The Flowchart of the ACO Solver – Part 2

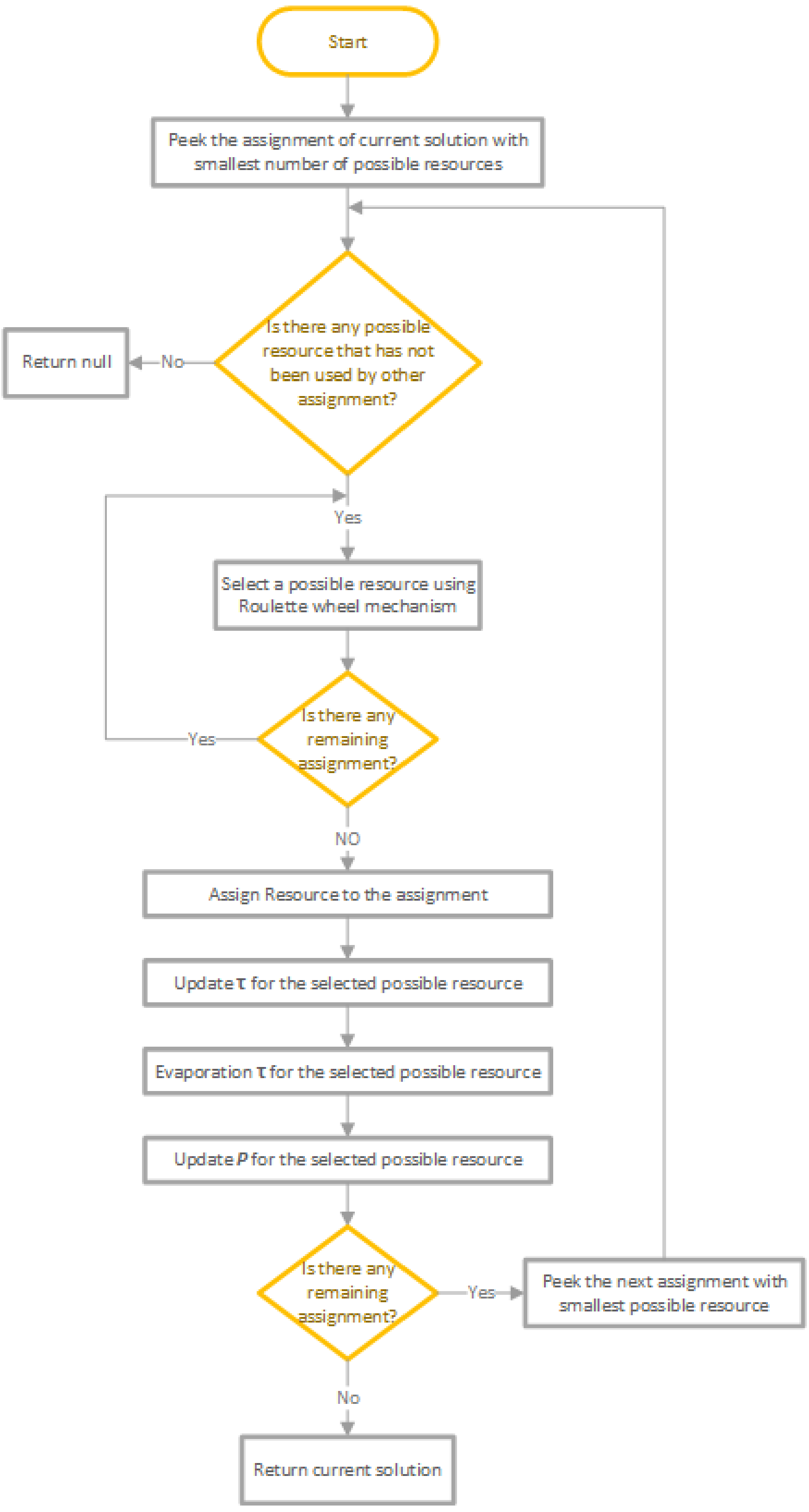

Start
Peek the assignment of current solution with smallest number of possible resources
Is there any possible resource that has not been used by other assignment?
No
Return null
Yes
Select a possible resource using Roulette wheel mechanism
Is there any remaining assignment?
Yes
NO
Assign Resource to the assignment
Update τ for the selected possible resource
Evaporation τ for the selected possible resource
Update P for the selected possible resource
Is there any remaining assignment?
Yes
Peek the next assignment with smallest possible resource
No
Return current solution

Figure 4: The Flowchart of An Ant.Run() Method

4 Evaluation

The performance, quality and scalability of the proposed solution in this paper is evaluated for five benchmarks provided in the case description. Table 1 shows the results of evaluation, in which population and iteration size increase in order to find a better objective for the solution. All tests were carried out on a standard Windows 7 PC using an Intel®Core™ i5-2430M with 2.40GHz processor and 4.00 GB RAM.

Table 1: Evaluation results for our solution includes scalability level, performance and quality

Scalability Level	Population Size	Iteration Size	Execution Time (Performance)	Best objective of the valid solution (Quality)
trivial	1	1	8 ms	226823.67
small	8	1	17 ms	34620.2
small-many-hw	8	5	25 ms	34620.2
small-complex-sw	80	1	991 ms	969897.3
medium	50000	100	222961 ms	8658117.329

5 Conclusion

In the paper the ACO algorithm is used to solve the Quality-based Software-Selection and Hardware-Mapping problem. 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. Additionally, better results in terms of performance are given in comparison with the results of the ILP algorithm in the case description.

6 References

[Fleck17] M. Fleck, J. Troya, M. Kessentini, M. Wimmer and B. Alkhazi. Model transformation modularization as a many-objective optimization problem. IEEE Transactions on Software Engineering, 43(11): 1009-1032, November 2017.

[Kappel12] 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, 197-215, Springer, 2012.

[Dorigo92] M. Dorigo. Optimization learning and natural algorithms, Ph.D. Thesis, Politecnico di Milano, 1992.

[Goss89] S. Goss, S. Aron, J.L. Deneubourg and J.M. Pasteels. Self-organized shortcuts in the Argentine ant. Naturwissenschaften, 76(12): 579-581, December 1989.